Operating system deitel pdf download pc windows 10 64-bit

Showing 1-30 Start your review of Operating Systems Apr 19, 2018 Amin rated it it was ok Want to read Peter Manuel rated it really liked it Nov 02, 2016 Longworth rated it it was amazing Sep 12, 2019 Jan rated it really liked it Oct 18, 2007 Kousik rated it did not like it May 29, 2020 Jojon rated it really liked it Feb 13, 2016 Richard Chua rated it really liked it Nov 25, 2012 Sarabjeet rated it really liked it Sep 18, 2014 Kundan Kumar rated it it was amazing May 25, 2015 Sanjay rated it really liked it Sep 02, 2020 Jestin Joy rated it really liked it Sep 28, 2011 Abhishek Kumar rated it really liked it Aug 15, 2014 Jonathan rated it it was ok Oct 23, 2008 Hải Phạm rated it it was amazing Oct 05, 2020 Lieb rated it it was ok Jul 17, 2008 Mark Wang rated it really liked it May 12, 2014 Sankalp Gupta rated it it was amazing Sep 16, 2013 Ivan Lance rated it really liked it Aug 25, 2016 Operating Systems Third Edition H. M. Deitel Deitel & Associates, Inc. P. J. Deitel Deitel & Associates, Inc. D. R. Choffnes Deitel & Associates, Inc. Upper Saddle River, NJ 07458 Deitel Books. Cyber Classrooms, Complete Training Courses and Web-Based Training Courses published by Prentice Hall Computer Science Series Operating Systems, 3/E How To Program Series Advanced Java™ 2 Platform How to Program C How to Program, 4/E C++ How to Program, 4/E C# ® How to Program e-Business and e-Commerce How to Program Internet and World Wide Web How to Program, 3/E Java™ How to Program, 5/E Perl How to Program Python How to Program Visual Basic® 6 How to Program Visual Basic® .NET How to Program, 2/E Visual C++® .NET How to Program Wireless Internet & Mobile Business How to Program XML How to Program Simply Series Simply C#®:An Application-Driven Tutorial Approach Simply Java™ Programming: An Application-Driven Tutorial Approach Simply Visual Basic® .NET: An ApplicationDriven Tutorial Approach (Visual Studio .NET2002 Edition) Simply Visual Basic® .NET: An ApplicationDriven Tutorial Approach (Visual Studio .NET2003 Edition) CS1 Programming Series Java™ Software Design .NET How to Program Series C#® How to Program Visual Basic® .NET How to Program, 2/E Visual C++® .NET How to Program Visual Studio® Series C#® How to Program Getting Started with Microsoft® Visual C++® 6 with an Introduction to MFC Simply C#®:An Application-Driven Tutorial Approach Simply Visual Basic® .NET: An ApplicationDriven Tutorial Approach (Visual Studio .NET2002 Edition) Simply Visual Basic® .NET: An ApplicationDriven Tutorial Approach (Visual Studio .NET 2003 Edition) Visual Basic® 6 How to Program Visual Basic® .NET How to Program, 2/E Visual C++® .NET How to Program Deitel® Developer Series Java ™ Web Services for Experienced Programmers Web Services A Technical Introduction For Managers Series e-Business and e-Commerce for Managers Interactive Web-Based Training Series Premium CourseCompass Version of Visual Basic® .NET Multimedia Cyber Classroom, 2/E Premium CourseCompass Version of Java™ 2 Multimedia Cyber Classroom, 5/E Premium CourseCompass Version of C++ Multimedia Cyber Classroom, 4/E The Complete Training Course Series The Complete C++ Training Course, 4/E The Complete C#® Training Course The Complete e-Business and e-Commerce Programming Training Course The Complete Internet and World Wide Web Programming Training Course, 2/E The Complete Java™ 2 Training Course, 5/E The Complete Perl Training Course The Complete Python Training Course The Complete Visual Basic® 6 Training Course The Complete Visual Basic® .NET Training Course, 2/E The Complete Wireless Internet & Mobile Business Programming Training Course The Complete XML Programming Training Course Interactive Multimedia Cyber Classroom Series C++ Multimedia Cyber Classroom, 4/E C#® Multimedia Cyber Classroom e-Business and e-Commerce Multimedia Cyber Classroom Internet and World Wide Web Multimedia Cyber Classroom, 2/E Javam2 Multimedia Cyber Classroom, 5/E Perl Multimedia Cyber Classroom Python Multimedia Cyber Classroom Visual Basic® 6 Multimedia Cyber Classroom Visual Basic® .NET Multimedia Cyber Classroom, 2/E Wireless Internet & Mobile Business Programming Multimedia Cyber Classroom XML Multimedia Cyber Classroom To follow the Deitel publishing program, please subscribe to the free Deitel® Buzz Online e-mail newsletter at: www.deitel.com/newsletter/subscribe.html To communicate with the authors, send e-mail to: For information on Deitel instructor-led, corporate training seminars offered worldwide visit: www.deitel.com For continuing updates on Prentice Hall and Deitel publications visit: www.deitel.com, www.prenhall.com/deitel or www.InformIT.com/deitel Vice President and Editorial Director, ECS: Marcia J. Horton Senior Acquisitions Editor: Kate Hargett Associate Editor: Jennifer Cappello Assistant Editor: Sarah Parker Editorial Assistant: Michael Giacobbe Vice President and Director of Production and Manufacturing, ESM: David W. Riccardi Executive Managing Editor: Vince O'Brien Managing Editor: Tom Manshreck Production Editors: Chirag Thakkar, John F. Loveli Production Editor, Media: Bob Engelhardt Production Assistant: Daniela Petrilli Director of Creative Services: Paul Belfanti A/V Production Editor: Xiaohong Zhu Art Studio: Artworks, York, PA Creative Director: Carole Anson Art Director: Geoffrey Cassar Cover Design: Harvey M. Deitel, David R. Choffnes, Suzanne Behnke Interior Design: Harvey M. Deitel, David R. Choffnes, Geoffrey Cassar, Anne DeMarinis Manufacturing Manager: Trudy Pisciotti Manufacturing Buyer: Lisa McDowell Marketing Manager: Pamela Hersperger Marketing Assistant: Barrie Reinhold

 Printed in the United States of America 10 9 8 7 6 5 4 3 2 1 ISBN 0-13-124696-8 Pearson Education Ltd., London Pearson Education Australia Pty. Ltd., Sydney Pearson Education Singapore, Pte. Ltd. Pearson Education North Asia Ltd., Hong Kong Pearson Education Canada, Inc., Toronto Pearson Educacion de Mexico, S.A. de CV. Pearson Education-Japan, Tokyo Pearson Education Malaysia, Pte. Ltd. Pearson Education, Inc., Upper Saddle River, New Jersey In memory of Edsger W. Dijkstra. Harvey M. Deitel Paul J. Deitel David R. Choffnes To my grandparents, Beverly and Jay Peddy: For your love, generosity and inspiration. David Contents List of Illustrations xxvii

Preface Live in fragments no longer. Only connect. Edward Morgan Forster Welcome to the world of operating systems. This text is intended primarily for use in the one-semester and two-semester operating systems courses (as defined in the most recent ACM/IEEE curriculum) that universities offer to juniors, seniors and graduate students in computer science. Operating systems designers and systems programmers will find the text useful as a reference as well. The text features extensive case studies on two of today's most important operating systems—Linux (94 pages) and Microsoft® Windows® XP (106 pages) —that represent two different operating system design paradigms—free, open-source development and licensed, corporate development, respectively. The Linux case study follows the development of the kernel through version 2.6. The Windows XP case study highlights the internals of the current version of the most widely used personal computer operating system. These case studies enable you to compare and contrast the design and implementation philosophies used in realworld operating systems. Both Linux and Windows XP are massive, complex operating systems containing millions of lines of source code. We survey the major components of each operating system. The case studies present issues in personal computer, workstation, multiprocessor, distributed and embedded environments, including a detailed discussion on why Linux and other UNIX-like operating systems have become prominent in the open-source and open-systems philosophies of major corporations. This preface introduces our teaching approach in Operating Systems, 3/e and the book's key content and design elements. We also discuss the ancillary support available with the text. The section entitled "Tour of the Book" overviews the rich coverage of operating systems this book provides. Operating Systems, 3/e, was reviewed by a team of distinguished academics and industry professionals; their names and affiliations are listed in the Acknowledgements section. xxxviii Preface As you read this book, if you have any questions, please send us an e-mail at ; we will respond promptly. Please visit our Web site. www.deitel.com, regularly and sign up for the Deitel Buzz Online e-mail newsletter at www.deitel.com/newsletter/subscribe.html. We use the Web site and the newsletter to keep our readers current on all Deitel publications and services. The site www.deitel.com/books/os3e is dedicated to Operating Systems, 3/e. At that site, we will post various ancillary materials for students and instructors, including dependency charts, content updates, errata, suggested student research projects (term projects, directed studies and theses), sample syllabi and the bibliography of research materials from the prior edition of the book. Prentice Hall provides a Companion Website for the book at www.prenhall.com/deitel. Book Design Operating Systems, 3/e features a completely new, two-color design inspired by the work of Leonardo DaVinci.The two-color design is visually appealing and allows us to use color to improve the pedagogy of the book. For example, defining occurrences of key terms appear in bold and color so they stand out to you. Operating Systems, 3/e Features Operating Systems, 3/e includes extensive new content. We also revised and updated much of the material from the second edition. The focus on current technologies and issues in distributed computing makes this book unique from its competition. Several new sections have been added to address embedded, real-time and distributed systems. Some key features of this edition are: • Conforms to all core requirements of the ACM/IEEE CC2001 Operating Systems course requirements. • Discusses all the CC2001 elective operating systems topics except shell scripting. • Provides an updated hardware introduction that includes leading-edge technologies and their impact on operating systems design. • Presents process, thread, memory and disk management solutions that reflect the needs of current applications. • Supplements the extensive coverage of general-purpose systems with concepts relevant to real-time, embedded and superscalar architectures. • Highlights key evaluation techniques that enable effective comparative analyses of operating system components. • Includes a richer treatment of networking concepts. • Enhances the security treatment to include current trends in authentication mechanisms, security protocols, anti-virus research, access control methods and wireless security. Teaching Approach xxxix • Enhances the distributed computing coverage to acknowledge the tremendous influence of the Internet and the World Wide Web on computing and operating systems. • Provides details of the ubiquitous Intel architecture. • Provides many diagrams, tables, working code examples, pseudocode examples and algorithms. • Includes a new chapter on threads. • Pseudocode is presented in Java-like syntax to capitalize on C/C++/Java literacy—virtually all computer-science students know one or more of these languages. • Provides multithreading treatments in both pseudocode and Java that demonstrate issues in concurrent programming—enabling instructors to cover the material the way they prefer. The Java treatment is new to this edition and is optional. Visit java.sun.com/j2se/downloads.html to obtain the latest version of Java. The download page contains a link to the installation instructions. • Enhances the multiprocessor management treatment. • Provides new sections on thread scheduling and real-time scheduling. • Includes a RAID discussion. • Provides a case study on UNIX processes. • Includes up-to-the-minute sections on memory-management and diskscheduling strategies. • Covers the important topic of I/O systems in many chapters—most notably Chapters 2, 12, 13 and the case study chapters (20 and 21). • Provides 730 self-review questions and answers (approximately two per section) for immediate feedback. • Includes extensive research with citations listed in the Works Cited section at the end of each chapter. Teaching Approach This book is divided into eight parts, each containing a set of related chapters. The parts are: 1. Introduction to Hardware, Software and Operating Systems 2. Processes and Threads 3. Real and Virtual Memory 4. Secondary Storage, Files and Databases 5. Performance, Processors and Multiprocessor Management 6. Networking and Distributed Computing xl Preface 7. Security 8. Case Studies The book's pedagogic features are described below. Quotations Each chapter begins with quotations—some are humorous, some are philosophical and some offer interesting insights. Many readers have told us that they enjoy relating the quotations to the chapter material. You might appreciate some of the quotations more after reading the chapters. Objectives Next, objectives tell you what to expect and give you an opportunity, after reading the chapter, to determine whether you have met these objectives. Outline The chapter outline helps you approach the material in a top-down fashion, so you can anticipate the concepts that will be presented in each chapter and set a comfortable learning pace. Sections and Self-Review Exercises Each chapter contains small sections that address important operating systems concepts. Most sections end with two self-review exercises and answers. These exercises enable you to test your knowledge, get immediate feedback and gauge your understanding of the material. These exercises also help prepare you for the end-of-chapter exercises and for quizzes and exams. Some of the self-review exercises cannot be answered only from the material presented in their corresponding sections; these are additional teaching and learning opportunities. Key Terms The defining occurrence of each term appears in bold and color. In addition, all chapters include a Key Terms section containing the terms defined in the chapter and their definitions (1800+ key terms in the text). A cumulative alphabetized glossary appears at the end of the book. The Key Terms sections and the Glossary are wonderful pedagogic devices for review and reference. Figures The text contains over 300 charts, diagrams, examples and illustrations that support the concepts presented in the text. Web Resources Each chapter contains Web resources that direct you to sites where you can locate valuable additional research materials. Feature Summary Detailed end-of-chapter summary sections help you review the key concepts presented in each chapter. Exercises, Suggested Projects and Suggested Simulations Each chapter includes many exercises that vary in difficulty from review of basic operating systems principles to complex reasoning and research projects (900+ exercises in the text). Many OS instructors like to assign term projects, so we have included Suggested Projects and Suggested Simulations sections at the end of each chapter's exercises. Recommended Readings Each chapter lists recommended books and articles, and provides a brief review of the most important sources for the chapter content so you can do additional research on your own. Works Cited This book required an extraordinary research effort. The entire book is thoroughly cited (2300+ citations). Each citation appears as a superscript in the text and corresponds to an entry in the Works Cited section at the end of the chapter. Many of these citations are to Web sites. Prior editions of this text used only standard book and literature citations; those books and papers were often difficult for readers to locate for further research. Now you can access these citations directly via the Web. Also, it is easy to use search engines to locate additional articles on subjects of interest. Many research journals are accessible online—some are free and others are available through personal or organizational memberships in professional societies. The Web is truly a research bonanza that leverages your learning experience. Index We provide a two-level index to help you locate any term or concept quickly. For example, each operating system mentioned in the text appears in the index both alphabetically and indented under the term "Operating Systems." Feature Boxes The second edition of Operating Systems included a full chapter on analytic modeling with queuing theory and Markov processes. This edition omits that material, recognizing that operating systems is for the most part not a mathematical field. Rather it has a basis in what we call "systems thinking"—operating systems is largely a field of empirical results. To illuminate these issues, we have included in this edition four types of feature boxes, presenting material to challenge, entertain and enrich you. Boxes xli xlii Preface Biographical Notes Eighteen Biographical Notes provide interesting details about people who have made significant contributions to the field of operating systems—Edsger Dijkstra, Linus Torvalds, David Cutler, Ken Thompson, Dennis Ritchie, Doug Engelbart, Tim Berners-Lee, Richard Stallman, Gordon Moore, Fernando J. Corbato, Leslie Lamport, Per Brinch Hansen, Peter Denning, Seymour Cray, Bill Gates, Ronald Rivest, Adi Shamir and Leonard Adleman. Please do not attach any particular significance to our choices; these are only a few among thousands of important contributors. Please let us know of other people you feel deserve mention. Mini Case Studies In addition to the detailed case studies on the Linux and Windows XP operating systems, 14 Mini Case Studies focus on other important operating systems of historical research or commercial interest. These Mini Case Studies include Mach, CTSS and Multics, UNIX Systems, Real-Time Operating Systems, Atlas, IBM Mainframe Operating Systems, Early History of the VM Operating System, MS-DOS, Supercomputers, Symbian OS, OpenBSD, Macintosh, User-Mode Linux (UML) and OS/2. Anecdotes One of the authors, HMD, accumulated many anecdotes over decades of teaching operating systems in academia and industry. We have included 16 of these anecdotes. Some are humorous; others are thought provoking—raising deep philosophical issues. Each is a (hopefully) pleasant diversion from the book's technical discussions and concludes with a Lesson to operating systems designers. Operating Systems Thinking Academics enjoy the luxury of being able to study what is interesting about operating systems, especially clever algorithms, data structures and occasionally, areas that lend themselves nicely to mathematical analysis. Industry professionals must build real systems that work and meet the demanding cost, performance and reliability requirements of customers. Both kinds of thinking are rich with interesting issues. There are considerable overlaps as well as significant differences in what academics and industry professionals think about. This book aims to present a balanced treatment of both the academic and industry sides of operating systems theory and practice. What is "operating systems thinking?" Forty-three Operating Systems Thinking features explore that question. Indeed, some aspects of operating systems lend themselves to sophisticated mathematical analysis. But the author's (HMD's) extended experience in the computer industry—42 years, including working (as a very junior person) on major operating systems research and development efforts at IBM and MIT, writing two earlier editions of this book and teaching operating systems in academia and industry dozens of times—has shown that these systems are far too complex for a significant mathematical treatment at the undergraduate or early graduate level. Even at the advanced graduate level, except for narrow areas of interest, operating systems defy mathematical analysis. Case Studies xliii If there is not a mathematical basis for evaluating aspects of operating systems, how can we think about them productively? The answer is what we call "systems thinking," and the text certainly covers this. However, the Operating Systems Thinking features are our effort to capture key concepts that are prevalent in operating systems design and implementation. The Operating Systems Thinking features are: Innovation; Relative Value of Human and Computer Resources; Performance; Keep It Simple (KIS); Architecture; Caching; Legacy Hardware and Software; Principle of Least Privilege; Protection; Heuristics; Customers Ultimately Want Applications; Data Structures in Operating Systems; Asynchronism vs. Synchronism; Concurrency; Parallelism; Standards Conformance; Scalability; Information Hiding; Waiting, Deadlock and Indefinite Postponement; Overhead; Predictability; Fairness; Intensity of Resource Management vs. Relative Resource Value; There Are No Upper Limits to Processing Power, Memory, Storage and Bandwidth; Change Is the Rule Rather Than the Exception; Spatial Resources and Fragmentation; Visualization; Empirical Results and Locality-Based Heuristics; Lazy Allocation; Computer Theory in Operating Systems; Space-Time Trade-offs; Saturation and Bottlenecks; Compression and Decompression; Redundancy; Fault Tolerance; Mission-Critical Systems; Encryption and Decryption; Security; Backup and Recovery; Murphy's Law and Robust Systems; Graceful Degradation; Data Replication and Coherency and Ethical Systems Design Case Studies Chapters 20 and 21 cover in depth the Linux and Windows XP operating systems, respectively. These thorough case studies were carefully reviewed by key Linux and Windows XP operating systems developers. The outlines for each of these case studies mimic the text's table of contents. The case studies truly reinforce the text's key concepts—the text presents the principles; the case studies show how these principles are applied in building today's two most widely used operating systems. The Linux case study follows the development of the latest kernel release (v. 2.6) and contains 262 citations. The Windows XP case study reflects the latest Windows operating system features and contains 485 citations. Tour of the Book This section provides an overview of the eight parts and 21 chapters of Operating Systems, 3/e. Part 1—Introduction to Hardware, Software and Operating Systems— includes two chapters that introduce the notion of operating systems, present a history of operating systems and lay the groundwork of hardware and software concepts the reader will use throughout the book. Chapter 1—Introduction to Operating Systems—defines the term "operating system" and explains the need for such systems. The chapter provides a historical perspective on operating systems, tracing their development decade by decade xliv Preface through the second half of the 20th century. The batch-processing systems of the 1950s are considered. We see the 1960s trend towards parallelism with the advent of mutiprogramming—both batch mutiprogramnming and interactive timesharing systems. We follow the development of key operating systems including CTSS, Multics, CP/CMS and Unix. The chapter considers the kinds of thinking that operating systems designers did in an era when computing resources were far more expensive than people resources (today the reverse is true). We follow the 1970s evolution of computer networking, the Internet and the TCP/IP protocol suite, and see the beginnings of the personal computing revolution. Personal computing matures in the 1980s with the release of the IBM personal computer and the Apple Macintosh, the latter popularizing the graphical user interface (GUI). We see the beginnings of distributed computing and the development of the client/server model. In the 1990s, Internet usage literally explodes with the availability of the World Wide Web. Microsoft becomes the world's dominant software maker and releases its Windows NT operating system (the ancestor of today's Windows XP operating system—the focus of Chapter 21). Object technology becomes the dominant development paradigm with languages like C++ and Java becoming popular. The rapid rise of the open-source-software movement leads to the phenomenal success of the Linux operating system—the focus of Chapter 20. We discuss how operating systems provide a platform for applications development. Embedded systems are considered with an emphasis on how crucial it is that mission-critical and business-critical systems be extraordinarily reliable. We consider the core components of operating systems and key operating systems goals. (The book maintains a focus on performance issues in virtually every aspect of operating systems.) Operating system architectures are introduced, including monolithic architecture, layered architecture, microkernel architecture and networked and distributed operating systems. Chapter 2—Hardware and Software Concepts—summarizes the hardware and software resources operating systems manage. The chapter covers how trends in hardware design—most notably phenomenal increases in processing power, memory capacity and communications bandwidth—have affected operating system design and vice versa. Hardware components including mainboards, processors, clocks, main memory, secondary storage devices, buses, direct memory access (DMA), peripheral devices and more are discussed. We present recent and emerging hardware technologies, and discuss hardware support for operating systems, including processor execution modes, privileged instructions, timers, clocks, bootstrapping and plug-and-play Performance enhancement techniques like caching and buffering are considered. Software concepts are examined including compiling, linking, loading, machine languages, assembly languages, interpreters, compilers, high-level languages, structured programming, object-oriented programming and application programming interfaces (APIs). The chapter also explains firmware and middleware. Part 2—Processes and Threads—includes six chapters that present the notions of processes, threads, process- and thread-state transitions, interrupts, con- Tour of the Book xlv text switching, asynchronism, mutual exclusion, monitors, deadlock and indefinite postponement, and processor scheduling of processes and threads. Chapter 3—Process Concepts—begins our discussion of operating system primitives by defining the fundamental notion of process. We consider the life cycle of a process as it transitions between process states. The representation of a process as a process-control block or process descriptor is discussed with an emphasis on the importance of data structures in operating systems. The chapter motivates the need for process structures in an operating system and describes operations that can be performed on them, such as suspend and resume. Multiprogramming considerations, including suspending process execution and context switching, are introduced. The chapter discusses interrupts—a key to the successful implementation of any multiprogrammed environment. We consider interrupt processing and interrupt classes, interprocess communication with signals and message passing. We conclude with a case study of UNIX processes. Chapter 4—Thread Concepts—extends our discussion of process concepts to a smaller unit of concurrent program execution: the thread. The chapter defines what threads are and explains their relationship to processes. The life cycle of a thread and how threads transition among various thread states is discussed. We consider various threading architectural models, including user-level threads, kernellevel threads and combining user- and kernel-level threads. The chapter presents thread implementation considerations, including thread signal delivery and thread termination. We discuss the POSIX standard and its threading specification, Pthreads. The chapter concludes with a presentation of the Linux, Windows XP and Java threading implementations. The Java presentation is accompanied by a complete, working Java program with sample outputs. [Note: The Java source code for all of the Java programs in the book is available for download at www.deitel.com/ books/os3e.] Chapter 5—Asynchronous Concurrent Execution—discusses the issues of concurrency encountered in multiprogrammed systems. The chapter introduces the problem of mutual exclusion and how threads must manage access to shared resources. A feature of the chapter is the Java Multithreading Case Study: A Producer/Consumer Relationship in Java—which uses a complete, working Java program and several sample outputs to clearly illustrate what happens when concurrent threads access shared data without synchronization. This example clearly shows that sometimes such a concurrent program will work fine and sometimes it will produce erroneous results. We show how to solve this problem with a multithreaded Java program in Chapter 6. The concept of the critical section in program code is introduced. Several software mechanisms that protect access to critical sections are presented as solutions to the mutual exclusion problem; these include Dekker's Algorithm, Peterson's Algorithm and Nthread mutual exclusion with Lamport's Bakery Algorithm. The chapter also discusses hardware mechanisms that facilitate the implementation of mutual exclusion algorithms; these include disabling interrupts, the test-and-set instruction and the swap instruction. Finally, semaphores are presented as a high-level mechanism for xlvi Preface implementing mutual exclusion and thread synchronization; both binary semaphores and counting semaphores are considered. Chapter 6—Concurrent Programming—explains the notion of monitors (a high-level mutual exclusion construct) then proceeds to solve several classic problems in concurrent programming, using first pseudocode monitors then complete working Java programs with sample outputs. The monitor solutions are provided in a C/C++/Java-like pseudocode syntax. We explain how monitors enforce information hiding and discuss how monitor condition variables differ from "conventional" variables. The chapter illustrates how a simple monitor may be used to control access to a resource that requires exclusive use. We then discuss two classic problems in concurrent programming—the circular buffer and the readers-and-writers problem; we implement solutions to each of these with pseudocode monitors. Java monitors are explained and the differences between Java monitors and the ones described by the classic literature are discussed. The chapter continues our optional Java Multithreading Case Study by implementing a producer/consumer relationship and a circular buffer in Java. The student can use this latter program as a basis for implementing a solution to the readers-and-writers problem in Java. Chapter 7—Deadlock and Indefinite Postponement—introduces two potentially disastrous consequences of waiting: deadlock and indefinite postponement. The key concern is that systems that manage waiting entities must be carefully designed to avoid these problems. Several examples of deadlock are presented, including a traffic deadlock, the classic one-lane bridge, a simple resource deadlock, deadlock in spooling systems and deadlock in Dijkstra's charming Dining Philosophers problem. Key resource concepts are considered, including preemptibility, sharing, reentrancy and serial reusability. The chapter formally defines deadlock and discusses deadlock prevention, avoidance, detection and recovery (almost invariably painful). The four necessary conditions for deadlock are explained, namely the "mutual-exclusion," "wait-for," "no-preemption" and "circular-wait" conditions. We examine Havender's methods for preventing deadlock by denying, individually, any of the last three conditions. Deadlock avoidance enables more flexible resource allocation than deadlock prevention. The chapter explains deadlock avoidance with Dijkstra's Banker's Algorithm, showing examples of a safe state, an unsafe state and a safe-state-to-unsafe-state transition. Weaknesses in the Banker's Algorithm are discussed. We explain deadlock detection with the resource-allocation graph reduction technique. The chapter concludes with a discussion of deadlock strategies in current and future systems. Chapter 8—Processor Scheduling—discusses concepts and algorithms related to allocating processor time to processes and threads. Scheduling levels, objectives and criteria are considered. Preemptive and nonpreemptive scheduling approaches are compared. We explain how to carefully set priorities and

quanta (finite-sized allocations of processor time) in scheduling algorithms. Several classic and current scheduling algorithms are considered, including first-in-first-out (FIFO), roundrobin (RR), shortest-process-first (SPF), highest-response-ratio-next (HRRN), Tour of the Book xlvii shortest-remaining-time (SRT), multilevel feedback queues and fair-share scheduling. Each algorithm is evaluated using metrics such as throughput, average response time and the variance of response times. We discuss Java thread scheduling, soft real-time scheduling, hard real-time scheduling and deadline scheduling. Part 3—Real and Virtual Memory—includes three chapters that discuss memory organization and memory management in real and virtual memory systems. Chapter 9—Real Memory Management and Organization—presents a historical discussion of how real memory operating systems have organized and managed physical memory resources. The schemes have gone from the simple to the complex, ultimately seeking optimal usage of the relatively precious main memory resource. We review the memory hierarchy consisting of cache(s), primary memory and secondary storage. Then, three types of memory management strategies are discussed, namely fetch, placement and replacement. We present contiguous and noncontiguous memory allocation schemes. Single-user contiguous memory allocation is considered with a discussion of overlays, protection and single-stream batch processing. We trace the evolution of multiprogramming memory organizations from fixed-partition multiprogramming to variable-partition multiprogramming, considering issues including internal and external memory fragmentation, and presenting memory compaction and coalescing as means of reducing fragmentation. The chapter discusses the first-fit, best-fit and worst-fit memory placement strategies, and concludes with a discussion of multiprogramming with memory swapping. Chapter 10—Virtual Memory Organization—describes fundamental virtual memory concepts and the hardware capabilities that support virtual memory. The chapter motivates the need for virtual memory and describes typical implementations. The key approaches to virtual memory organization—paging and segmentation—are explained and their relative merits are analyzed. We discuss paging systems, focusing on paging address translation by direct mapping, paging address translation by associative mapping, paging address translation with direct/associative mapping, multilevel page tables, inverted page tables and sharing in paging systems. The chapter investigates segmentation systems, focusing on segmentation address translation by direct mapping, sharing in a segmentation system, protection and access control in segmentation systems. We also examine hybridized segmentation/paging systems, considering dynamic address translation, sharing and protection in these systems. The chapter concludes by examining the popular IA-32 Intel architecture virtual-memory implementation. Chapter 11—Virtual Memory Management—continues the discussion of virtual memory by analyzing how operating systems attempt to optimize virtual memory performance. Because paging systems have become dominant, we focus on page management in detail, most notably on page-replacement strategies. The chapter considers one of the most important empirical results in the field of operating systems, namely the locality phenomenon, and we consider the results from both the temporal and spatial perspectives. We discuss when pages should be brought into memory, examining both demand paging and anticipatory paging. xlviii Preface When available memory becomes scarce, incoming pages must replace pages already in memory—an operating system's page-replacement strategy can have an enormous impact on performance. Many page-replacement strategies are examined, including random, first-in-first-out (FIFO and Belady's Anomaly), leastrecently-used (LRU), least-frequently-used (LFU), not-used-recently (NUR), second-chance, clock and segmented queue (SEGQ), far page and page-fault-frequency (PFF). Denning's classic working set model of program behavior is considered as well as how various page-replacement strategies attempt to achieve its goals. The chapter discusses the possibility of voluntary page release. We carefully examine the arguments for small pages and large pages, and how programs behave under paging. The chapter discusses Linux page replacement and concludes with a discussion of global vs. local page-replacement strategies. Part 4—Secondary Storage, Files and Databases—includes two chapters that present techniques which operating systems employ to manage data on secondary storage. Hard-disk performance optimization, and file and database systems are discussed. Our treatment of I/O systems is distributed throughout the book, most notably in Chapters 2, 12 and 13 and in the case studies on Linux and Windows XP (Chapters 20 and 21, respectively). Chapter 12—Disk Performance Optimization—focuses on the characteristics of moving-head disk storage and how the operating system can optimize its performance. The chapter discusses the evolution of secondary-storage devices and examines the characteristics of moving-head disk storage. During a half century of explosive technological development, moving-head disk storage devices continue to endure. We define why disk scheduling is necessary and demonstrate why it is needed to achieve high-performance from moving-head disk devices. The evolution of diskscheduling strategies is presented, including seek-optimization strategies, including first-come-first-served (FCFS), shortest-seek-time-first (SSTF), SCAN, C-SCAN, FSCAN, N-Step SCAN, LOOK, C-LOOK and VSCAN (introduced in the exercises), and rotational-optimization strategies, including SLTF, SPTF and SATF. Other popular disk-system performance techniques are discussed, including caching, buffering, defragmentation, data compression and blocking. One of the most important additions to this third edition is the extensive treatment of RAID systems in this chapter. RAID (Redundant Arrays of Independent Disks) is a set of technologies that enable disk systems to achieve higher performance and fault tolerance. The chapter presents various key RAID "levels," including level 0 (striping), level 1 (mirroring), level 2 (bit-level Hamming ECC parity), level 3 (bit-level XOR ECC parity), level 4 (blocklevel XOR ECC parity) and level 5 (block-level distributed XOR ECC parity). Chapter 13—File and Database Systems—discusses how operating systems organize and manage collections of data called files and databases. We explain key concepts, including the data hierarchy, files, file systems, directories, links, metadata and mounting. The chapter presents various file organizations, including sequential, direct, index sequential and partitioned. File allocation techniques are examined, including contiguous file allocation, linked-list noncontiguous file allocation, tabu- Tour of the Book xlix lar noncontiguous file allocation and indexed noncontiguous file allocation. We explain file access control by user classes and access control matrices. Data access techniques are discussed in the context of basic access methods, queued access methods, anticipatory buffering and memory-mapped files. We explain techniques for ensuring data integrity, including protection, backup, recovery, logging, atomic transactions, rollback, commitment, checkpointing, shadow paging and using logstructured file systems. The chapter considers file servers and distributed systems and points the reader to the extensive treatment of these topics and others in Chapters 16-18. The chapter also introduces database systems, and analyzes their advantages, data access, the relational database model and operating system services that support database systems. Part 5—Performance, Processors and Multiprocessor Management—includes two chapters that discuss performance monitoring, measurement and evaluation techniques, and focus on the extraordinary performance that can be achieved with systems that employ multiple processors. Chapter 14—Performance and Processor Design—focuses on one of operating systems designers' most preeminent goals —system performance —and discusses the important role of specific processor types in achieving that goal. The chapter surveys performance measures, considering issues including absolute performance measures, ease of use, turnaround time, response time, system reaction time, variance in response times, throughput, workload, capacity and utilization. We discuss performance evaluation techniques, including tracing and profiling, timings, microbenchmarks, application-specific evaluation, analytic models, benchmarks, synthetic programs, simulation and performance monitoring. Bottlenecks and saturation are considered. We demonstrate how systems dynamically adjust to positive and negative feedback. A system's performance depends heavily on the performance of its processor(s), which is, in turn, heavily influenced by its instruction set architecture. The chapter examines key architectures, including complex instruction set computing (CISC), reduced instruction set computing (RISC) and various kinds of post-RISC processors. The chapter concludes with a discussion of explicitly parallel instruction computing (EPIC). Chapter 15—Multiprocessor Management—provides an in-depth introduction to the hardware and software aspects of multiprocessing. One way to build ever more powerful computing systems is to employ multiple processors and possibly massive numbers of them. The discussion begins with a Mini Case Study on supercomputers followed by a Biographical Note on Seymour Cray, the father of supercomputing. We investigate multiprocessor architecture, considering issues including the classification of sequential and parallel architectures, processor interconnection schemes and loosely coupled vs. tightly coupled systems. Multiprocessor operating system organizations are examined, including master/slave, separate kernels and symmetrical organization. We explain memory access architectures, including uniform memory access (UMA), nonuniform memory access (NUMA), cacheonly memory architecture (COMA) and no remote memory access (NORMA). l Preface The chapter discusses multiprocessor memory sharing, considering issues of cache coherence, page replication and migration and shared virtual memory. We continue the discussion on processor scheduling which we began in Chapter 8, presenting job-blind multiprocessor scheduling and job-aware multiprocessor scheduling. Process migration is considered and issues of flow of process migration and process migration strategies are examined. We discuss multiprocessor load balancing, examining both static and dynamic load balancing strategies. The chapter explains how to enforce multiprocessor mutual exclusion with spin locks, sleep/wakeup locks and read/write locks. Part 6—Networking and Distributed Computing—includes three chapters that present networks, networked systems and distributed systems. Chapter 16—Introduction to Networking—introduces computer networking to lay the foundation for the following two chapters on distributed computing. The chapter discusses network topologies, including buses, rings, meshes, fully connected meshes, stars and trees and we explain the unique challenges posed by wireless networks. We consider local-area networks (LANs) and wide-area networks (WANs), and present a major treatment of the TCP/IP protocol stack. Application layer protocols including the Hypertext Transfer protocol (HTTP) and the File Transfer Protocol (FTP) are examined. We explain transport layer protocols including the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). At the network layer the chapter explores the Internet Protocol (IP) and its latest version—Internet Protocol version 6 (IPv6). The chapter discusses link layer protocols, including Ethernet, Token Ring, Fiber Distributed Data Interface (FDDI) and IEEE 802.11 (wireless). The chapter concludes with a discussion of the client/server model and n-tier systems. Chapter 17—Introduction to Distributed Systems —introduces distributed operating systems and discusses the attributes of distributed systems, including performance, scalability, connectivity, security, reliability, fault tolerance and transparency. We compare and contrast network operating systems and distributed operating systems. Communication in distributed systems and the crucial role played by "middleware" technologies are discussed, including remote procedure call (RPC), Remote Method Invocation (RMI), CORBA (Common Object Request Broker Architecture), and Microsoft's DCOM (Distributed Component Object Model). The chapter considers process migration, synchronization and mutual exclusion in distributed systems, examining mutual exclusion without shared memory, and Agrawala and Ricart's distributed mutual exclusion algorithm. We discuss deadlock in distributed systems, focusing on deadlock prevention and deadlock detection. The chapter concludes with case studies on the Sprite and Amoeba distributed operating systems. Chapter 18—Distributed Systems and Web Services—continues the study of distributed systems, focusing on distributed file systems, clustering, peer-to-peer computing, grid computing, Java distributed computing and Web services. The chapter begins by considering some characteristics and concerns of distributed systems, Tour including transparency, scalability, security, fault tolerance and consistency and we present case studies on key distributed file systems, including the Network File System (NFS), Andrew File System (AFS), Coda File System and Sprite File System. We discuss clustering, considering high-performance clusters, high-availability clusters and load-balancing clusters; we investigate various examples of clusters including Linux-based Beowulf clusters Windows-based clusters. Peer-to-peer (P2P) distributed computing is explored considering the relationship between P2P and client/server applications, centralized vs. decentralized P2P applications, peer discovery and searching. We examine Sun Microsystems' Project JXTA and we consider how JXTA creates a framework for building P2P applications. The chapter considers grid computing and how it makes possible solutions to problems that require a truly massive amount of computation by using available unused processing power on personal and business computers worldwide. We discuss key Java distributed computing technologies, including Java servlets and JavaServer Pages (JSP), Jini, JavaSpaces and Java Management Extensions (JMX). The chapter concludes by surveying the exciting new technology of Web services and exploring two key Web services platforms—Microsoft's .NET Platform and Sun Microsystems' Sun ONE platform. Part 7—Security—includes one chapter. Chapter 19—Security—presents a general introduction to computer and network security techniques that operating systems can use to provide secure computing environments. The chapter discusses secret-key and public-key cryptography and the popular RSA and PGP (Pretty Good Privacy) public key schemes. Authentication is considered: password protection, password salting, biometrics, smart cards. Kerberos and single sign-on. We discuss access control, exploring issues of access rights, protection domains, access control models, access control policies and access control mechanisms, including access control matrices, access control lists and capability lists. The chapter presents the great variety of security attacks that have been attempted, including cryptanalysis, viruses, worms, denial-of-service (DoS) attacks, software exploitation and system penetration. We survey attack prevention and security solutions, including firewalls and intrusion detection systems (IDSs), antivirus software, security patches and secure file systems. We discuss the federal government's Orange Book Security classification system and present a Mini Case Study on OpenBSD, arguably the most secure operating system available. The chapter explores secure communication and consider the requirements for a successful, secure transaction—privacy, integrity, authentication, authorization and nonrepudiation. Key agreement protocols—the process by which two parties exchange secret keys over an unsecure medium—are discussed. We explain digital signatures —a technology that is enormously crucial to the future of electronic commerce—and their implementation. The chapter presents an in-depth treatment of public-key infrastructure, including digital certificates and certificate authorities. Various secure communication protocols are considered including Secure Sockets Layer (SSL), Virtual Private Networks (VPNs), IP Security (IPSec) and wireless security. We discuss the intriguing technology of steganography—the practice of of the Book li lii Preface hiding information within other information. This can hide secret messages in publicly transmitted messages; it can also protect intellectual property rights, with digital watermarks for example. The chapter compares and contrasts proprietary and open-source security solutions and concludes with a case study on UNIX Security. Part 8—Operating Systems Case Studies—includes two chapters that provide in-depth case studies on the Linux 2.6 kernel and the Microsoft Windows XP operating system,. The case studies follow the book's outline to make it convenient for the student to comprehend topics discussed earlier in the book. Chapter 20—Case Study: Linux—provides an in-depth study of Linux 2.6. This extensive (94 pages) case study was written and updated throughout the development of the release (by following the version 2.5 development kernel and having the material carefully reviewed by key Linux kernel developers). The chapter discusses the history, community and software distributions that have created the most popular open-source operating system in the world. The chapter examines the core components of the Linux operating system, with particular attention to their implementation in the context of concepts studied in previous chapters. Kernel architecture, process management (processes, threads, scheduling), memory organization and management, file systems (virtual file system, virtual file system caches, ext2fs and proc fs), I/O management (device drivers, character device I/O, block device I/ O, network device I/O, unified device model, interrupts), synchronization (spin locks, reader/writer locks, seqlocks and kernel semaphores), IPC (signals, pipes, sockets, message queues, shared memory and System V semaphores), networking (packet processing, netfilter framework and hooks), scalability (symmetric multiprocessing, nonuniform memory access, other scalability features, embedded Linux) and security are all analyzed and explained. Chapter 21—Case Study: Windows XP—complements the Linux case study by examining the internals of the most popular commercial operating system— Windows XP. This case study (106 pages) examines the core components of the Windows XP operating system and how they interact to provide services to users. We discuss the history of Windows operating systems, including Biographical notes on Bill Gates and David Cutler. The chapter presents the design goals of Windows XP and overviews its system architecture, considering topics including the Hardware Abstraction Layer (HAL), the microkernel, the executive, the environment subsystems, dynamic link libraries (DLLs) and system services. We discuss system management mechanisms, including the registry, Object Manager, interrupt request levels (IRQLs), asynchronous procedure calls (APCs), deferred procedure calls (DPCs) and system threads. We examine process and thread organization, considering control blocks, thread local storage (TLS), creating and terminating processes, jobs, fibers and thread pools. Thread scheduling is explained, and thread states, the thread scheduling algorithm, determining thread priorities and multiprocessor scheduling are considered. We investigate thread synchronization, examining dispatcher objects, event objects, mutex objects, semaphore objects, waitable timer objects, kernel mode locks and other synchronization tools. Memory management. Ancillary Package for Operating Systems, 3/e liii and the concepts of memory organization, memory allocation and page replacement are explained. We explore file systems management, discussing file system drivers and NTFS topics including the Master File Table (MFT), data streams, file compression, file encryption, sparse files, reparse points and mounted volumes. We study input/output management, explaining device drivers, Plug-and-Play, power management, the Windows Driver Model (WDM), input/output processing, I/O request packets (IORPs), synchronous I/O, asynchronous I/O, data transfer techniques, interrupt handling and file cache management. We consider interprocess communication mechanisms including pipes (anonymous and named), mailslots, shared memory, and local and remote procedure calls. Microsoft's Component Object Model (COM) is overviewed. We explain drag-and-drop and compound documents. We discuss networking capabilities, including network input/output, network driver architecture, network protocols (IPX, SPX, NetBEUI, NetBIOS over TCP/IP, WinHTTP, WinlNet and Winsock 2. Network services topics including Active Directory, Lightweight Directory Access Protocol (LDAP) and Remote Access Service (RAS) are discussed. We overview Microsoft's new .NET technology which is replacing DCOM. We examine scalability, considering both symmetric multiprocessing (SMP) and Windows XP Embedded. The chapter concludes with a discussion of security topics, including authentication, authorization, Internet Connection Firewall (ICF) and other security features. Whew! Well, that completes the Tour of the Book. There is much here for operating systems students and professionals. Ancillary Package for Operating Systems, 3/e Operating Systems, 3/e has extensive ancillary materials for instructors. The Instructor's Resource CD (IRCD) contains solutions to the vast majority of the end-ofchapter exercises. This CD is available only to instructors through their Prentice Hall representatives. Visit vig.prenhall.com/replocator to locate your Prentice Hall representative. [NOTE: Please do not write to us requesting the instructor's CD. Distribution of this CD is limited strictly to college professors teaching from the book. Instructors may obtain the solutions manual only from their Prentice Hall representatives.] The ancillaries for the book also include a Test Item File of multiple-choice questions. In addition, we provide PowerPoint lecture notes containing all figures, illustrations, diagrams and code in the text and bullet points that summarize key portions of the text. Instructors can customize the slides. The PowerPoint® slides are downloadable from www.deitel.com and as part of Prentice Hall's companion Web site (www.prenhall.com/deitel) for Operating Systems, 3/e, which offers resources for both instructors and students. For instructors, the Companion Web site includes a Syllabus Manager, which helps instructors plan courses interactively and create online syllabi. Students also benefit from the functionality of the Companion Web site. Book-specific resources for students include: liv Preface • Customizable PowerPoint lecture notes • Source code for the Java programs Chapter-specific resources available for students include: • Chapter objectives • Outline • Highlights (e.g., chapter summary) • Web links Acknowledgments One of the great pleasures of writing a textbook is acknowledging the efforts of many people whose names may not appear on the cover, but whose hard work, cooperation, friendship and understanding were crucial to the production of the book. Many of our colleagues at Deitel & Associates, Inc. devoted long hours to this book and its ancillaries: Jeff Listfield, Su Zhang, Abbey Deitel, Barbara Deitel and Christi Kelsey We would like to include a special note of thanks to Ben Wiedermann for organizing the effort to revise the material in the second edition of this book. He reviewed and updated much of the existing writing and diagrams. He also organized and directed a team of Deitel interns who researched the latest developments in Operating Systems for inclusion in this new edition. Ben currently is pursuing a Ph.D. in Computer Science at the University of Texas, Austin. We would also like to thank the participants in the Deitel & Associates, Inc., College Internship Program who worked on the content of this book and its ancillary package: Jonathan Goldstein (Cornell University), Lucas Ballard (Johns Hopkins University), Tim Christensen (Boston College), John Paul Casiello (Northeastern University), Mira Meyerovich (Ph.D. candidate at Brown University), Andrew Yang (Carnegie Mellon University), Kathryn Ruigh (Boston College), Chris Gould (University of Massachusetts at Lowell), Marc Manara (Harvard University), Jim O'Leary (Rensselaer Polytechnic Institute), Gene Pang (Cornell University) and Randy Xu (Harvard University). We would like to acknowledge Howard Berkowitz (formerly of the Corporation for Open Systems International) for co-authoring Chapter 16, Distributed Computing, in the second edition of this text. His work was the basis for our expanded, enhanced and updated treatment of networking and distributed computing in Chapters 16 through 18 of Operating Systems, 3/e. We are fortunate to have worked on this project with the talented and dedicated team of publishing professionals at Prentice Hall. We especially appreciate the extraordinary efforts of our Computer Science Editor, Kate Hargett and her boss and our mentor in publishing—Marcia Horton, Editorial Director of Prentice-Hall's Engineering and Computer Science Division. Vince O'Brien, Tom Manshreck, John Lovell and Chirag Thakkar did a marvelous job managing the production of the book. Sarah Parker managed the publication of the book's ancillary package. Acknowledgments We would like to thank the design team that created a completely new look and feel for Operating Systems, 3/e—Carole Anson, Paul Belfanti, Geoffrey Cassar and John Root. We would also like to thank the talented creative team at Artworks, including Kathryn Anderson, Jay McElroy, Ronda Whitson, Patricia Burns, Matt Haas, Xiaohong Zhu, Dan Missildine, Chad Baker, Sean Hogan, Audrey Simonetti, Mark Landis, Royce Copenheaver, Stacy Smith, Scott Wieber, Pam Taylor, Anna Whalen, Cathy, Shelly, Ken Mooney, Tim Nguyen, Carl Smith, Jo Thompson, Helenita Zeigler and Russ Crenshaw. The most important acknowledgment is to the thousands of authors represented in the Recommended Readings and Works Cited sections in the chapters; their research papers, articles and books have provided the diversity of interesting material that makes operating systems such a fascinating area. We wish to acknowledge the efforts of our Third Edition reviewers and to give a special note of thanks to Carole Snyder and Jennifer Cappello of Prentice Hall who managed this extraordinary review effort. Adhering to a tight time schedule, these reviewers scrutinized the text, providing countless suggestions for improving the accuracy and completeness of the presentation. It is a privilege to have the guidance of such talented and busy professionals. Third Edition reviewers: Tigran Aivazian, Veritas Software, Ltd. Jens Axboe, SUSE Labs Dibyendu Baksi, Scientific Technologies Corporation Columbus Brown, IBM Fabian Bustamante, Northwestern University Brian Catlin, Azius Developer Training Stuart Cedrone, Hewlett-Packard Randy Chow, University of Florida Alan Cox, Red Hat UK Matthew Dobson, Independent Consultant Ulrich Drepper, Red Hat, Inc. Alessio Gaspar, University of South Florida, Lakeland Campus Allison Gehrke, Colorado Technical University Michael Green, Central Texas College Rob Green, Columbia Data Products, Inc. James Huddleston, Independent Consultant Scott Kaplan, Amherst College Salahuddin Khan, Microsoft Robert Love, MontaVista Software, Inc. Barry Margolin, Level 3 Communications Cliff Mather, Hewlett-Packard Jeanna Matthews, Cornell University/Clarkson University Manish Mehta, Syntel, India, Ltd. Dejan Milojicic, Hewlett-Packard lv lvi Preface Euripides Montagne, University of Central Florida Andrew Morton, Diego, Inc. Gary Newell, Northern Kentucky University Bill O'Farrell, IBM Mike Panetta, Robotic Sciences, Inc./Netplanner Systems, Inc. Rohan Phillips, Microsoft Atul Prakash, University of Michigan Bina Ramamurthy, SUNY Buffalo Eric Raymond, Thyrsus Enterprises Jeffrey Richter, Wintellect Sven Schreiber, Platon Data Technology Wennie Wei Shu, University of New Mexico David Solomon, David Solomon Expert Seminars Brandon Taylor, Sun Microsystems Bob Toxen, Fly-By-Day Consulting, Inc. Joshua Uziel, Sun Microsystems Carlos Valcarcel, EinTech, Inc. Melinda Ward, Hewlett-Packard Gerard Weatherby, Charles Consulting, LLC. /Rensselaer at Hartford Xiang Xu, RSA Security, Inc. First and Second Edition reviewers and contributors (and their affiliations at that time): Howard Berkowitz, Corporation for Open Systems Dale A. Brown, The College of Wooster Steven J. Buroff, AT&T Bell Laboratories David D. Busch John H. Carson, George Washington University Ronald Curtis, Canisius College Larry K. Flanigan, University of Michigan Jon Forrest, Sybase Jay Gadre, Corporation for Open Systems Carlos M. Gonzalez, George Mason University Mike Halvorson, Microsoft Press Wayne Hathaway, Ultra Network Technologies Christopher T Haynes, Indiana University Lee Hollaar, University of Utah William Horst, Corporation for Open Systems James Johannes, University of Alabama Ralph Johnson, University of Illinois Dennis Kafura, Virginia Polytechnical Institute Herb Klein, Corporation for Open Systems Michael S. Kogan, IBM Thomas LeBlanc, University of Rochester Contacting Deitel & Associates lvii T. F. Leibfried, University of Houston, Clear Lake David Litwack, Corporation for Open Systems Mark Measures, Baylor University Charles Oualline, Jr., East Texas State University Ravi Sandhu, Ohio State University Richard Schlichting, University of Arizona Alan Southerton John J. Zenor, California State University James Peterson, University of Texas at Austin Richard Wexelblatt, Sperry Univac Paul Ross, Millersville State University Anthony Lucido, Intercomp Steve Paris, Prime Computer Bart Guerreri, DSD Laboratories Nathan Tobol, Codex Corporation and Chairman, IEEE 802 Local-Area Networking Subcommittee Larry Nelson, AVCO Services Barry Shein, Harvard University Eliezer Garni, MIT Anat Gafni, Boston University Josefina Bondoc, Boston University Contacting Deitel & Associates We would sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence to: We will respond promptly. Deitel Buzz® Online E-mail Newsletter Our free e-mail newsletter, the DEITEL BUZZ® Online, includes updates to Operating Systems 3/e, commentary on industry trends and developments, links to free articles and resources from our published books and upcoming publications, product-release schedules, errata, challenges, anecdotes, information on our corporate instructor-led training courses and more. To subscribe, visit www.deitel.com/newsletter/subscribe.html About Deitel & Associates, Inc. Deitel & Associates, Inc., is an internationally recognized corporate training and content-creation organization specializing in Internet/World Wide Web software technology, e-business/e-commerce software technology, object technology, operating systems and computer programming languages education. The company pro- lvii Preface vides instructor-led courses on Internet and Web programming, wireless Internet programming, object technology, and major programming languages and platforms, including C, C++, Visual C++® .NET, Visual Basic® .NET, C#, Java, J2EE, XML, Perl, Python and more. The founders of Deitel & Associates, Inc., are Dr. Harvey M. Deitel and Paul J. Deitel. The company's clients include many of the world's largest computer companies, government agencies, branches of the military and business organizations. Through its 27-year publishing partnership with Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks, professional books, interactive CD-based multimedia Cyber Classrooms, Complete Training Courses, Web-based training courses and course management systems econtent for popular CMSs such as WebCT™, Blackboard™ and CourseCompass SM . To learn more about Deitel & Associates, Inc., its publications and its corporate on-site training curriculum, see the last few pages of this book or visit: www.deitel.com Individuals wishing to purchase Deitel books, Cyber Classrooms, Complete Training Courses and Web-based training courses can do so through bookstores, online booksellers and the following Web sites: www.deitel.com www.prenhall.com/deitel www.InformIT.com/deitel www.InformIT.com/cyberclassrooms Bulk orders by corporations and academic institutions should be placed directly with Prentice Hall. See www.prenhall.com/deitel for worldwide ordering instructions and to find your local Prentice Hall sales representative. Welcome to the exciting world of operating systems. We sincerely hope you enjoy learning with this book. Dr. Harvey M. Deitel Paul J. Deitel David R. Choffnes About the Authors Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 42 years experience in the computing field, including extensive industry and academic experience. Dr. Deitel studied operating systems at the Massachusetts Institute of Technology where he earned B.S. and M.S. degrees. Dr. Deitel earned his Ph.D. from Boston University. While an undergraduate and master's candidate at MIT, he worked on the pioneering virtual-memory operating-systems projects at IBM (OS/360 and TSS/360) and MIT (Multics) that developed techniques now widely implemented in systems including UNIX, Linux and Windows About the Authors lix XP. He taught operating systems for 20 years in academia and industry He earned tenure and served as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., with his son, Paul J. Deitel. He and Paul are the co-authors of several dozen books and multimedia packages and they are writing many more. With translations published in Japanese, German, Russian, Spanish, Traditional Chinese, Simplified Chinese, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels' texts have earned international recognition. Dr. Deitel has delivered professional seminars to major corporations, government organizations and the military. Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of the Massachusetts Institute of Technology's Sloan School of Management, where he studied information technology and operating systems. Through Deitel & Associates, Inc., he has delivered hundreds of courses to Fortune 500, academic, government and military clients, and many other organizations. He has lectured for the Boston Chapter of the Association for Computing Machinery. He and his father, Dr. Harvey M. Deitel, are the world's best-selling computer science textbook authors. David R. Choffnes is a graduate of Amherst College, where he earned degrees in Physics and French while pursuing advanced topics in Computer Science. His research interests include the fields of operating systems, computer architecture, computational biology and molecular computing. David has contributed to other Deitel publications including Simply Java™ Programming and Simply Visual Basic® .NET. Introduction to Hardware, Software and Operating Systems Intelligence ...is the faculty of making artificial objects, especilly tolls to make tools. —Henri Bergson— Part 1 The following chapters define the term operating system, present a history of operating systems and lay a foundation of hardware and software concepts. As you study the history of operating systems you will see a constant emphasis on performance improvement and on increasing the intensity of resource management to achieve that improvement. You will see hardware power increasing dramatically as costs decline equally dramatically. You will see the emergence of two dominant operating systems, Linux and Windows XP, built with the open-source approach (Linux) vs. the proprietary approach (Windows XP). You will study key operating system architectures. You will study hardware that operating systems manage for users and hardware that supports operating systems functions. You will study your first performance enhancement techniques—a process that you will continue throughout the book. Every test carries with it the spirit by which it has been created. —Werner Karl Heisenberg— Live in fragments no longer. Only connect... — Edward Morgan Forster — Efficiency is getting the job done right. Effectiveness is getting the right job done. - Z i g Ziglar- Nothing endures but change. —Heraclitus — Open sesame! - T h e History of Ali Baba- Chapter 1 Introduction to Operating Systems Objectives After reading this chapter, you should understand: • what an operating system is. • a brief history of operating systems. • a brief history of the Internet and the World Wide Web. • core operating system components. • goals of operating systems. • operating system architectures. Chapter Outline 1.1 Introduction 1.2 What Is an Operating System? 1.3 Early History: The 1940s and 1950? Operating Systems Thinking: Innovation 1.4 The 1960s Operating Systems Thinking: Relative Value of H u m a n and Computer Resources Biographical Note: Ken Thompson and Dennis Ritchie 1.5 The 1970s Biographical Note: Tim Berners-lee 1.8 The 1 9 9 0 s Biographical Note: Linus Torvalds Biographical Note: Richard Stallman 1.9 2000 and Beyond 1.10 Application Bases 1.11 Operating System Environments Caution 1.12 Operating System Components and Goals 1.6 The 1980s Biographical Note: Doug Engelbart 1.12.1 Core Operating System Components 1.12.2 Operating System Goals Operating Systems Thinking: Performance Anecdote: 4 1.7 History of the Internet and World Wide Web Abraham lincoln's Technology 1.13 Operating Operating System 1.13.1 Monolithic Architecture Architectures Systems Thinking: Keep It Simple (KIS) 1.13.4 Anecdote: System Architect vs. Engineer Operating Systems Thinking: Architecture 1.13.2 Layered Architecture 1.13.3 Microkernel Architecture Networked and Distributed System Operating Systems Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited 5 6 Introduction to Operating Systems 1.1 Introduction Welcome to the world of operating systems. During the past several decades, computing has evolved at an unprecedented pace. Computer power continues to increase at phenomenal rates as costs decline dramatically. Today, computer users have desktop workstations that execute billions of instructions per second (BIPS). and supercomputers that execute over a trillion instructions per second have been built,1, 2 numbers that just a few years ago were inconceivable. Processors are becoming so inexpensive and powerful that computers can be employed in almost every aspect of our lives. On personal computers, users can edit documents, play games, listen to music, watch videos and manage their personal finances. Portable devices, including laptop computers, personal digital assistants (PDAs), cell phones, and MP3 players all have computers as key components. Wired and wireless network architectures are increasing our interconnectivity—allowing users to communicate instantly over vast distances. The Internet and World Wide Web have revolutionized business, creating demand for networks of large, powerful computers that service vast numbers of transactions per second. Networks of computers have become so powerful that they are used to conduct complex research and simulation projects, such as modeling the Earth's climate, emulating human intelligence and constructing lifelike 3D animations. Such pervasive and powerful computing is reshaping the roles and responsibilities of operating systems. In this book, we review operating system principles and discuss cutting-edge advances in computing that are redefining operating systems. We investigate the structure and responsibilities of operating systems. Design considerations, such as performance, fault tolerance, security, modularity and cost, are explored in detail. We also address more recent operating system design issues arising from the rapid growth in distributed computing, made possible by the Internet and the World Wide Web. We have worked hard to create what we hope will be an informative, entertaining and challenging experience for you. As you read this book, you may want to refer to our Web site at www.deitel.com for updates and additional information on each topic. You can reach us at 1.2 What Is an Operating System? In the 1960s, the definition of an operating system might have been the software that controls the hardware. But the landscape of computer systems has evolved significantly since then, requiring a richer definition. Today's hardware executes a great variety of software applications. To increase hardware utilization, applications are designed to execute concurrently. If these applications are not carefully programmed, they might interfere with one another. As a result, a layer of software called an operating system separates applications from the hardware they access and provides services that allow each application to execute safely and effectively. 1.3 Early History: The 1940s and 1950s 7 An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the kernel. Operating systems can be found in devices ranging from cell phones and automobiles to personal and mainframe computers. In most computer systems, a user requests that the computer perform an action (e.g., execute an application or print a document) and the operating system manages the software and hardware to produce the desired result. To most users, the operating system is a "black box" between the applications and the hardware they run on that ensures the proper result, given appropriate inputs. Operating systems are primarily resource managers —they manage hardware, including processors, memory, input/output devices and communication devices. They must also manage applications and other software abstractions that, unlike hardware, are not physical objects. In the next several sections we present a brief history of operating systems from the simple, single-user batch systems of the 1950s to the complex, multiprocessor, distributed, multiuser platforms of today. Self Review 1. (T/F) Operating systems manage only hardware. 2. What are the primary purposes of an operating system? Ans: 1) False. Operating systems manage applications and other software abstractions, such as virtual machines. 2) The primary purposes of an operating system are to enable applications to interact with a computer's hardware and to manage a system's hardware and software resources. 1.3 Early History: The 1940s and 1950s Operating systems have evolved over the last 60 years through several distinct phases or generations that correspond roughly to the decades (see the Operating Systems Thinking feature, Innovation). 3 In the 1940s, the earliest electronic digital computers did not include operating systems.4, 5, 6 Machines of the time were so primitive that programmers often entered their machine-language programs one bit at a time on rows of mechanical switches. Eventually, programmers entered their machine-language programs on punched cards. Then, assembly languages—which used English-like abbreviations to represent the basic operations of the computer— were developed to speed the programming process. General Motors Research Laboratories implemented the first operating system in the early 1950s for its IBM 701 computer. 7 The systems of the 1950s generally executed only one job at a time, using techniques that smoothed the transition between jobs to obtain maximum utilization of the computer system.8 A job constituted the set of program instructions corresponding to a particular computational task, such as payroll or inventory. Jobs typically executed without user input for minutes, hours or days. These early computers were called single-stream batch-pro- 8 Introduction to Operating Systems cessing systems, because programs and data were submitted in groups or batches by loading them consecutively onto tape or disk. A job stream processor read the job control language statements (that defined each job) and facilitated the setup of the next job. When the current job terminated, the job stream reader read in the control-language statements for the next job and performed appropriate housekeeping chores to ease the transition to the next job. Although operating systems of the 1950s reduced interjob transition times, programmers often were required to directly control system resources such as memory and input/output devices. This was slow, difficult and tedious work. Further, these early systems required that an entire program be loaded into memory for the program to run. This limited programmers to creating small programs with limited capabilities.9 Self Review 1. Why were assembly languages developed? 2. What limited the size and capabilities of programs in the 1950s? Ans: 1) Assembly languages were developed to speed the programming process. They enabled programmers to specify instructions as English-like abbreviations that were easier for humans to work with than machine-language instructions. 2) The entire program had to be loaded into memory to execute. Because memory was relatively expensive, the amount of memory available to those computers was small. Operating Systems Thinking Innovation Innovation is a fundamental challenge for operating systems designers. If we are going to make the massive investment required to produce new operating systems or new versions of existing operating systems, we must constantly be evaluating new technologies, new applications of computing and communications and new thinking about how systems should be built. We have provided thousands of citations and hundreds of Web resources for you to do additional readings on topics that are of interest to you. You should consider belonging to professional organizations like the ACM (www.acm.org), the IEEE (www.ieee.org) and USENIX (www.usenix.org) that publish journals on the latest research and development efforts in the computer field. You should access the Web frequently to follow important developments in the field. There is always a high degree of risk when innovating, but the rewards can be substantial. 1.4 The 1960s 9 1.4 The 1960s The systems of the 1960s were also batch-processing systems, but they used the computer's resources more efficiently by running several jobs at once. Systems included many peripheral devices such as card readers, card punches, printers, tape drives and disk drives. Any one job rarely used all the system's resources efficiently. A typical job would use the processor for a certain period of time before performing an input/output (I/O) operation on one of the system's peripheral devices. At this point, the processor would remain idle while the job waited for the I/O operation to complete. The systems of the 1960s improved resource utilization by allowing one job to use the processor while other jobs used peripheral devices. In fact, running a mixture of diverse jobs—some jobs that mainly used the processor (called processorbound jobs or compute-bound jobs) and some jobs that mainly used peripheral devices (called I/O-bound jobs)—appeared to be the best way to optimize resource utilization. With these observations in mind, operating systems designers developed multiprogramming systems that managed several jobs at once.10, 11, 12 In a multiprogramming environment, the operating system rapidly switches the processor from job to job, keeping several jobs advancing while also keeping peripheral devices in use. A system's degree of multiprogramming (also called its level of multiprogramming) indicates how many jobs can be managed at once. Thus, operating systems evolved from managing one job to managing several jobs at a time. In multiprogrammed computing systems, resource sharing is one of the primary goals. When resources are shared among a set of processes, each process maintaining exclusive control over particular resources allocated to it, a process may be made to wait for a resource that never becomes available. If this occurs, that process will be unable to complete its task, perhaps requiring the user to restart it, losing all work that the process had accomplished to that point. In Chapter 7, Deadlock and Indefinite Postponement, we discuss how operating systems can deal with such problems. Normally, users of the 1960s were not present at the computing facility when their jobs were run. Jobs were submitted on punched cards or computer tapes and remained on input tables until the system's human operator could load them into the computer for execution. Often, a user's job would sit for hours or even days before it could be processed. The slightest error in a program, even a missing period or comma, would "bomb" the job, at which point the (often frustrated) user would correct the error, resubmit the job and once again wait hours or days for the next attempt at execution. Software development in that environment was painstakingly slow. In 1964, IBM announced its System/360 family of computers ("360" refers to all points on a compass to denote universal applicability).13, 14, 15, 16 The various 360 computer models were designed to be hardware compatible, to use the OS/360 operating system and to offer greater computer power as the user moved upward in the series.17 Over the years, IBM evolved its 360 architecture to the 370 series18, 19 and, more recently, the 390 series20 and the zSeries.21 10 Introduction to Operating Systems More advanced operating systems were developed to service multiple interactive users at once. Interactive users communicate with their jobs during execution. In the 1960s, users interacted with the computer via "dumb terminals" (i.e., devices that supplied a user interface but no processor power) which were online (i.e., directly attached to the computer via an active connection). Because the user was present and interacting with it, the computer system needed to respond quickly to user requests; otherwise, user productivity could suffer. As we discuss in the Operating Systems Thinking feature, Relative Value of Human and Computer Resources, increased productivity has become an important goal for computers because human resources are extremely expensive compared to computer resources. Timesharing systems were developed to support simultaneous interactive users.22 Many of the timesharing systems of the 1960s were multimode systems that supported batch-processing as well as real-time applications (such as industrial process control systems).23 Real-time systems attempt to supply a response within a certain bounded time period. For example, a measurement from a petroleum refinery indicating that temperatures are too high might demand immediate attention to Operating Systems Thinking Relative Value of Human and Computer Resources In 1965, reasonably experienced programmers were earning about $4 per hour. Computer time on mainframe computers (which were far less powerful than today's desktop machines) was commonly rented for $500 or more per hour—and that was in 1965 dollars which, because of inflation, would be comparable to thousands of dollars in today's currency! Today, you can buy a top-of-the-line, enormously powerful desktop computer for what it cost to rent a far less powerful mainframe computer for one hour 40 years ago! As the cost of computing has plummeted, the cost of man-hours has risen to the point that today human resources are far more expensive than computing resources. Computer hardware, operating systems and software applications are all designed to leverage people's time, to help improve efficiency and productivity. A classic example of this was the advent of timesharing systems in the 1960s in which these interactive systems (with almost immediate response times) often enabled programmers to become far more productive than was possible w i t h the batch-processing systems response times of hours or even days. Another classic example was the advent of the graphical user interface (GUI) originally developed at the Xerox Palo Alto Research Center (PARC) in the 1970s. With cheaper and more powerful computing, and with the relative cost of people-time rising rapidly compared to that of cornputing, operating systems designers must provide capabilities that favor the human over the machine, exactly the opposite of what early operating systems did. 1.4 avert an explosion. The resources of a real-time system are often heavily underutilized—it is more important for such systems to respond quickly than it is for them to use their resources efficiently. Servicing both batch and real-time jobs meant that operating systems had to distinguish between types of users and provide each with an appropriate level of service. Batch-processing jobs could suffer reasonable delays, whereas interactive applications demanded a higher level of service and real-time systems demanded extremely high levels of service. The key timesharing systems development efforts of this period included the CTSS (Compatible Time-Sharing System)24, 25 developed by MIT, the TSS (Time Sharing System) 26 developed by IBM, the Multics system27 developed at MIT, GE and Bell Laboratories as the successor to CTSS and the CP/CMS (Control Program/Conversational Monitor System)—which eventually evolved into IBM's VM (Virtual Machine) operating system—developed by IBM's Cambridge Scientific Center.28, 29 These systems were designed to perform basic interactive computing tasks for individuals, but their real value proved to be the manner in which they shared programs and data and demonstrated the value of interactive computing in program development environments. The designers of the Multics system were the first to use the term process to describe a program in execution in the context of operating systems. In many cases, users submitted jobs containing multiple processes that could execute concurrently. In Chapter 3, Process Concepts, we discuss how multiprogrammed operating systems manage multiple processes at once. In general, concurrent processes execute independently, but multiprogrammed systems enable multiple processes to cooperate to perform a common task. In Chapter 5, Asynchronous Concurrent Execution, and Chapter 6, Concurrent Programming, we discuss how processes coordinate and synchronize activities and how operating systems support this capability. We show many examples of concurrent programs, some expressed generally in pseudocode and some in the popular Java™ programming language. Turnaround time —the time between submission of a job and the return of its results —was reduced to minutes or even seconds. The programmer no longer needed to wait hours or days to correct even the simplest errors. The programmer could enter a program, compile it, receive a list of syntax errors, correct them immediately, recompile and continue this cycle until the program was free of syntax errors. Then the program could be executed, debugged, corrected and completed with similar time savings. The value of timesharing systems in support of program development was demonstrated when MIT, GE and Bell Laboratories used the CTSS system to develop its own successor, Multics. Multics was notable for being the first major operating system written primarily in a high-level language (EPL—modeled after IBM's PL/1) instead of an assembly language. The designers of UNIX learned from this experience; they created the high-level language C specifically to implement UNIX. A family of UNIX-based operating systems, including Linux and Berkeley The 1960s 11 12 Introduction to Operating Systems Software Distribution (BSD) UNIX, have evolved from the original system created by Dennis Ritchie and Ken Thompson at Bell Laboratories in the late 1960s (see the Biographical Note, Ken Thompson and Dennis Ritchie). TSS, Multics and CP/CMS all incorporated virtual memory, which we discuss in detail in Chapter 10, Virtual Memory Organization, and Chapter 11, Virtual Memory Management. In systems with virtual memory, programs are able to address more memory locations than are actually provided in main memory, also called real memory or physical memory.30-31 (Real memory is discussed in Chapter 9, Real Memory Organization and Management.) Virtual memory systems help remove much of the burden of memory management from programmers, freeing them to concentrate on application development. Biographical Note Ken Thompson and Dennis Ritchie Ken Thompson and Dennis Ritchie are well known in the field of operating systems for their development of the UNIX operating system and the C programming language. They have received several awards and recognition for their accomplishments, including the ACM Turing Award, the National Medal of Technology, the NEC C&C Prize, the IEEE Emmanuel Piore Award, the IEEE Hamming Medal, induction into the United States National Academy of Engineering and the Bell Labs National Fellowship. 32 Ken Thompson attended the University of California at Berkeley, where he earned a B.S. and M.S. in Computer Science, graduating in 1966.33 After college Thompson worked at Bell Labs, where he eventually joined Den- nis Ritchie on the Multics project. 34 While working on that project, Thompson created the B language that led to Ritchie's C language. 35 The Multics project eventually led to the creation of the UNIX operating system in 1969. Thompson continued to develop UNIX through the early 1970s, rewriting it in Ritchie's C programming language. 36 After Thompson completed UNIX, he made news again in 1980 with Belle. Belle was a chess-playing computer designed by Thompson and Joe Condon that won the World Computing Chess Championship. Thompson worked as a professor at the University of California at Berkeley and at the University of Sydney, Australia. He continued to work at Bell Labs until he retired in 2000. 37 Dennis Ritchie attended Harvard University, earning a Bachelor's degree in Physics and a Ph.D. in Mathematics. Ritchie went on to work at Bell Labs, where he joined Thompson on the Multics project in 1968. Ritchie is most recognized for his C language, which he completed in 1972.38 Ritchie added some extra capabilities to Thompson's B language and changed the syntax to make it easier to use. Ritchie still works for Bell Labs and continues to work with operating systems.39 Within the past 10 years he has created t w o new operating systerns. Plan 9 and Inferno. 40 The Plan 9 system is designed for communication and production quality. 4 1 lnferno is a system intended for advanced networking. 4 2 1.5 The 1970s 13 Once loaded into main memory, programs could execute quickly; however, main memory was far too expensive to contain large numbers of programs at once. Before the 1960s, jobs were largely loaded into memory using punched cards or tape, a tedious and time-consuming task, during which the system could not be used to execute jobs. The systems of the 1960s incorporated devices that reduced system idle time by storing large amounts of rewritable data on relatively inexpensive magnetic storage media such as tapes, disks and drums. Although hard disks enabled relatively fast access to programs and data compared to tape, they were significantly slower than main memory. In Chapter 12, Disk Performance Optimization, we discuss how operating systems can manage disk input/output requests to improve performance. In Chapter 13, File and Database Systems, we discuss how operating systems organize data into named collections called files and manage space on storage devices such as disks. We also discuss how operating systems protect data from access by unauthorized users and prevent data from being lost when system failures or other catastrophic events occur. Self Review 1. How did interactive computing and its improvement in turnaround time increase programmer productivity? 2. What new concept did TSS, Multics and CP/CMS all incorporate? Why was it so helpful for programmers? Ans: 1) The time between submission of a job and the return of its results was reduced from hours or days to minutes or even seconds. This enabled programmers to interactively enter, compile and edit programs until their syntax errors were eliminated, then use a similar cycle to test and debug their programs. 2) TSS, Multics, and CP/CMS all incorporated virtual memory. Virtual memory allows applications access to more memory than is physically available on the system. This allows programmers to develop larger, more powerful applications. Also, virtual memory systems remove much of the memory management burden from the programmer. 1.5 The 1970s The systems of the 1970s were primarily multimode multiprogramming systems that supported batch processing, timesharing and real-time applications. Personal computing was in its incipient stages, fostered by early and continuing developments in microprocessor technology.43 The experimental timesharing systems of the 1960s evolved into solid commercial products in the 1970s. Communications between computer systems throughout the United States increased as the Department of Defense's TCP/IP communications standards became widely used—especially in military and university computing environments. 44 ' 4546 Communication in local area networks (LANs) was made practical and economical by the Ethernet standard developed at Xerox's Palo Alto Research Center (PARC). 4 7 4 8 In Chapter 16, Introduction to Networking, we discuss TCP/IP, Ethernet and fundamental networking concepts. 14 Introduction to Operating Systems Security problems increased with the growing volumes of information passing over vulnerable communications lines (see the Anecdote, Abraham Lincoln's Technology Caution). Encryption received much attention—it became necessary to encode proprietary or private data so that, even if the data was compromised, it was of no value to anyone other than the intended receivers. In Chapter 19, Security, we discuss how operating systems secure sensitive information from unauthorized access. During the 1970s, operating systems grew to encompass networking and security capabilities and continued to improve in performance to meet commercial demands. The personal computing revolution began in the late 1970s with such systems as the Apple II, and exploded in the 1980s. Self Review 1. What developments in the 1970s improved communication between computer systems? 2. What new problem was introduced by the increased communication between computers? How was this problem addressed? Ans: 1) The DoD's TCP/IP standards became widely used in network communications— primarily in university and military computing environments. Also, Xerox's PARC developed the Ethernet standard, which made relatively high-speed local area networks (LANs) practical and economical. 2) Communication between computers introduced security problems because data was sent over vulnerable communication lines. Encryption was employed to make data unreadable to anyone other than the intended recipient. Anecdote Abraham Lincoln's Technology Caution The story goes that during the Civil War one of President Lin- tlefields for this wonderful new technology called the telegraph. light!" The older and wiser President Lincoln looked d o w n at the coin's young lieutenants came running up to him eager to speak Do you know what that means, Mr. President?" "No lieutenant, lieutenant and said calmly, "Yes, lieutenant, but we'll also be able w i t h the President. "What is it, what does it mean?" "Mr. Presi- to make wrong decisions at the lieutenant?" "Mr. President, Mr. dent, it means we'll be able to speed of light!" President, we are wiring the bat- make decisions at the speed of Lessons to operating systems cons. You will inevitably spend a making things happen faster may designers: Every new technology great deal of your time concerned have unpleasant consequences, you will evaluate has its pros and with performance issues. But, 16 The 1980s 15 1.6 The 1980s The 1980s was the decade of the personal computer and the workstation. 49 Microprocessor technology evolved to the point where high-end desktop computers called workstations could be built that were as powerful as the mainframes of a decade earlier. The IBM Personal Computer released in 1981 and the Apple Macintosh personal computer released in 1984 made it possible for individuals and small businesses to have their own dedicated computers. Communication facilities could be used to transmit data quickly and economically between systems. Rather than bringing data to a central, large-scale computer installation for processing, computing was distributed to the sites at which it was needed. Software such as spreadsheet programs, word processors, database packages and graphics packages helped drive the personal computing revolution by creating demand from businesses that could use these products to increase their productivity. Personal computers proved to be relatively easy to learn and use, partially because of graphical user interfaces (GUI) that used graphical symbols such as windows, icons and menus to facilitate user interaction with programs. Xerox's Palo Alto Research Center (PARC) developed the mouse and GUI (for more on the origins of the mouse, see the Biographical Note, Doug Engelbart); Apple's release of the Macintosh personal computer in 1984 popularized their use. In Macintosh computers, the GUI was embedded in the operating system so that all applications would have a similar look and feel.50 Once familiar with the Macintosh GUI, the user could learn to use new applications faster. As technology costs declined, transferring information between computers in computer networks became more economical and practical. Electronic mail, file transfer and remote database access applications proliferated. Distributed computing (i.e., using multiple independent computers to perform a common task) became widespread under the client/server model. Clients are user computers that request various services; servers are computers that perform the requested services. Servers often are dedicated to one type of task, such as rendering graphics, managing databases or serving Web pages. The software engineering field continued to evolve, a major thrust coming from the United States government aimed at providing tighter control of Department of Defense software projects.51 Some goals of the initiative included realized code reusability and the early construction of prototypes so developers and users could suggest modifications early in the software design process.52 Self review 1. What aspect of personal computers, popularized by the Apple Macintosh, made them especially easy to learn and use? 2. (T/F) A server cannot be a client. Ans: 1) Graphical User Interfaces (GUIs) facilitated personal computer use by providing an easy-to-use, uniform interface to every application. This enabled users to learn new applications faster. 2) False. A computer can be a client and server. For example, a Web server can 16 Introduction to Operating Systems be both a client and server. When users request a Web page, it is a server; if the server then requests information from a database system, it becomes a client of the database system. 1.7 History of the Internet and World Wide Web In the late 1960s ARPA —the Advanced Research Projects Agency of the Department of Defense rolled out the blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions. They were to be connected with communications lines operating at a then-stunning 56 kilobits per second (Kbps) —1 Kbps is equal to 1,000 bits per second —at a time when most people (of the few who could be) were connecting over telephone lines to computers at a rate of 110 bits per second. HMD vividly recalls the excitement at Biographical Note Doug Engelbart Doug Engelbart invented the San Francisco, Engelbart and his NLS system was to create a system computer mouse and was one of coworkers displayed their com- that could help people solve prob- the primary designers of the origi- puter system, NLS (oNLine System) lems faster and enhance i n t e r - nal graphical displays and w i n - which featured Engelbart's com- gence. Engelbart founded the dows. puter mouse and a graphical Bootstrap Institute to foster interface w i t h windows. 5 6 This worldwide awareness of his mis- original mouse, called an X-Y Posi- sion. Bootstrapping, according to Engelbart's background was in electronics. During World War II he worked as an electronics tech- tion Indicator for a Display Sys- Engelbart, is the idea of improv- nician on a variety of systems tern, had only one button. 5 7 The ing one's methods of improve- including RADAR and SONAR.53 mouse had t w o wheels on the ment. He believes this is the best After leaving the military, he bottom, one horizontal and one way to improve human intelli- went back to Oregon State to vertical, to detect movement. 58 gence. 59 complete a degree in Electrical The mouse and the graphical w i n - Engineering in 1948.54 He went dows were interdependent. The Today, Engelbart is still working w i t h the Bootstrap Insti- on to receive his Ph.D. from the mouse made it significantly easier tute. He has received recognition University of California at Berke- to switch between windows, and for his work including the Lemel- ley, then took a job at the Stan- w i t h o u t windows the mouse was son-MIT Prize, the National Medal f o r d Research Institute (SRI), not as useful. of Technology and induction into where he gained his first experi- Engelbart has dedicated his ence w i t h computers. 55 In 1968, at life to augmenting human intel- the Joint Computer Conference in lect. His original idea behind the the National Inventors Hall of Fame. 60 1.7 History of the Internet and World Wide Web 17 that conference. Researchers at Harvard talked about communicating with the Univac 1108 "supercomputer" across the country at the University of Utah to handle the massive computations related to their computer graphics research. Academic research was about to take a giant leap forward. Shortly after this conference, ARPA proceeded to implement what quickly became called the ARPAnet—the grandparent of today's Internet. Although the ARPAnet did enable researchers to network their computers, its chief benefit proved to be its capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on the Internet today, with e-mail, instant messaging and file transfer facilitating communications among hundreds of millions of people worldwide and growing rapidly. The ARPAnet was designed to operate without centralized control. This meant that if a portion of the network should fail, the remaining working portions would still be able to route data packets from senders to receivers over alternative paths. The protocols (i.e., sets of rules) for communicating over the ARPAnet became known as the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP was used to manage communication between applications. The protocols ensured that messages were routed properly from sender to receiver and that those messages arrived intact. The advent of TCP/IP promoted worldwide computing growth. Initially, Internet use was limited to universities and research institutions; later, the military adopted the technology. Eventually, the government decided to allow access to the Internet for commercial purposes. This decision led to some concern among the research and military communities—it was felt that response times would suffer as "the Net" became saturated with users. In fact, the opposite occurred. Businesses rapidly realized that they could use the Internet to tune their operations and to offer new and better services to their clients. Companies spent vast amounts of money to develop and enhance their Internet presence. This generated intense competition among communications carriers, hardware suppliers and software suppliers to meet the increased infrastructure demand. The result is that bandwidth (i.e., the informationcarrying capacity of communications lines) on the Internet has increased tremendously, and hardware and communications costs

have plummeted. The World Wide Web (WWW) allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animation, audio or video) on almost any subject. Although the Internet was developed more than three decades ago, the introduction of the World Wide Web (WWW) was a relatively recent event. In 1989, Tim Berners-Lee of CERN (the European Center for Nuclear Research) began to develop a technology for sharing information via hyperlinked text documents (see the Biographical Note, Tim Berners-Lee). To implement this new technology, Berners-Lee created the HyperText Markup Language (HTML). Berners-Lee also implemented the Hypertext Transfer Protocol (HTTP) to form the communications backbone of his new hypertext information system, which he called the World Wide Web. 18 Introduction to Operating Systems Surely, historians will list the Internet and the World Wide Web among the most important and profound creations of humankind. In the past, most computer applications ran on "stand-alone" computers (computers that were not connected to one another). Today's applications can be written to communicate among the world's hundreds of millions of computers. The Internet and World Wide Web merge computing and communications technologies, expediting and simplifying our work. They make information instantly and conveniently accessible to large numbers of people. They enable individuals and small businesses to achieve worldwide exposure. They are changing the way we do business and conduct our personal lives. And they are changing the way we think of building operating systems. Today's operating systems provide GUIs that enable users to "access the world" over the Internet and the Web as seamlessly as accessing the local system. The operating systems of the 1980s were concerned primarily with managing resources on the local computer. Today's distributed operating systems may utilize resources on computers worldwide. This creates many interesting challenges that we discuss throughout the book, especially in Chapters 16-19, which examine networking, distributed computing and security. Biographical Note Tim Berners-Lee The World Wide Web was fellowship at the European Ceninvented by Tim Berners-Lee in ter for Nuclear Research (CERN) in 1990. The Web allows computer 1984, where he gained experience users to locate and view multimein communication software for dia-based documents (i.e., docureal-time networked 62, 63 ments w i t h text, graphics, systems.61, animation, audio or video) on Berners-Lee invented HTTP almost any subject. (the HyperText Transfer Protocol), Until 1993, Berners-Lee individually managed changes and suggestions for HTTP and HTML, sent from the early Web users. By 1994 the Web community had grown large enough that he started the World Wide Web Consortium (W3C; www.w3.org) to Berners-Lee graduated from Queen's College at Oxford University with a degree in Physics in 1976. In 1980 he wrote a program called Enquire, which used hypertext links to help him quickly navigate the numerous documents in a large project. He entered into a monitor and establish Web technology standards. 67 As director of the organization, he actively promotes the principle of freely available information accessed by open technologies. 68 HTML (Hypertext Markup Language) and the first World Wide Web server and browser in 1989, while working at CERN.64, 65 He intended the Web to be a mechanism for open, available access to all shared knowledge and experience. 66 1.8 The 1990s 19 Self Review 1. How did the ARPAnet differ from traditional computer networks? What was its primary benefit? 2. What creations did Berners-Lee develop to facilitate data sharing over the Internet? Ans: 1) The ARPAnet was decentralized, so the network continued to be able to pass information even if portions of the network failed. The primary benefit of the ARPAnet was its capability for quick and easy communication via e-mail. 2) Berners-Lee developed the HyperText Markup Language (HTML) and the Hypertext Transfer Protocol (HTTP), making possible the World Wide Web. 1.8 The1990s Hardware performance continued to improve exponentially in the 1990s.69 By the end of the decade, a typical personal computer could execute several hundred million instructions per second (MIPS) and store over a gigabyte of information on a hard disk; some supercomputers could execute over a trillion operations per second.70 Inexpensive processing power and storage enabled users to execute large, complex programs on personal computers and enabled small- to mid-size companies to use these economical machines for the extensive database and processing jobs that were once delegated to mainframe systems. Falling technology costs also led to an increase in the number of home computers, which were used both for work and for entertainment. In the 1990s, the creation of the World Wide Web led to an explosion in the popularity of distributed computing. Originally, operating systems performed isolated resource management inside a single computer. With the creation of the World Wide Web and increasingly fast Internet connections, distributed computing became commonplace among personal computers. Users could request data stored at remote locations or request that programs run on distant processors. Large organizations could use distributed multiprocessors (i.e., networks of computers containing more than one processor) to scale resources and increase efficiency.71 Distributed applications, however, were still limited by the fact that communication over a network occurred at relatively slow speeds compared to the internal processing speeds of individual computers. Distributed computing is discussed in detail in Chapter 17, Introduction to Distributed Systems, and Chapter 18, Distributed Systems and Web Services. As demand for Internet connections grew, operating system support for networking tasks became standard. Users at home and in organizations increased productivity by accessing the resources on networks of computers. However, increased connectivity led to a proliferation of computer security threats. Operating system designers developed techniques to protect computers from these malicious attacks. Ever more sophisticated security threats continued to challenge the computer industry's ability to counter such attacks. 20 Introduction to Operating Systems Microsoft Corporation became dominant in the 1990s. In 1981, Microsoft released the first version of its DOS operating system for the IBM personal computer. In the mid-1980s, Microsoft developed the Windows operating system, a graphical user interface built on top of the DOS operating system. Microsoft released Windows 3.0 in 1990; this new version featured a user-friendly interface and rich functionality. The Windows operating system became incredibly popular after the 1993 release of Windows 3.1, whose successors, Windows 95 and Windows 98, virtually cornered the desktop operating system market by the late 90s. These operating systems, which borrowed from many concepts (such as icons, menus and windows) popularized by early Macintosh operating systems, enabled users to navigate multiple concurrent applications with ease. Microsoft also entered the corporate operating system market with the 1993 release of Windows NT, which quickly became the operating system of choice for corporate workstations.72 Windows XP, which is based on the Windows NT operating system, is discussed in Chapter 21, Case Study: Windows XP. Object Technology Object technology became popular in many areas of computing, as the number of applications written in object-oriented programming languages, such as C++ or Java, increased steadily. Object concepts also facilitated new approaches to computing. Each software object encapsulates a set of attributes and a set of actions. This allows applications to be built with components that can be reused in many applications, reducing software development time. In object-oriented operating systems (OOOS), objects represent components of the operating system and system resources. 73 Object-oriented concepts such as inheritance and interfaces were exploited to create modular operating systems that were easier to maintain and extend than operating systems built with previous techniques. Modularity facilitates operating system support to new and different architectures. The demand for object integration across multiple platforms and languages led to support for objects in programming languages such as Sun's Java and Microsoft's .NET languages (e.g., Visual Basic .NET, Visual C++ .NET and C#). Open-Source Movement Another development in the computing community (particularly in the area of operating systems) during the 1990s was the movement toward open-source software. Most software is created by writing source code in a high-level programming language. However, most commercial software is sold as object code (also called machine code or binaries)—the compiled source code that computers can understand. The source code is not included, enabling vendors to hide proprietary information and programming techniques. However, free and open-source software became increasingly common in the 1990s. Open-source software is distributed with the source code, allowing individuals to examine and modify the software before compiling and executing it. For example, the Linux operating system and the Apache Web 1.8 The 1990s 21 server, both of which are free and open source, were downloaded and installed by millions of users during the 1990s, and the number of downloads is increasing rapidly in the new millennium.74 Linux, created by Linus Torvalds (see the Biographical Note, Linus Torvalds), is discussed in Chapter 20, Case Study: Linux. In the 1980s, Richard Stallman, a software developer at MIT, launched a project to recreate and extend most of the tools for AT&T's UNIX operating system and to make the code available at no charge. Stallman (see the Biographical Note, Richard Stallman) founded the Free Software Foundation and created the GNU project—which is a recursive name that stands for "GNU's Not UNIX" — because he disagreed with the concept of selling the permission to use software.75 He believed that granting users the freedom to modify and distribute software would lead to better software, driven by user needs instead of personal or corporate profit. When Linus Torvalds created the original version of the Linux operating system, he employed many of the tools published by GNU for free under the General Public License (GPL). The GPL, published online at www.gnu.org/1icenses/ gpl.html, specifies that anyone can freely modify and redistribute software under its license, provided that the modifications are clearly indicated and any derivative of the software is also distributed under the GPL. 76 Although most GPL-licensed software is available free of charge, the GPL requires only that its software be free in the sense that users can freely modify and redistribute it. Therefore, vendors can charge a fee for providing GPL-licensed software and its source code, but cannot Biographical Note Linus Torvalds Linus Torvalds was born in 1969 in Public License (GPL)80 as openTorvalds is currently a fellow i Helsinki, Finland. As a child he source code and gladly accepted of the Open Source Development taught himself how to program additions, corrections and free Labs (OSDL), which funds his fullby playing w i t h a Commodore programs f r o m other programtime work on the kernel. He con82 VIC-20. In 1988 he entered the mers.81, By 1994 Linux had tinues to lead the open-source University of Helsinki to study accrued enough applications to Linux project, managing changes computer science. While there, he be a complete, usable operating and releasing new versions of the 87 wrote a UNIX clone based on Prosystem, and version 1.0 was kernel. 86, Linux has become one 83 fessor Andrew Tanenbaum's Minix released. Programmers and proof the largest and best-known to run on his new PC.77, 78 In 1991 fessors using UNIX on large sysopen-source developments in cornhe completed the first version of terns liked Linux because it puting history and has become the basic Linux kernel, which ran brought the features and power particularly successful in the server 79 on the Intel 80386 processor. He of UNIX to inexpensive desktop market. Linux is discussed in distributed Linux under the GNU systems for free. 84, 85 Chapter 20, Case Study: Linux. 22 Introduction to Operating Systems prevent end users from modifying and redistributing them. In the late 90s, the Open Source Initiative (OSI) was founded to protect open-source software and promote the benefits of open-source programming (see www.opensource.org). Open-source software facilitates enhancements to software products by permitting anyone in the developer community to test, debug and enhance applications. This increases the chance that subtle bugs, which could otherwise be security risks or logic errors, are caught and fixed. Also, individuals and corporations can modify the source to create custom software that meets the needs of a particular environment. Many open-source software vendors remain profitable by charging individuals and organizations for technical support and customizing software.88 Though most systems in the 1990s still ran proprietary operating systems, such as IBM mainframe operating systems, UNIX systems, Apple's Macintosh and Biographical Note Richard Stallman Richard Stallman was the original developer of the GNU project, started in the 1980s to create free software. Stallman graduated from Harvard in 1974 w i t h a degree in Physics.89 While at Harvard, he worked at the MIT Artificial Intelligence Lab. After graduating, he continued at the lab, where he and his colleagues worked with shared software. 90 The idea was that someone receiving someone else's executable program would also receive its source code. This was advantageous because a programmer could add more functionality to someone else's program. 9 1 Stallman's specific job was to modify and improve the ITS operating system the lab used. However, as the 1980s arrived, there was little shared software. 92 The lab's new operating system was not shared. Stallman became frustrated w i t h operating systems, drivers, and the like, that he could no longer modify. 93 In 1984 he left the MIT Artificial Intelligence Lab to work on a new shared operating systern, which he called GNU (GNU's Not UNIX).94 As interest in Stallman's GNU project grew, he created the Free Software Foundation (FSF) in 1985 to promote free software and continue to develop the GNU operating system.95 Stallman and his associates at FSF created a number of GNU programs, including GNU Emacs (a text editor), GCC (a C compiler) and GDB (a debugger), to name a few. 96 In 1992 Stallman used the Linux ker- nel to complete his system. The system, known as GNU/Linux, is a fully functional operating system and includes a variety of programs. 97 Stallman has received numerous awards and recognition for his work, including the Grace Hopper Award, MacArthur Foundation Fellowship, the Electric Frontier Foundation Pioneer Award, the Yuri Rubinksi Award, the Takeda Award, election to the National Academy of Engineering and t w o honorary doctorates f r o m the Institute of Technology in Sweden and the University of Glasgow. Stallman continues to promote the free software cause and speaks about free software all over the world. 9 8 1.9 Microsoft's Windows, open-source operating systems, such as Linux, FreeBSD and OpenBSD, became viable competition. In the future, they will undoubtedly continue to gain ground on proprietary solutions as a result of product improvement, industry standardization, interoperability, product customization and cost savings. In the 1990s, operating systems became increasingly user friendly. The GUI features that Apple built into its Macintosh operating system in the 1980s became more sophisticated in the 1990s. "Plug-and-play" capabilities were built into operating systems, enabling users to add and remove hardware components dynamically without manually reconfiguring the operating system. Operating systems also maintained user profiles—serving authentication needs and enabling per-user customization of the operating system interface. Self Review 1. How did object-oriented technology affect operating systems? 2. What are some of the benefits of open-source development? Ans: 1) Operating systems designers could reuse objects when developing new components. Increased modularity due to object-oriented technology facilitated operating system support for new and different architectures. 2) Open-source software can be viewed and modified by anyone in the software development community. Because these people constantly test, debug and use the software, there is a greater chance that bugs will be found and fixed. Also, open-source software enables users and organizations to modify a program to meet their particular needs. 1.9 2000 and Beyond In the current decade, middleware, which is software that links two separate applications (often over a network), has become vital as applications are published on the World Wide Web and consumers use them via affordable, high-speed Internet connections over cable television lines and digital subscriber lines (DSL). Middleware is common in Web applications, in which a Web server (the application that sends data to the user's Web browser) must generate content to satisfy a user's request with the help of a database. The middleware acts as a courier to pass messages between the Web server and the database, simplifying communication between multiple different architectures. Web services encompass a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. A Web service communicates over a network to supply a specific set of operations that other applications can invoke. The data is passed back and forth using standard protocols such as HTTP, the same protocol used to transfer ordinary Web pages. Web services operate using open, text-based standards that enable components written in different languages and on different platforms to communicate. They are ready-touse pieces of software on the Internet. Web services will help drive the shift toward true distributed computing. For example, the online retailer Amazon.com allows developers to build online stores that search Amazon's product databases and display detailed product information 2000 and Beyond 23 24 Introduction to Operating Systems via Amazon.com Web Services (www.amazon.com/gp/aws/landing.html). The Google search engine also can be integrated with other functionality through the Google Web APIs (www.google.com/apis), which connect to Google's indices of Web sites using Web services. We discuss Web services in more detail in Chapter 18, Distributed Systems and Web Services. Multiprocessor and network architectures are creating numerous opportunities for research and development of new hardware and software design techniques. Sequential programming languages that specify one computation at a time are now complemented by concurrent programming languages, such as Java, that enable the specification of parallel computations; in Java the units of parallel computing are specified via threads. We discuss threads and the technique of multithreading in Chapter 4, Thread Concepts. An increasing number of systems exhibit massive parallelism; they have large numbers of processors so that many independent parts of computations can be performed in parallel. This is dramatically different in concept from the sequential computing of the past 60 years; there are significant and challenging problems in developing the software appropriate for dealing with such parallelism. We discuss parallel computing architectures in Chapter 15, Multiprocessor Management. Operating systems are standardizing user and application interfaces so that they are easier to use and support a greater number of programs. Microsoft has already merged the consumer and professional lines of its Windows operating system into Windows XP. In its next operating system (code-named Longhorn). Microsoft plans to integrate the formats of different types of files. This will, for example, allow users to search their systems for all files (documents, spreadsheets, e-mails, etc.) containing certain keywords. Longhorn will also include an enhanced 3D user interface, improved security and support for recordable digital versatile discs (DVDs). 99, 100 Open-source operating systems, such as Linux, will become more widely used and will employ standard application programming interfaces (APIs) such as the Portable Operating System Interface (POSIX) to improve compatibility with other UNIX-based operating systems. Computing on mobile devices, such as cell phones and PDAs, will become more common as mobile devices are equipped with increasingly powerful processors. Today, these devices are used for such functions as e-mail, Web browsing and digital imaging. Resource-intensive applications, such as full-motion video, will proliferate on these devices. Because mobile device resources are limited by the devices' small size, distributed computing will play an even larger role, as PDAs and cell phones will request increasing amounts of data and processing power from remote computers. Self Review 1. What technologies can be used to bridge the gap between different operating systems? How would these technologies make it possible to execute the same application on multiple platforms? 2. Why is distributed computing useful for computations performed by mobile devices? 1.10 Ans: 1) Virtual machines and operating system emulators bridge the gap between different operating systems. Applications can be written once to use the functionality of the virtual machine or emulator. The virtual machines or emulators can be implemented to hide the representation of the underlying platform from the applications. 2) Distributed computing allows a mobile device to delegate jobs to other machines with more resources. The mobile device, having limited resources and battery life, can request data and processing power from larger computers across a network. 1.10 Application Bases When the IBM Personal Computer (often called simply "the PC") appeared in 1981, it immediately spawned a huge software industry in which independent software vendors (ISVs) were able to market packages for the IBM PC to run under the MS-DOS operating system (IBM's version was called DOS). Operating systems free applications software developers from having to deal with the messy details of manipulating computer hardware to manage memory, perform input/output, deal with communication lines, and so on. The operating system provides a series of application programming interface (API) calls which applications programmers use to accomplish detailed hardware manipulations and other operations. The API provides system calls by which a user program instructs the operating system to do the work; the application developer simply has to know what routines to call to accomplish specific tasks (Fig. 1.1). Note that in Fig. 1.1, the area above the dashed line, user space, indicates software components that are not part of the operating system and cannot directly access the system's physical resources. The area below the dashed line, kernel space, indicates software components that are part of the operating system and have unrestricted access to system resources. We frequently use this convention in our diagrams to indicate the privilege with which software components execute. If an application attempts to misuse system resources, or if the Figure 1.1 | Interaction between applications and the operating system. Application Bases 25 26 Introduction to Operating Systems application attempts to use resources that it has not been granted, the operating system must intervene to prevent the application from damaging the system or interfering with other user applications. If an operating system presents an environment conducive to developing applications quickly and easily, the operating system and the hardware are more likely to be successful in the marketplace. The applications development environment created by MS-DOS encouraged the development of tens of thousands of application software packages. This in turn encouraged users to buy IBM PCs and compatibles. Windows could well have an application base of a hundred thousand applications. Once an application base (i.e., the combination of the hardware and the operating system environment in which applications are developed) is widely established, it becomes extremely difficult to ask users and software developers to convert to a completely new application development environment provided by a dramatically different operating system. Thus, it is likely that new architectures evolving over the next several years will make every effort to support the existing major application bases. 1.11 Operating System Environments This book focuses on operating system concepts related to general-purpose computers with a range of resources, including sizable amounts of main memory, high processor speeds, high-capacity disks, various peripheral devices, and so on. Such computers are typically used as personal computers or as workstations. Many of the concepts that apply to general-purpose computers also apply to high-end Web and database servers, which contain high-performance hardware. Operating systems intended for high-end environments must be designed to support large main memories, special-purpose hardware, and large numbers of processes. We discuss these considerations in Chapter 15, Multiprocessor Management. Embedded systems provide a different operating system design challenge. They are characterized by a small set of specialized resources that provide functionality to devices such as cell phones and PDAs. In embedded environments, efficient resource management is the key to building a successful operating system. Storage is often limited, so the operating system must provide services using a minimal amount of code. Considerations such as power management and the need for userfriendly interfaces create other challenges in embedded operating system design. Real-time systems require that tasks be performed within a particular (often short) time frame. For example, the autopilot feature of an aircraft must constantly adjust speed, altitude and direction. Such actions cannot wait indefinitely—and sometimes cannot wait at all—for other nonessential tasks to complete. Real-time operating systems must enable processes to respond immediately to critical events. Soft realtime systems ensure that real-time tasks execute with high priority, but do not guarantee which, if any, of these tasks will complete on time. Hard real-time systems guarantee that all of their tasks complete on time. We discuss how Linux and Windows XP 1.11 Operating System Environments 27 handle real-time applications in Chapters 20 and 21, respectively. These systems are found in many settings including robotics, avionics and other system control applications. Often, they are used in mission-critical systems, where the system fails to meet its objectives (i.e., mission) if any of its tasks are not successfully completed on time. In mission-critical systems such as those for air traffic control, nuclear reactor monitoring and military command and control, people's lives could be at risk. Business-critical systems, such as Web servers and databases, must consistently meet their objectives. In e-business, this could mean guaranteeing fast response times to users purchasing products over the Internet; in large corporations, it could mean enabling employees to share information efficiently and ensuring that important information is protected from problems such as power failures and disk failures. Unlike mission-critical systems, the business does not necessarily fail if a business-critical system does not always meet its objectives. Some operating systems must manage hardware that may or may not physically exist in the machine. A virtual machine (VM) is a software abstraction of a computer that often executes as a user application on top of the native operating system.101 A virtual machine operating system manages the resources provided by the virtual machine. One application of virtual machines is to allow multiple instances of an operating system to execute concurrently. Another is emulation— using software or hardware that mimics the functionality of hardware or software not present in the system. Virtual machines interface with the hardware in a system via the underlying operating system; other user programs can interact with VMs. A VM can create software components that represent the contents of physical systems —such as processors, memory, communication channels, disks and clocks (Fig. 1.2).102 This allows multiple users to share hardware under the illusion of being serviced by a dedicated machine. By providing this illusion, virtual machines promote portability, the ability for software to run on multiple platforms. The Java Virtual Machine (JVM) is one of the most widely used virtual machines. The JVM is the foundation of the Java platform and allows Java applications to execute on any JVM of the correct version, regardless of the platform on which the JVM is installed. The company VMware Software also provides virtual machines, particularly for the Intel architecture, enabling owners of Intel x86-based computers to run operating systems such as Linux and Windows concurrently on one computer (each virtual machine appears in its own window).103 Virtual machines tend to be less efficient than real machines because they access the hardware indirectly (or simulate hardware that is not actually connected to the computer). Indirect or simulated hardware access increases the number of software instructions required to perform each hardware action.104 Self Review 1. What type of system would a temperature monitor in a nuclear power plant probably be described as? Why? 2. Describe the advantages and disadvantage of virtual machines. 28 Introduction to Operating Systems Figure 1.2 | Schematic if a virtual machine. Ans: 1) A hard real-time system would monitor the temperature in a nuclear power plant to ensure that it is always in an appropriate range, and would notify operators in real time (i.e., instantly) if there was a problem. 2) Virtual machines promote portability by enabling software to run on multiple platforms, but they tend to be less efficient than real machines, because virtual machines must execute software instructions that simulate hardware operations. 1.12 Operating System Components and Goals Computer systems have evolved from early systems containing no operating system, to multiprogramming machines, to timesharing machines, to personal computers and finally to truly distributed systems. As the demand for new features and improved efficiency grew and hardware changed, operating systems evolved to fill new roles. This section describes various core operating system components and explains several goals of operating systems. 1.12.1 Core Operating System Components A user interacts with the operating system via one or more user applications, and often through a special application called a shell, or command interpreter. 105 Most of today's shells are implemented as text-based interfaces that enable the user to issue commands from a keyboard or as GUIs that allow the user to point and click and drag and drop icons to request services from the operating system (e.g., to open an application). For example, Microsoft Windows XP provides a GUI through 1.12 Operating System Components and Goals 29 which users can issue commands; alternatively, the user can open a command prompt window that accepts typed commands. The software that contains the core components of the operating system is referred to as the kernel. Typical operating system core components include: • the process scheduler, which determines when and for how long a process executes on a processor. • the memory manager, which determines when and how memory is allocated to processes and what to do when main memory becomes full. • the I/O manager, which services input and output requests from and to hardware devices, respectively. • the interprocess communication (IPC) manager, which allows processes to communicate with one another. • the file system manager, which organizes named collections of data on storage devices and provides an interface for accessing data on those devices. Almost all modern operating systems support a multiprogrammed environment in which multiple applications can execute concurrently. One of the most fundamental responsibilities of an operating system is to determine which processor executes a process and for how long that process executes. A program may contain several elements that share data and that can be executed concurrently. For example, a Web browser may contain separate components to read a Web page's HTML, retrieve the page's media (e.g., images, text and video) and render the page by laying out its content in the browser window. Such program components, which execute independently but perform their work in a common memory space, are called threads. Threads are discussed in Chapter 4, Thread Concepts. Typically, many processes compete to use the processor. The process scheduler can base its decisions on several criteria, such as importance of a process, its estimated running time, or how long it has waited to obtain the processor. We discuss processor scheduling in Chapter 8, Processor Scheduling. The memory manager allocates memory to the operating system and to processes. To ensure that processes to do not interfere with the operating system or with one another, the memory manager prevents each process from accessing memory that has not been allocated to it. Almost all of today's operating systems support virtual memory, as discussed in Chapters 10 and 11. Another core function of the operating system is to manage the computer's input/output (I/O) devices. Input devices include keyboards, mice, microphones and scanners; output devices include monitors, printers and speakers. Storage devices (e.g., hard disks, rewritable optical discs and tape) and network cards function as both input and output devices. When a process wishes to access an I/O device, it must issue a system call to the operating system. That system call is subsequently handled by a device driver, which is a software component that interacts directly with hardware, often containing device-specific commands and other instructions to perform the requested input/output operations. 30 Introduction to Operating Systems Most computer systems can store data persistently (i.e., after the computer is turned off). Because main memory is often relatively small and loses its data when the power is turned off, persistent secondary storage devices are used, most commonly hard disks. Disk I/O — one of the most common forms of I/O —occurs when a process requests access to information on a disk device. Secondary storage, however, is much slower than processors and main memory. The disk scheduler component of an operating system is responsible for reordering disk I/O requests to maximize performance and minimize the amount of time a process waits for disk I/O. Redundant Array of Independent Disks (RAID) systems attempt to reduce the time a process waits for disk I/O by using multiple disks at once to service I/O requests. We discuss disk scheduling algorithms and RAID systems in Chapter 12, Disk Performance Optimization. Operating systems use file systems to organize and efficiently access named collections of data called files located on storage devices. File system concepts are addressed in Chapter 13, File and Database Systems. Often, processes (or threads) cooperate to accomplish a common goal. Thus, many operating systems provide interprocess communication (IPC) and synchronization mechanisms to simplify such concurrent programming. Interprocess communication enables processes to communicate via messages sent between the processes (and threads); synchronization provides structures that can be used to ensure that processes (and threads) share data properly. Processes and threads are discussed in Chapters 3 through 8. Self Review 1. Which operating system components perform each of the following operations? a. Write to disk. b. Determine which process will run next. c. Determine where in memory a new process should be placed. d. Organize files on disk. e. Enable one process to send data to another. 2. Why is it dangerous to allow users to perform read or write operations to any region of disk at will? Ans: 1) a) I/O manager; b) processor scheduler; c) memory manager; d) file system manager; e) interprocess communication (IPC) manager. 2) It is dangerous because users could accidentally or maliciously overwrite critical data (such as operating system files) or read sensitive information (such as confidential documents) without authorization. 1.12.2 Operating System Goals Users have come to expect certain characteristics of operating systems, such as: • efficiency • robustness 1.12 • scalability • extensibility • portability • security • interactivity • usability Operating System Components and Goals An efficient operating system achieves high throughput and low average turnaround time. Throughput measures the amount of work a processor can complete within a certain time period. Recall that one role of an operating system is to provide services to many applications. An efficient operating system minimizes the time spent providing these services (see the Operating Systems Thinking feature, Performance). A robust operating system is fault tolerant and reliable —the system will not fail due to isolated application or hardware errors, and if it fails, it does so gracefully (i.e., by minimizing loss of work and by preventing damage to the system's hard- Operating Systems Thinking Performance One of the most important goals However, performance is often "in of an operating system is to maxi- the eye of the beholder"—there Memory Management, discusses mize system performance. We are performance conscious in our are many ways to classify operating system performance. For allocating memory to processes to reduce their execution times, everyday lives. We measure our batch-processing systems, through- Chapter 12, Disk Performance cars' gasoline mileage, we record various speed records, professors put is an important measure; for interactive timesharing systems, Optimization, focuses on improving disk performance by reorder- assign grades to students, employees receive performance evalua- fast response times are more important. ing I/O requests. In Chapter 14, Performance and Processor tions from their employers, a corporate executive's performance is measured by company Throughout the book we present many performance improvement techniques. For Design, we discuss evaluating systerns according to several important performance criteria, profits, politicians' performance is measured in frequent polls of their constituents and so on. example, Chapter 8, Processor Scheduling, discusses allocating processor time to processes to Chapters 20, and 21 discuss performance issues in the Linux and Windows XP operating systems. High performance is essential improve system performance as respectively, to successful operating systems. measured by interactivity and throughput. Chapter 11, Virtual 31 32 Introduction to Operating Systems ware). Such an operating system will provide services to each application unless the hardware it relies on fails. A scalable operating system is able to use resources as they are added. If an operating system is not scalable, then it will quickly reach a point where additional resources will not be fully utilized. A scalable operating system can readily adjust its degree of multiprogramming. Scalability is a particularly important attribute of multiprocessor systems —as more processors are added to a system, ideally the processing capacity should increase in proportion to the number of processes, though, in practice, that does not happen. Multiprocessing is discussed in Chapter 15, Multiprocessor Management. An extensible operating system will adapt well to new technologies and provide capabilities to extend the operating system to perform tasks beyond its original design. A portable operating system is designed such that it can operate on many hardware configurations. Application portability is also important, because it is costly to develop applications, so the same application should run on a variety of hardware configurations to reduce development costs. The operating system is crucial to achieving this kind of portability. A secure operating system prevents users and software from accessing services and resources without authorization. Protection refers to the mechanisms that implement the system's security policy. An interactive operating system allows applications to respond quickly to user actions, or events. A usable operating system is one that has the potential to serve a significant user base. These operating systems generally provide an easy-to-use user interface. Operating systems such as Linux, Windows XP and MacOS X are characterized as usable operating systems, because each supports a large set of applications and provides standard user interfaces. Many experimental and academic operating systems do not support a large number of applications or provide userfriendly interfaces and therefore are not considered to be usable. Self Review 1. Which operating system goals correspond to each of the following characteristics? a. Users cannot access services or information without proper authorization. b. The operating system runs on a variety of hardware configurations. c. System performance increases steadily when additional memory and processors are added. d. The operating system supports devices that were not available at the time of its design. e. Hardware failure does not necessarily cause the system to fail. 2. How does device driver support contribute to an operating system's extensibility? ANS: 1) a) security; b) portability; c) scalability; d) extensibility; e) robustness. 2) Device drivers enable developers to add support for hardware that did not exist when the operating system was designed. With each new type of device that is added to a system a corresponding device driver must be installed. 1.13 Operating System Architectures 33 1.13 Operating System Architectures Today's operating systems tend to be complex because they provide many services and support a variety of hardware and software resources (see the Operating Systems Thinking feature, Keep It Simple (KIS) and the Anecdote,). Operating system architectures can help designers manage this complexity by organizing operating system components and specifying the privilege with which each component executes. In the monolithic design, every component of the operating system is contained in the kernel; in the microkernel design, only the essential components are included. In the sections that follow, we survey several important architectures (see the Operating Systems Thinking feature, Architecture). 1.13.1 Monolithic Architecture The monolithic operating system is the earliest and most common operating system architecture. Every component of the operating system is contained in the kernel and can directly communicate with any other (i.e., simply by using function calls). The kernel typically executes with unrestricted access to the computer system (Fig. 1.3). OS/360, VMS and Linux are broadly characterized as monolithic operating systems.106 Direct intercommunication between components makes monolithic operating systems highly efficient. Because monolithic kernels group components together, Operating Systems Thinking Keep It Simple (KIS) Complex systems are costly to trivial to program, but runs slowly see the trade-offs between sim- design, implement, test, debug and maintain. Often, operating systems designers will choose the compared to a more elegant and complicated binary search. Tree data structures can be more com- plicity and complexity. As you read these solutions, you might be inclined to favor certain simplest of several approaches to solving a particular problem. plex to work w i t h than arrays, but make it easier and faster to per- approaches. The systems you work with in the future may Sometimes, though, a more complex approach can yield perfor- form certain types of insertions and deletions. We typically con- demand different approaches, Our philosophy is to present the mance benefits or other improvements that make such an approach worthwhile. Such trade- sider alternate approaches to solving operating systems problems and developing resource pros and cons of the popular approaches to help you prepare to make your own best judgmer offs are common in computing. A management strategies. As you calls in industry, simple linear search of an array is read these discussions, you will 34 Introduction to Operating Systems Figure 1.3 | Monolithic operating System kernel architecture. however it is difficult to isolate the source of bugs and other errors. Further, because all code executes with unrestricted access to the system, systems with monolithic kernels are particularly susceptible to damage from errant or malicious code. Self Review 1. What is the defining characteristic of a monolithic operating system? 2. Why do monolithic operating systems tend to be efficient? What is a key weakness of monolithic kernels? Anecdote System Architect vs. System Engineer If you enter the field of operating systems development and you become more senior, you may be given a title like systems architect or systems engineer. A professional software colleague was a guest speaker at a conference some years ago. He was introduced as a systems architect. He said: "You may wonder how that differs f r o m being a systems engineer." He humbly explained the difference w i t h a building analogy: "When an engineer builds a building, it's very well built, but it's so ugly that the people tear it down; when an architect builds a building, it's very beautiful, but it falls d o w n ! " Lesson to operating systems designers: You need to combine aspects of both architecture and engineering to insure that your systems are both well built and elegant. The latter goal is less important. 1.13 Operating System Architectures 35 Ans: 1) In a monolithic operating system every component of the operating system is contained in the kernel. 2) Monolithic kernels tend to be efficient because few calls cross from user space to kernel space. Because all OS code in monolithic kernels operates with unrestricted access to the computer's hardware and software, these systems are particularly susceptible to damage from errant code. 1.13.2 Layered Architecture As operating systems became larger and more complex, purely monolithic designs became unwieldy. The layered approach to operating systems attempts to address this issue by grouping components that perform similar functions into layers. Each layer communicates exclusively with those immediately above and below it. Lowerlevel layers provide services to higher-level ones using an interface that hides their implementation. Layered operating systems are more modular than monolithic operating systems, because the implementation of each layer can be modified without requiring any modification to other layers. A modular system has self-contained componen that can be reused throughout the system. Each component hides how it performs Operating Systems Thinking Architecture Just as architects use different increasingly difficult to change calls the services provided by the approaches to designing build- approaches the further into the layer below it, while the imple- ings, operating systems designers development you proceed, so it is mentation of that layer is hidden employ different architectural important to choose the proper from the layer above. Layering approaches to designing operating systems. Sometimes these architecture early in system development. More generally, it is combines the virtues of the software engineering techniques of approaches are pure in that one architectural approach is used much easier to build the building correctly in the first place than it modularity and information hiding to provide a solid basis for building throughout the system. Sometimes hybridized approaches are is to modify the building after it has been built. quality systems. We discuss layered software approaches throughout used, mixing the advantages of several architectural styles. The One of the most common architectural approaches the book, starting with a historic mention of Dijkstra's THE system approach the designer chooses will have monumental consequences on the initial implemen- employed in software systems such as operating systems is called layering. This software is divided into (see Section 1.13.2, Layered Architecture) and continuing to explanations of how layering is used tation and the evolution of the modules called layers that each in Linux and Windows XP in operating system. It becomes perform certain tasks. Each layer Chapters 20, and 21, respectively. 36 Introduction to Operating Systems its job and presents a standard interface that other components can use to request its services. Modularity imposes structure and consistency on the operating system—often simplifying validation, debugging and modification. However, in a layered approach, a user process's request may need to pass through many layers before it is serviced. Because additional methods must be invoked to pass data from one layer to the next, performance degrades compared to that of a monolithic kernel, which may require only a single call to service a similar request. Also, because all layers have unrestricted access to the system, layered kernels are also susceptible to damage from errant or malicious code. The THE operating system is an early example of a layered operating system (Fig. 1.4).107 Many of today's operating systems, including Windows XP and Linux, implement some level of layering. Self Review 1. How are layered operating systems more modular than monolithic operating systems? 2. Why do layered operating systems tend to be less efficient than monolithic operating systems? Ans: 1) In layered operating systems, the implementation and interface are separate for each layer. This allows each layer to be tested and debugged separately. It also enables designers to change each layer's implementation without needing to modify the other layers. 2) In layered operating systems, several calls may be required to communicate between the layers, whereas this overhead does not exist in monolithic kernels. 1.13.3 Microkernel Architecture A microkernel operating system architecture provides only a small number of services in an attempt to keep the kernel small and scalable. These services typically include low-level memory management, interprocess communication and basic pro- Figure 1.4 | Layers of the THE operating system. 1.13 Operating System Architectures 37 cess synchronization to enable processes to cooperate. In microkernel designs, most operating system components—such as process management, networking, file system interaction and device management—execute outside the kernel with a lower privilege level (Fig. 1.5).108, 109, 110, 111 Microkernels exhibit a high degree of modularity, making them extensible, portable and scalable.112 Further, because the microkernel does not rely on each component to execute, one or more components can fail, without causing the operating system to fail. However, such modularity comes at the cost of an increased level of intermodule communication, which can degrade system performance. Although few of today's popular operating systems fully embrace the microkernel design, Linux and Windows XP, for example, contain modular components.113 Self Review 1. What is the difference between a purely layered architecture and a microkernel architecture? 2. How do microkernels promote portability? Ans: 1) A layered architecture enables communication exclusively between operating system components in adjacent layers. A microkernel architecture enables communication between all operating system components via the microkernel. 2) The microkernel does not depend on a particular hardware platform; support for new hardware can be provided by loading a new module. Figure 1.5 | Microkernel operating system architecture. 38 Introduction to Operating Systems 1.13.4 Networked and Distributed Operating Systems Advances in telecommunications technology have profoundly affected operating systems. A network operating system enables its processes to access resources (e.g., files) that reside on other independent computers on a network.114 The structure of many networked and distributed operating systems is often based on the client/server model (Fig. 1.6). The client computers in such a network request resources—such as files and processor time—via the appropriate network protocol. The servers respond with the appropriate resources. In such networks, operating system designers must carefully consider how to manage data and communication among computers. Some operating systems are more "networked" than others. In a networked environment, a process can execute on the computer on which it is created or on another computer on the network. In some network operating systems, users can specify exactly where their processes run; in others, the operating system determines where processes are executed. For example, the system may determine that a process can be more efficiently executed on a computer experiencing a light load.115 Networked file systems are an important component of networked operating systems. At the lowest level, users acquire resources on another machine by explicitly connecting to that machine and retrieving files. Higher-level network file systems enable users to access remote files as if they were on the local system. Examples of network file systems include Sun's Network File System (NFS) and Figure 1.6 | Client/server networked operating system model. Web Resources 39 CMU's Andrew and Coda file systems. Networked file systems are discussed in detail in Chapter 18, Distributed Systems and Web Services. A distributed operating system is a single operating system that manages resources on more than one computer system. Distributed systems provide the illusion that multiple computers are a single powerful computer, so that a process can access all of the system's resources regardless of the process's location within the distributed system's network of computers. 116 Distributed operating systems are often difficult to implement and require complicated algorithms to enable processes to communicate and share data. Examples of distributed operating systems are MIT's Chord operating system and the Amoeba operating system from the Vrije Universiteit (VU) in Amsterdam. 117, 118 We discuss distributed systems in Chapter 17, Introduction to Distributed Systems. Now that we have presented a seemingly endless stream of facts, issues, and acronyms, we proceed with a discussion of the basic principles of computer hardware and software in Chapter 2, Hardware and Software Concepts. Self Review 1. What is the major difference between networked and distributed operating systems? 2. What is the primary advantage of a distributed operating system? What is the primary challenge of designing one? Ans: 1) A networked operating system controls one computer but cooperates with other computers on the network. In a distributed operating system, one operating system controls many computers in a network. 2) The primary advantage is that processes do not need to know the locations of the resources they use, which simplifies applications programming. This comes at the expense of the systems programmer, who must implement complicated algorithms to enable processes to communicate and share data among many computers, creating the illusion of there being only a single larger computer. Web Resources www.bel1-1abs.com/history/unix/ developer.apple.com/darwin/history.html Provides a history of the UNIX operating system, from its oriCovers the evolution of Darwin, the core of Apple's OS X gins in the Multics system to the mature UNIX operating sysoperating system. terns of today. Discusses many of the design and architectural www.cryptonomicon.com/beginning.html considerations in the evolution of UNIX. Contains a link to the article, "In the Beginning Was the Cornwww.softpanorama.org/History/os_history.shtml mand Line," by Neal Stephenson. This text is a narrative Provides a wealth of information on open-source software account of the recent history of operating systems, making colfrom a historical perspective. orful use of anecdote and metaphor. www.microsoft.com/windows/WinHistoryIntro.mspx www.acm.org/sigcse/cc2001/225.html Provides a history of the Microsoft Windows family of operatLists the operating systems course curriculum recommendaing systems. tion from the ACM/IEEE Joint Task Force that completed the www.viewz.com/shoppingguide/os.shtml Computing Curricula 2001 (CC2001) project. The curriculum Compares several popular operating systems, including Winindicates key areas of coverage for a typical operating systems dows, Linux and MacOS, and provides a historical perspective. course. 40 Introduction to Operating Systems whatis.techtarget.com/ www.wikipedia.org/ Provides definitions for computer-related terms. www.webopedia.com/ Provides an online dictionary and a search engine for computer- and Internet-related terms. Wikipedia is a project aimed at creating a free, open and accurate encyclopedia online. Any user may modify the entries in the encyclopedia, and all entries are licensed under the GNU Free Documentation License (GNUFDL). Summary Some years ago an operating system was defined as the software that controls the hardware, but the landscape of computer systems has evolved significantly since then, requiring a more complicated description. To increase hardware utilization, applications are designed to execute concurrently. However, if these applications are not carefully programmed, they might interfere with one another. As a result, a layer of software called an operating system separates applications (the software layer) from the hardware they access. When a user requests that the computer perform an action (e.g., execute an application or print a document), the operating system manages the software and hardware to produce the desired result. Operating systems are primarily resource managers—they manage hardware, including processors, memory, input/output devices and communication devices. The operating system must also manage applications and other software abstractions that, unlike hardware, are not physical objects. Operating systems have evolved over the last 60 years through several distinct phases or generations that correspond roughly to the decades. In the 1940s, the earliest electronic digital computers did not include operating systems. The systems of the 1950s generally executed only one job at a time, but used techniques that smoothed the transition between jobs to obtain maximum utilization of the computer system. A job constituted the set of instructions that a program would execute. These early computers were called single-stream batch-processing systems, because programs and data were submitted in groups or batches by loading them consecutively onto tape or disk. The systems of the 1960s were also batch-processing systems, but they used the computer's resources more efficiently by running several jobs at once. The systems of the 1960s improved resource utilization by allowing one job to use the processor while other jobs used peripheral devices. With these observations in mind, operating systems designers developed multiprogramming systems that managed a number of jobs at once, that number being indicated by the system's degree of multiprogramming. In 1964, IBM announced its System/360 family of computers. The various 360 computer models were designed to be hardware compatible, to use the OS/360 operating system and to offer greater computer power as the user moved upward in the series. More advanced operating systems were developed to service multiple interactive users at once. Timesharing systems were developed to support large numbers of simultaneous interactive users. Real-time systems attempt to supply a response within a certain bounded time period. The resources of a real-time system are often heavily under-utilized. It is more important for real-time systems to respond quickly when needed than to use their resources efficiently. Turnaround time—the time between submission of a job and the return of its results—was reduced to minutes or even seconds. The value of timesharing systems in support of program development was demonstrated when MIT used the CTSS system to develop its own successor, Multics. TSS, Multics and CP/CMS all incorporated virtual memory, which enables programs to address more memory locations than are actually provided in main memory, which is also called real memory or physical memory. The systems of the 1970s were primarily multimode timesharing systems that supported batch processing, timesharing and real-time applications. Personal computing was in its incipient stages, fostered by early and continuing developments in microprocessor technology. Communications between computer systems throughout the United States increased as the Department of Defense's TCP/IP communications standards became widely used—especially in military and university computing environments. Security problems increased as growing volumes of information passed over vulnerable communications lines. The 1980s was the decade of the personal computer and the workstation. Rather than data being brought to a central, large-scale computer installation for processing, computing was distributed to the sites at which it was needed. Personal computers proved to be relatively easy to learn and use, partially because of graphical user interfaces (GUI), which used graphical symbols such as windows, Summary icons and menus to facilitate user interaction with programs. As technology costs declined, transferring information between computers in computer networks became more economical and practical. The client/server distributed computing model became widespread. Clients are user computers that request various services; servers are computers that perform the requested services. The software engineering field continued to evolve, a major thrust by the United States government being aimed especially at providing tighter control of Department of Defense software projects. Some goals of the initiative included realizing code reusability and a greater degree of abstraction in programming languages. Another software engineering development was the implementation of processes containing multiple threads of instructions that could execute independently. In the late 1960s, ARPA, the Advanced Research Projects Agency of the Department of Defense, rolled out the blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions. ARPA proceeded to implement what was dubbed the ARPAnet—the grandparent of today's Internet. ARPAnet's chief benefit proved to be its capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on today's Internet, with e-mail, instant messaging and file transfer facilitating communications among hundreds of millions of people worldwide. The ARPAnet was designed to operate without centralized control. The protocols (i.e., set of rules) for communicating over the ARPAnet became known as the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP was used to manage communication between applications. The protocols ensured that messages were routed properly from sender to receiver and arrived intact. Eventually, the government decided to allow access to the Internet for commercial purposes. The World Wide Web allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animations, audios or videos) on almost any subject. Even though the Internet was developed more than three decades ago, the introduction of the World Wide Web (WWW) was a relatively recent event. In 1989, Tim Berners-Lee of CERN (the European Center for Nuclear Research) began to develop a technology for sharing information via hyperlinked text documents. To implement this new technology, He created the Hypertext Markup Language (HTML). Berners-Lee also implemented the Hypertext Transfer Protocol (HTTP) to form the communications 41 backbone of his new hypertext information system, which he called the World Wide Web. Hardware performance continued to improve exponentially in the 1990s. Inexpensive processing power and storage allowed users to execute large, complex programs on personal computers and enabled small to mid-size companies to use these economical machines for the extensive database and processing jobs that earlier had been delegated to mainframe systems. In the 1990s, the shift toward distributed computing (i.e., using multiple independent computers to perform a common task) rapidly accelerated. As demand for Internet connections grew, operating system support for networking tasks became standard. Users at home and in large corporations increased productivity by accessing the resources on networks of computers. Microsoft Corporation became dominant in the 1990s. Its Windows operating systems, which borrowed from many concepts popularized by early Macintosh operating systems (such as icons, menus and windows), enabled users to navigate multiple concurrent applications with ease. Object technology became popular in many areas of computing. Many applications were written in object-oriented programming languages, such as C++ or Java. In object-oriented operating systems (OOOS), objects represent components of the operating system. Object-oriented concepts such as inheritance and interfaces were exploited to create modular operating systems that were easier to maintain and extend than operating systems built with previous techniques. Most commercial software is sold as object code. The source code is not included, enabling vendors to hide proprietary information and programming techniques. Free and open-source software became increasingly common in the 1990s. Open-source software is distributed with the source code, allowing individuals to examine and modify the software before compiling and executing it. The Linux operating system and the Apache Web server are both free and open source. In the 1980s, Richard Stallman, a developer at MIT, launched the GNU project to recreate and extend most of the tools for AT&T's UNIX operating system. Stallman created the GNU project because he disagreed with the concept of paying for permission to use software. The Open Source Initiative (OSI) was founded to further the benefits of open-source programming. Open-source software facilitates enhancements to software products by permitting anyone in the developer community to test, debug and enhance applications. This increases the chance that subtle 42 Introduction to Operating Systems bugs, which could otherwise be security risks or logic errors, will be caught and fixed. Also, individuals and corporations can modify the source to create custom software that meets the needs of a particular environment. In the 1990s, operating systems became increasingly user friendly. The GUI features that Apple had built into its Macintosh operating system in the 1980s were widely used in many operating systems and became more sophisticated. "Plug-and-play" capabilities were built into operating systems, enabling users to add and remove hardware components dynamically without manually reconfiguring the operating system. Middleware is software that links two separate applications, often over a network and often between incompatible machines. It is particularly important for Web services because it simplifies communication across multiple architectures. Web services encompass a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. They are readyto-use pieces of software on the Internet. When the IBM PC appeared, it immediately spawned a huge software industry in which independent software vendors (ISVs) were able to market software packages for the IBM PC to run under the MS-DOS operating system. If an operating system presents an environment conducive to developing applications quickly and easily, the operating system and the hardware are more likely to be successful in the marketplace. Once an application base (i.e., the combination of the hardware and the operating system environment in which applications are developed) is widely established, it becomes extremely difficult to ask users and software developers to convert to a completely new applications development environment provided by a dramatically different operating system. Operating systems intended for high-end environments must be designed to support large main memories, special-purpose hardware, and large numbers of processes. Embedded systems are characterized by a small set of specialized resources that provide functionality to devices such as cell phones and PDAs. In these environments, efficient resource management is the key to building a successful operating system. Real-time systems require that tasks be performed within a particular (often short) time frame. For example, the autopilot feature of an aircraft must constantly adjust speed, altitude and direction. Such actions cannot wait indefinitely—and sometimes cannot wait at all—for other nonessential tasks to complete. Some operating systems must manage hardware that may or may not physically exist in the machine. A virtual machine (VM) is a software abstraction of a computer that often executes as a user application on top of the native operating system. A virtual machine operating system manages the resources provided by the virtual machine. One application of virtual machines is to allow multiple instances of an operating system to execute concurrently. Another use for virtual machines is emulation—the ability to use software or hardware that mimics the functionality of hardware or software not present in the system. By providing the illusion that applications are running on different hardware or operating systems, virtual machines promote portability—the ability for software to run on multiple platforms —and many other benefits. A user interacts with the operating system via one or more user applications. Often, the user interacts with an operating system through a special application called a shell. The software that contains the core components of the operating system is referred to as the kernel. Typical operating system components include the processor scheduler, memory manager, I/O manager, interprocess communication (IPC) manager, and file system manager. Almost all modern operating systems support a multiprogrammed environment in which multiple applications can execute concurrently. The kernel manages the execution of processes. Program components, which execute independently but use a single memory space to share data, are called threads. When a process wishes to access an I/O device, it must issue a system call to the operating system. That system call is subsequently handled by a device driver—a software component that interacts directly with hardware—often containing device-specific commands and other instructions to perform the requested input and output operations. Users have come to expect certain characteristics of operating systems, such as efficiency, robustness, scalability, extensibility, portability, security and protection, interactivity and usability. In a monolithic operating system, every component is contained in the kernel. As a result, any component can directly communicate with any other. Monolithic operating systems tend to be highly efficient. A disadvantage of monolithic designs is that it is difficult to determine the source of subtle errors. The layered approach to operating systems attempts to address this issue by grouping components that perform similar functions into layers. Each layer communicates exclusively with the layers immediately above and below it. Key In a layered approach, a user process's request may need to pass through many layers before completion. Because additional methods must be invoked to pass data and control from one layer to the next, system throughput decreases compared to that with a monolithic kernel, which may require only a single call to service a similar request. A microkernel

operating system architecture provides only a small number of services in an attempt to keep the kernel small and scalable. Microkernels exhibit a high degree of modularity, making them extensible, portable Terms 43 and scalable. However, such modularity comes at the cost of an increased level of intermodule communication, which can degrade system performance. A network operating system runs on one computer and allows its processes to access resources such as files and processors on a remote computer. A distributed operating system is a single operating system that manages resources on more than one computer system. The goals of a distributed operating system include transparent performance, scalability, fault tolerance and consistency. Key Terms Advanced Research Projects Agency (ARPA)—Government agency under the Department of Defense that laid the groundwork for the Internet; it is now called the Defense Advanced Research Projects Agency (DARPA). application base—Combination of the hardware and the operating system environment in which applications are developed. It is difficult for users and application developers to convert from an established application base to another. application programming interface (API)—Specification that allows applications to request services from the kernel by making system calls. ARPAnet—Predecessor to the Internet that enabled researchers to network their computers. ARPAnet's chief benefit proved to be quick and easy communication via what came to be known as electronic mail (e-mail). bandwidth—Information-carrying capacity of a communications line. business-critical system—System that must function properly, but whose failure of which leads to reduced productivity and profitability; not as crucial as a mission-critical system, where failure could put human lives could be at risk. C—High-level programming language that was designed and used to implement the UNIX operating system. device driver—Software through which the kernel interacts with hardware devices. Device drivers are intimately familiar with the specifics of the devices they manage— such as the arrangement of data on those devices—and they deal with device-specific operations such as reading data, writing data and opening and closing a DVD drive's tray. Drivers are modular, so they can be added and removed as a system's hardware changes, enabling users to add new types of devices easily; in this way they contribute to a system's extensibility. disk scheduler—Operating system component that determines the order in which disk I/O requests are serviced to improve performance. distributed computing—Using multiple independent computers to perform a common task. distributed operating system—Single operating system that provides transparent access to resources spread over multiple computers. distributed system—Collection of computers that cooperate to perform a common task. efficient operating system—Operating system that exhibits high throughput and small turnaround time. client—Process that requests a service from another process (a server). The machine on which the client process runs is also called a client. embedded system—Small computer containing limited resources and specialized hardware to run devices such as PDAs or cellular phones. compute-bound—See processor-bound. extensible operating system—An operating system that can incorporate new features easily. CP/CMS—Timesharing operating system developed by IBM in the 1960s. CTSS—Timesharing operating system developed at MIT in the 1960s. degree of multiprogramming—Number of programs a system can manage at a time. fault tolerance—Operating system's ability to handle software or hardware errors. file system manager—Operating system component that organizes named collections of data on storage devices and provides an interface for accessing data on those devices. 44 Introduction to Operating Systems General Public License (GPL)—Open-source software license which specifies that software distributed under it must contain the complete source code, must clearly indicate any modifications to the original code and must be accompanied by the GPL. End users are free to modify and redistribute any software under the GPL. GNU—Project initiated by Stallman in the 1980s aimed at producing an open-source operating system with the features and utilities of UNIX. graphical user interface (GUI)—User-friendly point of access to an operating system that uses graphical symbols such as windows, icons and menus to facilitate program and file manipulation. HyperText Markup Language (HTML)—Language that specifies the content and arrangement of information on a Web page and provides hyperlinks to access other pages. Hypertext Transfer Protocol (HTTP)—Network protocol used for transferring HTML documents and other data formats between a client and a server. This is the key protocol of the World Wide Web. I/O-bound—Process (or job) that tends to use a processor for a short time before generating an I/O request and relinquishing a processor. I/O manager—Operating system component that receives, interprets and performs I/O requests. independent software vendor (ISV)—Organization that develops and sells software. ISVs prospered after the release of the IBM PC. job—Set of work to be done by a computer. kernel—Software that contains the core components of an operating system. layered operating system—Modular operating system that places similar components in isolated layers. Each layer accesses the services of the layer below and returns results to the layer above. level of multiprogramming—See degree of multiprogramming. massive parallelism—Property of a system containing large numbers of processors so that many parts of computations can be performed in parallel. memory manager— Operating system component that controls physical and virtual memory. microkernel operating system—Scalable operating system that puts a minimal number of services in the kernel and requires user-level programs to implement services generally delegated to the kernel in other types of operating systems. middleware—Layer of software that enables communication between different applications. Middleware simplifies application programming by performing work such as network communication and translation between different data formats. mission-critical system—System that must function properly; its failure could lead to loss of property, money or even human life. interactive operating system—Operating system that allows applications to respond quickly to user input. monolithic operating system—Operating system whose kernel contains every component of the operating system. The kernel typically operates with unrestricted access to the computer system. interactive users—Users that are present when the system processes their jobs. Interactive users communicate with their jobs during execution. Multics—One of the first operating systems to implement virtual memory. Developed by MIT, GE and Bell Laboratories as the successor to MIT's CTSS. Internet—Network of communication channels that provides the backbone for telecommunication and the World Wide Web. Each computer on the Internet determines which services it uses and which it makes available to other computers connected to the Internet. multiprogramming—Ability to store multiple programs in memory at once so that they can be executed concurrently. interprocess communication (IPC) manager—Operating system component that governs communication between processes. Java Virtual Machine (JVM)—Virtual machine that enables Java programs to execute on many different architectures without recompiling Java programs into the native machine language of the computer on which they execute. The JVM promotes application portability and simplifies programming by freeing the programmer from architecture-specific considerations. network operating system—Operating system that can manipulate resources at remote locations but does not hide the location of these resources from applications (as distributed systems can). object-oriented operating system (OOOS)—Operating system in which components and resources are represented as objects. Object-oriented concepts such as inheritance and interfaces help create modular operating systems that are easier to maintain and extend than operating systems built with previous techniques. Many operating systems use objects, but few are written entirely using object-oriented languages. Key online—State describing a computer that is turned on (i.e., active) and directly connected to a network. open-source software—Software that includes the application's source code and is often distributed under the General Public License (GPL) or a similar license. Opensource software is typically developed by teams of independent programmers worldwide. Terms 45 scalable operating system—Operating system that is able to employ resources as they are added to the system. It can readily adapt its degree of multiprogramming to meet the needs of its users. secure operating system—Operating system that prevents users and software from gaining unauthorized access to services and data. server—Process that provides services to other processes open-source initiative (OSI)—Group that supports and promotes open-source software (s e e w w w . o p e (called n s o clients). u r c The e . machine c o m)on. which these processes run is also called a server. operating system—Software that manages system resources to shell—Application (typically GUI or text based) that enables provide services that allow applications to execute propa user to interact with an operating system erly. An operating system may manage both hardware and software resources. Operating systems provide an applicasingle-stream batch-processing system—Early computer system programming interface to facilitate application develtem that executed a series of noninteractive jobs sequenopment. They also help make system resources tially, one at a time. conveniently available to users while providing a reliable, system call— Call from a user process that invokes a service of secure and responsive environment to applications and the kernel. users. thread—Entity that describes an independently executable portability—Property of software that can run on different stream of program instructions (also called a thread of platforms. execution or thread of control). Threads facilitate parallel execution of concurrent activities within a process. portable operating system—Operating system that is designed to operate on many hardware configurations. throughput—Amount of work performed per unit time. Portable Operating Systems Interface (POSEX)-API based timesharing system—Operating system that enables multiple on early UNIX operating systems. simultaneous interactive users. priority of a process—Importance or urgency of a process relative to other processes. process scheduler—Operating system component that determines which process can gain access to a processor and for how long. process—An executing program. processor-bound—Process (or job) that consumes its quantum when executing. These processes (or jobs) tend to be calculation intensive and issue few, if any, I/O requests. protection—Mechanism that implements a system's security policy by preventing applications from accessing resources and services without authorization. real-time system—System that attempts to service requests within a specified (usually short) time period. In missioncritical real-time systems (e.g., air traffic control and petroleum refinery monitors), money, property or even human life could be lost if requests are not serviced on time. robust operating system— Operating system that is fault tolerant and reliable—the system will not fail due to unexpected application or hardware errors (but if it must fail, it does so gracefully). Such an operating system will provide services to each application unless the hardware those services requires fails to function. Transmission Control Protocol/Internet Protocol (TCP/IP)— Family of protocols that provide a framework for networking on the Internet. TSS— Operating system designed by IBM in the 1960s that offered timesharing and virtual memory capabilities. Although it was never released commercially, many of its capabilities appeared in later IBM systems. turnaround time—Time it takes from the submission of a request until the system returns the result. UNIX—Operating system developed at Bell Laboratories that was written using the C high-level programming language. usable operating system—Operating system that has the potential to serve a significant user base by providing an easy-to-use interface and supporting a large set of useroriented applications. virtual machine—Application that emulates the functionality of a computer system. A virtual machine can execute applications that are not directly compatible with the physical system that runs the virtual machine. The user "sees" the computer not as the virtual machine, but as the underlying physical machine. virtual memory—Capability of operating systems that enables programs to address more memory locations than are 46 Introduction to Operating Systems actually provided in main memory. Virtual memory systems help remove much of the burden of memory management from programmers, freeing them to concentrate on application development. using open, text-based standards that enable components written in different languages and on different platforms to communicate. They are ready-to-use pieces of software on the Internet. VM operating system—One of the first virtual machine operating systems; developed at IBM in the 1960s and still used widely today; its latest version is the z/VM. World Wide Web (WWW)-Collection of hyperlinked documents accessible via the Internet using the Hypertext Transfer Protocol (HTTP). Web documents are typically written in languages such as Hypertext Markup Language (HTML) and Extensible Markup Language (XML). Web services—Set of services and related standards that can allow any two computer applications to communicate and exchange data over the Internet. Web services operate Exercises 1.1 Distinguish between multiprogramming and multiprocessing. What were the key motivations for the development of each? 1.2 Briefly discuss the significance of each of the following systems mentioned in this chapter: a. MS-DOS 1.8 What are the advantages and disadvantages of communication between computers? 1.9 Define, compare, and contrast each of the following terms: b. CTSS a. online c. Multics b. real time d. OS/360 c. interactive computing e. TSS d. timesharing f. UNIX g. Macintosh 1.3 1.7 How has distributed computing affected operating system design? What developments made personal computing feasible? 1.4 Why is it impractical to use a virtual machine for a hard real-time system? 1.10 How do middleware and Web services promote interoperability? 1.11 Evaluate monolithic, layered and microkernel architectures according to a. efficiency. 1.5 What role did the development of graphical user interfaces play in the personal computer revolution? b. robustness. 1.6 The GNU Public License (GPL) promotes software that is free, as in "freedom." How does the GPL provide such freedom? d. security. c. extensibility. Suggested Projects 1.12 Prepare a research paper on the Linux operating system. In what ways does it support Stallman's "free as in freedom" doctrine for software? In what ways does Linux conflict with this philosophy? as in both "freedom" and "price." How do the GPL and similar licenses promote open-source software? 1.13 Prepare a research paper on the Internet and how its pervasive use affects operating system design. 1.15 Prepare a research paper on the evolution of operating systems. Be sure to mention the key hardware, software and communications technologies that encouraged each new operating system innovation. 1.14 Prepare a research paper on the open-source software movement. Discuss whether all open-source software is free, 1.16 Prepare a research paper on the future of operating systems. Recommended Reading 47 1.17 Prepare a research paper giving a thorough taxonomy of past and present operating systems. 1.18 Prepare a research paper on Web services. Discuss the key technologies on which the Web services infrastructure is being built. How will the availability of Web services affect applications development? 1.19 Prepare a research paper on business-critical and missioncritical applications. Discuss the key attributes of hardware, communications software and operating systems that are essential to building systems to support these types of applications. 1.20 Prepare a research paper on virtual machine systems. Be sure to investigate IBM's VM operating system and Sun's Java Virtual Machine (JVM). 1.21 Prepare a research paper on operating systems and the law. Survey legislation related to operating systems. 1.22 Prepare a research paper on the impact of operating systems on business and the economy. 1.23 Prepare a research paper on operating systems and security and privacy. Be sure to consider the issues of worms and viruses. 1.24 Prepare a research paper on the ethical issues with which operating systems designers must be concerned. Be sure to deal with issues such as the use of computer systems in warfare and in life-threatening situations, viruses and worms, and other important topics you discover as you do your research for your paper. 1.26 Prepare a research paper discussing the design of massively parallel systems. Be sure to compare large-scale multiprocessor systems (e.g., the Hewlitt-Packard Superdome supercomputer, which contains up to 64 processors; www.hp.com/productsl/servers/scalableservers/ superdome/) to clustered systems and server farms that contain hundreds or thousands of low-end computers that cooperate to perform common tasks (see, for example, www.beowulf.org). Use www.top500.org, a listing of the world's most powerful supercomputers, to determine the type of tasks that each of these massively parallel systems performs. 1.27 What trends are leading the way to dramatic increases in parallel computation? What challenges must be addressed by hardware designers and software designers before parallel computation will become widely used? 1.28 Prepare a research paper that compares MIT's Exokernel (www.pdos.lcs.mit.edu/exo.html) and CMU's Mach microkernel (www-2.cs.emu.edu/afs/cs.emu.edu/project/ mach/public/www/mach.html) research operating systems.119 What is the primary focus of each operating system? Be sure to mention how the researchers organized components such as memory management, disk scheduling and process management. Has either or both of these systems become commercially successful? Has either or both of these systems influenced the designs of commercially successful operating systems? 1.29 Why have UNIX and UNIX-based systems continued to be popular in recent decades? How does Linux impact the future of UNIX systems? 1.25 List several trends leading the way to future operating systems designs. How will each affect the nature of future systems? Recommended Reading This section will be used throughout the book to provide the reader with references to seminal books and papers and current research on the topic of each chapter. Many of the papers that are cited in each chapter can be found in one of several journals associated with either the Association of Computing Machinery (ACM) or the Institute of Electrical and Electronics Engineers (IEEE). Communications of the ACM is the ACM's flagship journal, containing research papers, editorials and features about contemporary topics in computer science. The ACM also sponsors several special-interest groups (SIGs), each dedicated to particular fields of computer science. SIGOPS, the SIG for the field of operating systems (www.acm.org/sigops/), holds annual conferences and publishes the Operating Systems Review.The IEEE Computer Society (www.computer.org), the largest IEEE society, publishes several journals related to computer science, including the popular IEEE Computer (www.computer.org/computer/). The reader is encouraged to join the ACM, SIGOPS and the IEEE Computer Society. In the Web Resources section, we have listed several links to sites that recount the history of operating systems. An enlightening account of the design and development of the OS/360 operating system can be found in Frederick P. Brooks, Jr.'s The Mythical Man-Month.120 Per Brinch Hansen's Classic Operating Systems: From Batch to Distributed Systems provides an anthology of papers describing the design and development of innovative and pioneering operating systems.121 The bibliography for this chapter is located on our Web site at www.deitel.com/books/ os3e/Bibliography.pdf. 48 Introduction to Operating Systems Works Cited 1. "Evolution of the Intel Microprocessor: 1971-2007," . 21. Berlind, D., "Mainframe Linux Advocates Explain It All," ZDNet, April 12, 2002, . 2. "Top 500 List for November 2002," . 22. Frenkel, K., "Allan L. Scherr: Big Blue's Time-Sharing Pioneer," Communications of the ACM, Vol. 30, No. 10, October 1987, pp. 824-829. 3. Weizer, N., "A History of Operating Systems," Datamation, January 1981, pp. 119-126. 4. Goldstein, H., The Computer from Pascal to von Neumann, Princeton: Princeton University Press, 1972. 5. Stern, N., From ENIAC to UNIVAC: An Appraisal of the EckertMauchly Computers Bedford: Digital Press, 1981. 6. Bashe, C, et al., IBM's Early Computers (Cambridge: MIT Press, 1986). 7. Weizer, N., "A History of Operating Systems," Datamation, January 1981, pp. 119-126. 8. Grosch, H, "The Way It Was in 1957," Datamation, September 1977. 9. Denning, P., "Virtual Memory," ACM CSUR, Vol. 2, No. 3, September 1970, pp. 153-189. 10. Codd, E.; E. Lowry; E. McDonough; and C. Scalzi, "Multiprogramming STRETCH: Feasibility Considerations," Communications of the ACM, Vol. 2,1959, pp. 13-17. 11. Critchlow, A., "Generalized Multiprocessor and Multiprogramming Systems," Proc. AFIPS, FJCQ Vol. 24,1963, pp. 107-125. 12. Belady, L., et al., "The IBM History of Memory Management Technology," IBM Journal of Research and Development, Vol. 25, No. 5, September 1981, pp. 491-503. 23. Harrison, T, et al, "Evolution of Small Real-Time IBM Computer Systems," IBM Journal of Research and Development, Vol. 25, No. 5, September 1981, pp. 441-451. 24. Corbato, E, et al., The Compatible Time-Sharing System, A Programmer's Guide, Cambridge: MIT Press, 1964. 25. Crisman, P., et al., eds., The Compatible Time-Sharing System, Cambridge: MIT Press, 1964. 26. Lett, A., and W. Konigsford, "TSS/360: A Time-Shared Operating System," Proceedings of the Fall Joint Computer Conference, AFIPS, Vol. 33, Part 1,1968, pp. 15-28. 27. Bensoussan, A.; C. Clingen; and R. Daley, "The Multics Virtual Memory: Concepts and Designs," Communications of the ACM, Vol. 15, No. 5, May 1972, pp. 308-318. 28. Creasy, R., "The Origins of the VM/370 Time-Sharing System," IBM Journal of Research and Development, Vol. 25, No. 5, pp. 483^190. 29. Conrow, K., "The CMS Cookbook," Computing and Networking Services, Kansas State University, June 29, 1994, < w w w . k s u . e d u / cns/pubs/cms/cms-cook/cms-cook.pdf>. 30. Denning, P., "Virtual Memory," ACM Computing Surveys, Vol. 2, No. 3, September 1970, pp. 153-189. 31. Parmelee, R., et al., "Virtual Storage and Virtual Machine Concepts," IBM Systems Journal, Vol. 11, No. 2,1972. 13. "The Evolution of S/390," < w w w - t i . i n f o r m a t i k . u n i - t u e b i n gen.de/os390/arch/history.pdf>. 32. Ritchie, D, "Dennis M. Ritchie," . 14. Amdahl, G.; G. Blaauw; and F. Brooks, "Architecture of the IBM System/360," IBM Journal of Research and Development, Vol. 8, No. 2, April 1964, pp. 87-101. 33. Bell Labs Lucent Technologies, "Ken Thompson," 2002, . 15. Weizer, N, "A History of Operating Systems," Datamation, January 1981, pp. 119-126. 16. Evans, B., "System/360: A Retrospective View," Annals of the History of Computing, Vol. 8, No. 2, April 1986, pp. 155-179. 17. Mealy, G; B. Witt; and W. Clark, "The Functional Structure of OS/360," IBM Systems Journal, Vol. 5, No. 1,1966, pp. 3-51. 18. Case, R., and A. Padeges, "Architecture of the IBM System/370," Communications of the ACM, Vol. 21, No. 1, January 1978, pp. 73-96. 19. Gifford, D., and A. Spector, "Case Study: IBM's System/360-370 Architecture," Communications of the ACM, Vol. 30, No. 4, April 1987, pp. 291-307. 20. "The Evolution of S/390," < w w w - t i . i n f o r m a t i k . u n i - t u e b i n gen.de/os390/arch/history.pdf>. 34. Mackenzie, R., "A Brief History of UNIX," . 35. Cook, D; J. Urban, and S. Hamilton, "UNIX and Beyond: An Interview with Ken Thompson," Computer, May 1999, pp. 58-64. 36. Cook, D.; J. Urban, and S. Hamilton, "UNIX and Beyond: An Interview with Ken Thompson," Computer, May 1999, pp. 58-64. 37. Bell Labs Lucent Technologies, "Ken Thompson," 2002, . 38. Bell Labs Lucent Technologies, "Dennis Ritchie," . 2002, 39. Bell Labs Lucent Technologies, "Dennis Ritchie," . 2002, 40. Ritchie, D, "Dennis M. Ritchie," . Work Cited 49 41- Reagan, P., and D. Cunningham, "Bell Labs Unveils Open Source Release of Plan 9 Operating System," June 7, 2000, . 62. Quittner, X, "Tim Berners-Lee," March 29, 1999, . 42. Bell Labs, "Vita Nuova Publishes Source Code for Inferno Operating System, Moving Network Computing into the 21st Century," . 63. Berners-Lee, T, et al, "The World-Wide Web," Communications of the ACM, Vol. 37, No. 8, August 1994, pp. 76-82. 43 Kildall, G., "CP/M: A Family of 8- and 16-bit Operating Systems," Byte, Vol. 6, No. 6, June 1981, pp. 216-232. 44. Quarterman, J. S., and J. C. Hoskins, "Notable Computer Networks," Communications of the ACM, Vol. 29, No. 10, October 1986, pp. 932-971. 45. Stefik, M., "Strategic Computing at DARPA: Overview and Assessment," Communications of the ACM, Vol. 28, No. 7, July 1985, pp. 690-707. 46. Comer, D., Internetworking with TCP/IP: Principles, Protocols, and Architecture, Englewood Cliffs, NJ: Prentice Hall, 1988. 47. Martin, J., and K. K. Chapman, Local Area Networks: Architectures and Implementations, Englewood Cliffs, NJ: Prentice Hall, 1989. 64. World Wide Web Consortium, "Longer Bio for Tim BernersLee," . 65. Quittner, J., "Tim Berners-Lee," March 29, 1999, . 66. Berners-Lee, T, et al, "The World-Wide Web," Communications of the ACM, Vol. 37, No. 8, August 1994, pp. 76-82. 67. World Wide Web Consortium, "Longer Bio for Tim BernersLee," . 68. Quittner, J., "Tim Berners-Lee," March 29, 1999, . 69. Moore, G, "Cramming More Components onto Integrated Circuits," Electronics, Vol. 38, No. 8, April 19,1965. 48. Metcalfe, R., and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Communications of the ACM, Vol. 19, No. 7, July 1976. 70. "One Trillion-Operations-Per-Second," Intel Press Release, December 17, 1996, . 49. Balkovich, E.; S. Lerman; and R. Parmelee, "Computing in Higher Education: The Athena Experience," Computer, Vol. 18, No. 11, November 1985, pp. 112-127. 71. Mukherjee, B.; K. Schwan; and P. Gopinath, "A Survey of Multiprocessor Operating Systems," Georgia Institute of Technology, November 5,1993, p. 2. 50. Zmoelnig, Christine, "The Graphical User Interface. Time for a Paradigm Shift?" August 30, 2001, . 72. "Microsoft Timeline," . 51. Engelbart, D., "Who we are. How we think. What we do." June 24,2003,. 52. Martin, E., "The Context of STARS," Computer, Vol. 16, No. 11, November 1983, pp. 14-20. 53. Ecklund, J., "Interview with Douglas Engelbart," May 4, 1994, . 54. Stauth, D., "Prominent Oregon State Alum Receives Leading Prize for Inventors," April 9, 1997, < o r e g o n s t a t e . e d u / d e p t / ncs/newsarch/1997/Apri197/engelbart.htm>. 73. Lea, R.; P. Armaral; and C. Jacquemot, "COOL-2: An Object-oriented Support Platform Built Above the CHORUS Micro-kernel," Proceedings of the International Workshop on Object Orientation in Operating Systems 1991, October, 1991. 74. Weiss, A., "The Politics of Free (Software)," netWorker, September 2001, p. 26. 75. "The GNU CNU>. Manifesto," . on Principles "Lamport on Mutual Exclusion: 27 Years of PlantAugust 2001, . 31. Lamport, L., "TLA: The Temporal Logic of Actions," August 26, 2003, . 32. Lamport, L., et al., "Specifying and Verifying Systems with TLA+," September 2002, < r e s e a r c h . m i c r o s o f t . c o m / u s e r s / lamport/pubs/spec-and-verifying.pdf>. 33. Anderson, J., "Lamport on Mutual Exclusion: 27 Years of Planting Seeds," August 2001, . Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, pp. 3—12. 34. Anderson, J., "Lamport on Mutual Exclusion: 27 Years of Planting Seeds," August 2001, . Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, pp. 3-12. 35. Lamport, L. "A Fast Mutual Exclusion Algorithm," November 14, 1985, . 36. Lamport, L., "The Writings of Leslie Lamport," September 8, 2003, . 37. Anderson, X, "Lamport on Mutual Exclusion: 27 Years of Planting Seeds," August 2001, . Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, pp. 3-12. 38. Lamport, L„ "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, Vol. 21, No. 7, July 1978, pp.558-565, < r e s e a r c h . m i c r o s o f t . c o m / u s e r s / lamport/pubs/time-clocks.pdf>. 39. The LaTeX Project, "LaTeX Project home page," January 27. 2003, . 40. Gilbert, Philip, and W. J. Chandler, "Interference Between Communicating Parallel Processes," Communications of the ACM. Vol. 15, No. 6, June 1972, p. 436. 41. Presser, Leon, "Multiprogramming Coordination," ACM Computing Surveys (CSUR), Vol. 7, No. 1, January 1975, p. 38. 42. Kruskal, Clyde P.; Larry Rudolph; and Marc Snir, "Efficient Synchronization of Multiprocessors with Shared Memory," ACM Transactions on Programming Languages and Systems (TOPLAS), October 1988, p. 580. 43. Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, Netherlands, 1965, reprinted in F. Genuys, ed., Programming Languages, New York: Academic Press, 1968, pp. 43-112. 44. Computer Science Department of the University of Texas at Austin, "E. W. Dijkstra Archive: Obituary," August 2002. . 45. Dijkstra, E., "GOTO Considered Harmful," March 1968. . Reprinted from Communications of the ACM, Vol. 11. No. 3, March 1968, pp. 147-148. 46. Dijkstra, E„ "GOTO Considered Harmful," March 1968. . Reprinted from Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148. 47. Dijkstra, E., "Appendix of The Structure of the "THE" Multiprogramming System," May 1968, . Reprinted from Communications of the ACM, Vol. 11, No. 5. May 1968, pp. 345-346. 48. Dijkstra, E., "Structure of an Extendable Operating System." November 1969, . 49. Computer Science Department of the University of Texas at Austin, "E. W. Dijkstra Archive: Obituary," August 2002. . 50. T Whaley, "CS 313 — Edsger Wybe Dijkstra," . 51. Department of Computer and Information Science of the University of Michigan at Dearborn, "The ALGOL Programming Language," November 24, 1996, . 52. Computer Science Department of the University of Texas at Austin, "E. W. Dijkstra Archive: Obituary," August 2002. . 53. Dijkstra, E., "How Do We Tell Truths That Might Hurt?" 1975. . 54. Dijkstra, E., "The End of Computing Science?" 2000. . 55. Hansen, P. Brinch, "The Nucleus of a Multiprogramming System," Communications of the ACM, Vol. 13, No. 4, April 1970, pp. 238-241,250. 56. Stark, Eugene, "Semaphore Primitives and Starvation-Free Mutual Exclusion," Journal of the Association for Computing Machinery, Vol. 29, No. 4, October 1982, pp. 1049-1072. Works 57 . Denning, P. J.; Dennis, T.D.; and J. A. Brumfield, "Low Contention Semaphores and Ready Lists," Communications of'the ACM, Vol. 24, No. 10, October 1981, pp. 687-699. 58. v Brinch Hansen, P., "Edison: A Multiprocessor Language," Software Practice and Experience, Vol. 11, No. 4, April 1981, pp. 325361. 59. Linux kernel source code, version 2.5.75, < m i n e r . c s . w m . e d u / lxr3.linux/http/source/ipc/sem.c?v=2.5.75>. 60. Hansen, P. Brinch, "Distributed Processes—a Concurrent Programming Concept," Communications of the ACM, Vol. 21, No. 11, November 1978, pp. 934-941. Cited 243 71. Raynal, M., Algorithms for Mutual Exclusion, Cambridge, MA: MIT Press, 1986. 72. Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, Netherlands, 1965, reprinted in E Genuys, ed., Programming Languages, New York: Academic Press, 1968, pp. 43-112. 73. Peterson, G. L., "Myths About the Mutual Exclusion Problem," Information Processing Letters,Vol. 12, No. 3, June 1981, pp. 115116. 74. Lamport, L., "A New Solution of Dijkstra's Concurrent Programming Problem," Communications of the ACM, Vol. 17, No. 8, August 1974, pp. 453-455. 61. Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, Vol. 21, No. 7, July 1978, pp. 558-565. 75. Knuth, D., "Additional Comments on a Problem in Concurrent Programming Control," Communications of the A CM, Vol. 9, No. 5, May 1966, pp. 321-322. 62. Lamport, L., "The Implementation of Reliable Distributed Multiprocess Systems," Computer Networks, Vol. 2, No. 2, April 1978, pp. 95-114. 76. Eisenberg, M. A., and M. R. McGuire, "Further Comments on Dijkstra's Concurrent Programming Control Problem," Communications oftheACM,\o\. 15, No. 11, November 1972, p. 999. 63. Ricart, G., and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Communications of the ACM, Vol. 24, No. 1, January 1981, pp. 9-17. 77. Lamport, L., "A New Solution to Dijkstra's Concurrent Programming Problem," Communications of the ACM, Vol. 17, No 8, August 1974, pp. 453-455. 64. Eisenberg, M. A., and M. R. McGuire, "Further Comments on Dijkstra's Concurrent Programming Control Problem," Communications of the ACM,Vol. 15,No. 11, November 1972, p. 999. 78. Anderson, James H., and Yong-Jik Kim, "An Improved Lower Bound for the Time Complexity of Mutual Exclusion," Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, pp. 90-99, August 2001, Newport, Rhode Island, United States. 79. Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, Netherlands, 1965, reprinted in F Genuys, ed., Programming Languages, New York: Academic Press, 1968, pp. 43-112. 65. Andrews, G. R., and E B. Schneider, "Concepts and Notations for Concurrent Programming," ACM Computing Surveys, Vol. 15, No. 1, March 1983, pp. 3-44. 66. Bernstein, A. J., "Program Analysis for Parallel Processing," IEEE Transactions on Computers, Vol. 15, No. 5, October 1966, pp. 757-762. 67. Courtois, P. X; E Heymans; and D. L. Parnas, "Concurrent Control with Readers and Writers," Communications of the ACM, Vol. 14, No. 10, October 1971, pp. 667-668. 68. Lamport, L., "Concurrent Reading and Writing," Communications of the ACM,Vol. 20, No. 11, November 1977, pp. 806-811. 69. Ricart, G., and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Communications of the ACM, Vol. 24, No. 1, January 1981, pp. 9-17. 70. Jones, D. W, "An Empirical Comparison of Priority Queue and Event Set Implementations," Communications of the ACM, Vol. 29, No. 4, April 1986, pp. 300-311 80. Denning, P. X; Dennis, T.D.; and J. A. Brumfield, "Low Contention Semaphores and Ready Lists," Communications of the ACM, Vol. 24, No. 10, October 1981, pp. 687-699. 81. Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, Vol. 21, No. 7, July 1978, pp. 558-565. 82. Lamport, L., "The Implementation of Reliable Distributed Multiprocess Systems," Computer Networks,Vol. 2, No. 2, April 1978, pp. 95-114. 83. Ricart, G, and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Communications of the ACM, Vol. 24,No. 1,January 1981,pp. 9-17. High thoughts must have high language. —Aristophanes— As writers become more numerous, it is natural for readers to become more indolent. —Oliver Goldsmith— When the last reader reads no more. —Oliver Wendell Holmes— The first precept was never to accept a thing as true until I knew it as such witkout a sin- gle doubt. —Rene Descartes- This shows how much easier it is to be critical than to be correct. —Benjamin Disraeli— Chapter 6 Concurrent Programming After reading this chapter, you should understand: how monitors synchronize access to data. how condition variables are used with monitors. solutions for classic problems in concurrent programming such as readers and writers and circular buffer. Java monitors. remote procedure calls. Chapter Outline 6.1 Introduction 6.3 Java Monitors Anecdote: Exhaustive Testing Is Impossible 6.2 Monitors Operating Systems Thinking: Information Hiding 6.2.1 Condition Variables 6.4 Java Multithreading Case Study: Part III: Producer/Consumer Relationship in Java Biographical Note: Per Brinch Hansen 6.2.2 Simple Resource Allocation with Monitors 6.2.3 Monitor Example: Circular Buffer 6.2.4 Monitor Example: Readers and Writers 6.5 Java Multithreading Case Study, Part IV: Circular Buffer in Java Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited. 246 6.1 Introduction 247 6.1 Introduction In the last chapter we presented Dekker's Algorithm and Peterson's Algorithm for the implementation of mutual exclusion primitives, and we studied Dijkstra's semaphores. These mechanisms have a number of weaknesses. They are so primitive that it is difficult to use them to express solutions to more complex concurrency problems, and their presence in concurrent programs increases the already difficult problem of proving program correctness. Their malicious or accidental misuse could lead to subtle bugs that might corrupt the operation of a concurrent system. The semaphore approach, in particular, has many weaknesses.1 If a P operation is omitted, then mutual exclusion is not enforced. If a V operation is omitted, then threads waiting because of P operations could become deadlocked. Once the P operation begins, the thread cannol back out and take an alternate course of action while the semaphore remains in use. A thread may wait on only one semaphore at a time: this could lead to deadlock in resource-allocation situations. For example, two threads may each hold a resource that the other thread is waiting to use. This is the classic case of two-thread deadlock, which we will discuss in Chapter 7, Deadlock and Indefinite Postponement. To combat this problem, researchers have developed higher-level mutual exclusion constructs that simplify solving complex concurrency problems, facilitate proving program correctness and are difficult for a programmer to misuse or corrupt. Concurrent programming is much more difficult than sequential programming. Concurrent programs are harder to write, debug, modify, and prove correct (see the Anecdote. Exhaustive Testing Is Impossible). So whv is the programming community so intent on concurrent programming? There has been a surge of interest in concurrent programming languages because they enable us to express more natural solutions to problems that are inherently parallel. Furthermore, the true hardware parallelism possible with multiprocessors (see Chapter I?. Multiprocessor Management) and distributed systems (sec Chapter 17. Introduction to Distributed Systems) can only be harnessed through concurrent programming. The potential applications for concurrent programming are numerous. There has been much discussion on concurrency in computer networks.2 distributed systems 3 , 5, 6, 7 and real-time systems.8, 9, 10 Certainly, operating systems themselves are important examples of concurrent systems. So are air traffic control systems, mission-critical systems, and real-time process control systems (such as those that control gasoline refineries, chemical manufacturing plants and food processing plants). It is widely believed that human vision is an inherently parallel task. Weather forecasting will almost certainly lake great strides forward when massive parallelism reaches the scale of billions or even trillions of concurrent processors. In this chapter, we consider high-level constructs and languages for concurrent programming. In particular, we investigate monitors, condition variables, interprocess communication using remote procedure calls and the Java programming language's concurrent programming facilities. The pseudocode examples in 248 Concurrent Programming Section 6.2. Monitors, use a C-like syntax: the rest of the chapter uses complete Java programs. The chapter concludes with a lengthy literature section indicating the richness of concurrent programming as a research area. What approach should implementors take when building concurrent systems todav? Prior to the 1990s, high-level concurrent programming languages included to Ada. 11 Concurrent Pascal.12, 13 Distributed Processes,14 Concurrent C.15 Communicating Sequential Processes.16, 17 Modula-2. 18, 19, 20 VAL21 and *MOD (for distributed programming). 22 With the exception of Ada. these languages were generally Anecdote Exhaustive Testing Is Impossible Testing is a hard problem. As you may study in courses on automata theory, a computer may be viewed as a finite state machine. How many states can a computer have? One possible view is that if the machine stores n bits, then the number of states is 2 raised to the nth power. A machine that has even a small main memory— perhaps tens of megabytes— could have far more states than there are atoms in the universe! Consider this experiment: A complex piece of software takes a 1 or 0 as input and yields a 1 or 0 as output. Each time the tester inputs a 1 for a large number of test cases, the software outputs a 1; each time the tester inputs a 0, the software outputs a 0. The tester concludes that the software is simply passing its input through to its output and seems to be working fine. But suppose the software actually contains a counter of the number of times it has been called, and when the counter reaches a certain large value, it performs some other (possibly devastating) action. If in testing that software, it is called fewer than that number of times, then that test case will not occur and the software "problem" will not be detected. Dijkstra emphasized that "testing reveals the presence of bugs, but not their absence." Since it is simply not possible to exhaustively test every system we build, this means that even after a system has been "thoroughly" tested, it could still contain "lurking bugs" that may not surface until long after the system has been deployed. This places a huge burden on the shoulders of operating systems developers. When can we truly say that a system is ready for deployment? Obviously, this decision depends on the nature of the system. Is it a mission-critical or business-critical system? Are human lives at stake? Is it a specialpurpose system whose use will be carefully confined to a narrow, controlled audience, or is it a general-purpose system that could be employed in a great diversity of applications, many of which cannot be determined when the sysrem is under development. Lesson to operating systems designers: Since the systems you build cannot possibly be tested exhaustively, you must "design for testability," minimizing the chance of bugs and providing testing capabilities to help system testers certify that your system is ready for deployment. 6.2 Monitors 249 developed by academics for research purposes and tend to lack many of the features needed to implement real systems. Today, many popular programming languages support concurrency, including Java. C#. Visual C++ .NET. Visual Basic .NET and Python. Concurrent programming presents opportunities for computer science students as they prepare for careers in industry, where few people are experienced in this complex subject. Self Review 1. Why have researchers sought higher-level mutual exclusion constructs? 2. What notable high-level concurrent programming language developed prior to the 1990s did indeed include the features needed to implement real systems? Ans: 1) These constructs facilitate proving program correctness and are difficult to misuse or corrupt. 2) Ada. 6.2 Monitors A monitor is an object that contains both the data and procedures needed to perform allocation of a particular serially reusable shared resource or group of serially reusable shared resources. The notion of a monitor was suggested by Dijkstra.23 then by Brinch Hansen,24, 25 then refined by Hoare. 26 There has been much discussion in the literature on this important topic.27, 28, 29, 30, 31, 32, 33, 34, 35 Monitors have become an important software construct —in fact, the Java programming language makes extensive use of monitors to implement mutual exclusion. To accomplish resource allocation using monitors, a thread must call a monitor entry routine. Many threads may want to enter the monitor at the same time, but mutual exclusion is rigidly enforced at the monitor boundary —only one thread at a time is allowed to enter. A thread that tries to enter the monitor when it is in use is made to wait by the monitor. Because the monitor guarantees mutual exclusion, concurrency problems (such as indeterminate outcomes) are avoided. Data inside a monitor may be either global to all routines within the monitor or local to a specific routine. Monitor data is accessible only within the monitor: there is no way for threads outside the monitor to access monitor data. This is a form of information hiding a software architectural technique that improves modularity and facilitates the development of more reliable software systems (see the Operating Svstems Thinking feature. Information Hiding). If a thread calls a monitor entry routine while no other threads are executing inside the monitor, the thread acquires a lock on the monitor and enters it. While the thread is in the monitor, other threads may not enter the monitor to acquire the resource. If a thread calls a monitor entry routine while the monitor is locked, the monitor makes the calling thread wait outside the monitor until the lock on the monitor is released (i.e.. when a thread is no longer executing inside the monitor). Once inside the monitor, a thread might need to wait because the resource has been allocated to another thread. Because mutual exclusion is enforced at the boundary 250 Concurrent Programming of the monitory, a thread waiting for a resource must do so outside the monitor to allow another thread into the monitor to return the resource. Eventually, the thread that has the resource will call a monitor entry routine to release the resource. This routine could merely free the resource and wait for another requesting thread to arrive. But there may be threads waiting for the resource, so the monitor entry routine calls signal to allow one of the waiting threads to enter the monitor and acquire the resource. If a thread signals the return (also called the release) of the resource, and no threads are waiting, then the signal has no effect (but. of course, the monitor has recaptured the resource, which it can now allocate to arriving threads). To avoid indefinite postponement, a monitor gives a higher priority to waiting threads than to newly arriving ones. Self Review 1. Why must a thread wait for a resource outside a monitor? 2. How does a monitor prevent multiple threads from concurrently executing inside the monitor? Ans: 1) If the thread were to wait inside a monitor for a resource, no other threads could enter the monitor to return the resource. As we will see in Chapter 7. Deadlock and Indefinite Postponement, this could cause all the threads waiting for the resource to deadlock. 2) The monitor enforces mutual exclusion at its border using the kinds of techniques discussed in Chapter 5. Asynchronous Concurrent Execution. Operating Systems Thinking Information Hiding Information hiding is one of the most fundamental techniques of software engineering. It is implemented in many ways in operating systems. When one function or method calls another, the caller does not need to know the details of how the callee is implemented; the caller merely has to know the interface to the callee—what arguments must be passed and in what order, and what return val- ues are expected. Information hiding has many advantages. It makes the caller's job easier. It does not need to be familiar with the (possibly enormous) complexity of how the callee is implemented. It also makes it easier to modify systems—a called function can usually be replaced easily w i t h o u t requiring changes in the caller, as long as the interface to the callee remains the same. Today's operating systems arguably have tens of thousands (or more) components that are constantly being evolved to adapt to new hardware and software trends and tuned to perform better. Information hiding places a crucial role in making these enormous and highly modular systems understandable and maintainable. 6.2 Monitors 251 6.2.1 Condition Variables Monitors implement both mutual exclusion and synchronization between threads of execution. A thread currently inside a monitor may need to wait outside the monitor until another thread performs an action inside the monitor. For example, in the producer/consumer relationship, the producer finding that the consumer still has not read the value in a single shared buffer must wait outside the monitor governing the shared buffer so that the consumer can consume the buffer contents. Similarly, a consumer finding that the shared buffer is empty must wait outside the monitor until the producer fills the buffer. A thread inside a monitor uses a condition variable to wait on a condition outside the monitor. A monitor assicuates a separate condition variable with each distinct situation that might cause a thread to have to wait. We define the wait and signal operations as: wait (condition Variable) signal (conditionVariable) Condition variables are different from '"conventional" variables. Every condition variable has an associated queue. A thread calling wait on a particular condition variable is placed into the queue associated with that condition variable (while in that queue, the thread is outside the monitor, so that another thread can eventually enter the monitor and call s i g n a l) . A thread calling signal on a particular condition variable causes a thread waiting on that condition variable to be removed from the queue associated with it and to reenter the monitor. We may assume a first-in-first-out (FIFO) queue discipline, although priority schemes can be useful in certain sit nations. 36 ' 37 Before that thread can reenter the monitor, the thread calling signal must first exit the monitor. Brinch Hansen (see the Biographical Note. Per Brinch Hansen), noting that many signal statements immediately preceded a r e t u r n statement (i.e.. the thread exits the monitor), proposed a signal-and-exit monitor, in which a thread immediately exits the monitor upon signaling.38 The monitors in the following examples are signal-and-cxit monitors. Alternatively, a signal-and-continue monitor allows a thread inside the monitor to signal that the monitor will soon become available, but still maintain a lock on the monitor until the thread exits the monitor. A thread can exit the monitor by waiting on a condition variable or bv completing execution of the code protected bv the monitor. The thread released bv a signal-and-continue monitor must wait until the signalling thread exits the monitor. As we discuss in Section 6.3. Java Monitors, the Java programming language implements signal-and-continue monitors. Self Review 1. What problem might occur if a priority queue (instead of a FIFO queue) were used for a condition variable? 2. (T/F) Each monitor contains exactly one condition variable. 252 Concurrent Programming Ans: 1) A lower-priority thread could be indefinitely postponed by a stream of higher-priority threads calling wait in the monitor enter the priority queue. 2) False. A monitor contains a separate condition variable for each distinct situation that might cause a thread to call wait in the monitor. 6.2.2 Simple Resource Allocation with Monitors Suppose several threads are contending for a resource that requires exclusive access. A simple monitor for handling the allocation and deallocation of such a resource is shown in Fig. 6.1. Line 4 declares state variable inUse which keeps track of whether the resource is in use or not. Line 5 declares a condition variable that is waited on by a thread finding the resource unavailable and signaled by a thread returning the resource (and thus making it available). Lines 7-17 and 19-25 declare two monitor entry routines. To indicate that these are monitor entry routines (rather than routines private to the monitor), we prefix each routine with the pseudocode keyword monitorEntry. Biographical Note Per Brinch Hansen Per Brinch Hansen has devoted his research to concurrent programming, which is the technique of designing software that has parallel paths of execution. He received his Masters in Electrical Engineering from the Technical University of Denmark in 1962 and has been a professor at the California Institute of Technology, the University of Southern California and the University of Copenhagen, Denmark. He is currently a professor of Electrical Engineering and Computer Science at Syracuse University.39 While working at the Regnecentralen, Copenhagen, Brinch Hansen developed the RC 4000 allowed only sequential programMultiprogramming System for the ming), for users to define access RC (RegneCentralen) 4000 comcontrols on shared data so that puter. This operating system was programs written in Concurrent the first to separate system funcPascal could be checked at comtions into a kernel module. 40 pile time for sharing violations. 42 Brinch Hansen introduced As a demonstration of Concurrent the concept of the monitor conPascal, Brinch Hansen used it to struct—an object that encapsuwrite Solo, a simple operating lates both shared data and the system that was secure (meaning procedures that operate on that that errors would be detected by shared data—in his book, Operatthe system instead of crashing the ing Systems Principles, published program—not the more common by Prentice Hall in 1973 (see meaning of safe from attacks) Section 6.2, Monitors). 41 He incorw i t h o u t using hardware porated monitors into his Concur- support. 43, 44, 45 He later designed rent Pascal language (an the concurrent programming lanextension of Pascal, which guage Joyce.46 6.2 Monitors 253 1 // Fig. 6.1: Resource allocator monitor 2 3 // monitor initialization (performed only once) 4 boolean inUse = false; // simple state variable 5 Condition a v a i l a b l e ; // condition variable 6 7 // request resource 8 monitorEntry void getResource() 9 { 10 if (i n U s e) // is resource in use? 11 { 12 wait(available);//waituntilavailableissignalled 13 } // end if 14 15 inUse = true; // indicates resource is now in use 16 17 } // end getResource 18 19 // return resource 20 monitorEntry void returnResource() 21 { 2 2 inUse = false; // indicate resource is not in use 23 signal(available); // signal a waiting thread to process 24 25 } // end returnResource F i g u r e 6.1 | Simple resource allocation with a monitor in pseudocode. Line 8 begins method getResource. which a thread calls to request the resource associated with the condition variable a v a i l a b l e . Line 10 tests the state variable inUse to see whether the resource is in use. If this value is t r u e , meaning that the resource has been allocated to another thread, the calling thread must wai t on condition variable a v a i l a b l e (line 12). After calling wait, the thread exits the monitor and is placed in the queue associated with the condition variable aval 1 able. As we will see. this allows the thread that is using the resource to enter the monitor and release the resource by signaling condition variable a v a i l a b l e . When the resource is not in use. the requesting thread executes line 15. which sets inUse to t r u e , providing the thread exclusive access to the resource. Line 20 begins method returnResource. which a thread calls to release the resource. Line 22 sets the value of inUse to f a l s e to indicate that the resource is no longer in use and can be allocated to another thread. Line 23 calls signal on condition variable a v a i l a b l e to alert any waiting thread that the resource is now free. If there are any threads waiting in the queue associated with a v a i l a b l e (as a result of executing line 12). the next waiting thread reenters the monitor and executes line 15. obtaining exclusive access to the resource. If there are no threads waiting on a v a i l a b l e , signal has no effect. 254 Concurrent Programming The beauty of the monitor in Fig. 6.1 is that it performs exactly as a binary semaphore: method getResource acts as the P operation: method returnResource acts as the V operation. Because the simple one-resource monitor can be used to implement semaphores, monitors are at least as powerful as semaphores. Note that the monitor initialization (lines 3-5) is performed before threads begin using the monitor; in this case inUse is set to false to indicate that the resource is initially available. Self Review 1. What would happen if returnResource() did not signal the condition variable avalable? 2. What purpose does the keyword monitorEntry serve? Ans: 1) All of the threads waiting on condition variable a v a l a b l e would deadlock — they will wait forever. 2) Keyword monitorEntry distinguishes between monitor entry routines, which are accessible by all threads and can be executed by only one thread at a time, and private monitor methods, which are accessible only within the monitor. 6.2.3 Monitor Example: Circular Buffer In this section we discuss the circular buffer (sometimes called the bounded buffer.) and how it is useful in situations in which a producer thread passes data to a consumer thread. Because the producer and consumer access the same data and may operate at different rates, synchronization between the two threads is essential. We can use an array to implement the circular buffer. In the circular buffer implementation of the solution to the producer/consumer problem, the producer deposits data in the successive elements of the array. The consumer removes them in the order in which they were deposited (FIFO). The producer can be several items ahead of the consumer. Eventually, the producer fills the last element of the array. When it produces more data, it must "wrap around" and again begin depositing data in the first element of the array (assuming, of course, that the consumer has removed the data previously put there by the producer). The array effectively closes in a circle, hence the term circular buffer. Because of the fixed size of the circular buffer, the producer will occasionally find all the array elements full: in this case the producer must wait until the consumer empties an array element. Similarly, there will be times when the consume]' wants to consume, but the array will be empty: in this case the consumer must wait until the producer deposits data into an array element. The monitor in Fig. 6.2 (based on the example presented by Hoare) 47 implements a circular buffer and the appropriate synchronization mechanisms to handle the producer/consumer relationship. 1 // Fig. 6.2: Circular buffer monitor 2 Figure 6.2 | Monitor pseudocode implementation of a circular buffer. (Part 1 of 2.) 6.2 3 4 5 6 7 8 9 10 11 char circularBuffer[] = int writerPosition = 0; int readerPosition = 0; int occupiedSTots = 0; Condition hasData; Condition hasSpace; new char[BUFFER_SIZE],; // buffer // next slot to write to // next slot to read from // number of slots with data // condition variable // condition variable // monitor entry called by producer to write data monitorEntry void putChar(char slotData) 12 { 13 // wait on condition variable hasSpace if buffer is f u l l 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 if (occupiedSTots == BUFFER_SIZE) { wait(hasSpace); // wait until hasSpace is signaled } // end if // write character to buffer circularBuffer[writerPosition] = slotData; ++occupiedSlots; // one more slot has data writerPosition = (writerPosition + 1) % BUFFER_SIZE; signal(hasData); // signal that data is available } // end putChar // monitor entry called by consumer to read data monitorEntry void getChar(outputParameter slotData) { // wait on condition variable hasData if the buffer is empty if (occupiedSlots == 0) { wait(hasData); // wait until hasData is signaled } // end if // read character from buffer into output parameter slotData slotData = circularBuffer[readPosition]; occupiedSlots--; // one fewer slots has data readerPosition = (readerPosition + 1) % BUFFER_SIZE; signal(hasSpace); // signal that character has been read } // end getChar Figure 6.2 | Monitor pseudocode implementation of a circular buffer. (Part 2 of 2.) We shall assume that the array circularBuffer contains BUFFER_SIZE entries consisting of one character (line 3). Variables writerPosition and readerPosition (lines 4-5) indicate in which slot of the circular buffer the next item is to be placed by a producer and from which slot of the circular buffer the next item is to be removed by a consumer, respectively. Monitors 255 256 Concurrent Programming A producer adds data to the buffer by calling method putChar (lines 10-24). Line 14 tests if the buffer is full. If it is, the producer waits on the condition variable hasSpace (line 16). When it does this, the producer leaves the monitor and waits in the queue associated with the condition variable hasSpace. As we will soon see, this allows a consumer to enter the monitor, consume data in the circular buffer and signal condition variable hasSpace.This enables the producer to continue; it writes the data to the buffer (line 20), increments the number of occupied slots (line 21) and updates the w r i t e r P o s i t i o n variable (line 22). Finally, line 23 signals condition variable hasData to enable a waiting consumer, if there is one, to proceed. A consumer reads data from the buffer by calling method getChar (lines 2640). Note the use of the output parameter s l o t D a t a in the pseudocode (indicated by keyword outputParameter in line 27). Many languages support such a capability. Data written to the output parameter is immediately available in the caller's argument. Line 30 tests if the buffer is empty. If it is, the consumer waits on condition variable hasData (line 32). When the consumer can continue, it reads the data from the buffer directly into the output parameter s l o t D a t a (line 36), immediately making the data available in the caller's argument. The consumer then decrements the number of occupied slots (line 37) and updates readerPosi ti on (line 38). Line 39 signals condition variable hasSpace to enable a waiting producer, if there is one, to proceed. The beauty of the circular buffer is that it allows the producer to "get ahead" of the consumer. The producer can create a new value without waiting for the consumer to read the previous value (as is necessary in the single buffer producer/consumer relationship). These extra values are placed in the empty slots of the circular buffer. The consumer will still read the values in the correct order. The circular buffer reduces the amount of time the producer must wait before producing another value, thus improving system performance. The larger the circular buffer, the more values the producer can produce before it must wait for the consumer to empty the buffer. If the producer and consumer work at approximately the same speed, using the circular buffer can increase the average speed of the application. If there is a difference in the average speeds of the threads, this advantage is nullified. For example, if the producer works consistently faster than the consumer, the circular buffer will quickly become full and stay full, forcing the producer to wait each time for the consumer to free up space. Similarly, if the consumer works consistently faster than the producer, then the consumer will typically find the circular buffer empty and will almost always have to wait for the producer to create a value. In these last two cases, using the circular buffer will simply waste memory rather than increasing the speed of the application. Operating systems can use a circular buffer to implement spooling control One common example of spooling occurs when a thread generates lines to be printed on a relatively slow output device such as a printer. Because the thread can produce the lines much faster than the printer can print them, and because we would like the thread to be able to complete its execution as quickly as possible, the thread's output lines can be directed to a circular buffer. The circular buffer may be 6.2 in primary storage, or more likely on disk. The thread creating the lines to be printed is often called a spooler. Another thread reads the lines from the c i r c u l a r B u f f e r and writes them to the printer. But this second thread, often called a despooler, runs at the slower, speed of the printer. The c i r c u l a r B u f f e r has sufficient storage to "take up the slack" resulting from the mismatch in the speeds of the spooler and despooler threads. Of course, we assume the system does not indefinitely generate print lines faster than the printer can print them; if it did, the buffer would always be full and would be of little value in "smoothing" the printing operation. Self Review 1. Name the two situations in which a circular buffer is less efficient than a single-value buffer in the producer/consumer relationship. 2. Would this monitor support one producer and two consumers such that each element that the producer produces is consumed by exactly one consumer? Why or why not? Ans: 1) If the difference in the average speeds of the producer and consumer is significant, the advantage of the circular buffer is nullified. If the producer consistently ran faster than the consumer, it would almost always find the buffer full and would have to wait each time for a consumer to consume. If the consumer consistently ran faster than the producer, it would almost always find the buffer empty and would have to wait each time for a producer to produce. In these cases, a single-value buffer can be used without loss of performance. 2) Yes. Mutual exclusion would still be enforced by the monitor. The producer would still produce values as normal. Only one consumer would read each piece of data from the producer. 6.2.4 Monitor Example: Readers and Writers In computer systems, it is common to have some consumer threads (called readers) that read data and producer threads (called writers) that write it. For example, in an airline reservation system there may be many more readers than writers—many inquiries will be made against the database of available flight information before the customer actually selects and commits to a particular seat on a particular flight. Because readers do not change the contents of the database, many readers may access the database at once. But a writer can modify the data, so it must have exclusive access. When a writer is active, no other readers or writers may be active. This exclusion needs to be enforced only at the record level. It is not necessary to grant a writer exclusive access to the entire database —doing so could dramatically hurt performance. The problem of designing a concurrent program to control access of readers and writers to a database was first posed and solved by Courtois, Heymans, and Parnas.48 The solution in Fig. 6.3 is based on that developed by Hoare. 49 The monitor in Figure 6.3 may be used to control access to an entire database, a subset of the database consisting of many or few records, or even a single record. In any of these cases, the following discussion applies. Only one writer may be active at a time; when a writer is active, the boolean variable wri teLock (line 4) is true. No readers may be active when a writer is active.The integer variable, readers (line 3), indicates the number of active readers. When the number of readers is Monitors 257 258 Concurrent Programming 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 // Fig. 6.3: Readers/writers problem int readers = 0; // number of readers boolean writeLock = false; // true if a writer is writing Condition canWrite; // condition variable Condition canRead; // condition variable // monitor entry called before performing read monitorEntry void beginRead() { // wait outside monitor if writer is currently writing or if // writers are currently waiting to write if (writeLock || queue(canWrite)) { wait(canRead); // wait until reading is allowed } // end if ++readers; // there is another reader signal(canRead); // allow waiting readers to proceed } // end beginRead // monitor entry called after reading monitorEntry void endReadO { --readers; // there are one fewer readers //if no more readers are reading, allow a writer to write if (readers == 0) { signal (canWrite); // allow a writer to proceed } // end if } // end endRead // monitor entry called before performing write monitorEntry void beginWrite() { / / w a i t i f readers are reading o r i f a w r i t e r i s w r i t i n g if (readers > 0 || writeLock) { w a i t (canWrite) ; / / w a i t u n t i l w r i t i n g i s allowed } / / end i f Figure 6.3 | Monitor pseudocode for solving the readers and writers problem. (Part 1 of 2.) 6.2 Monitors 259 45 writeLock = true; // "lock out all readers and writers 46 } // end beginWrite 47 48 // monitor entry called after performing write 49 monitorEntry void endWrite() 50 51 52 53 54 55 56 57 58 59 60 61 62 63 { writeLock = false; // release lock // if a reader is waiting to enter, signal a reader if (queue(canRead)) { signal(canRead); // cascade in waiting readers } // end if else // signal a writer if no readers are waiting { signal(canWrite); // one waiting writer can proceed } // end else } // end endWrite Figure 6.3 | Monitor pseudocode for solving the readers and Writers problem. (Part 2 of 2.) reduced to zero (line 29), then one waiting writer (if there is one) becomes active (line 31). If a new reader cannot proceed (line 13), it waits on the condition variable canRead (line 15). If a new writer cannot proceed (line 40), it waits on the condition variable canWrite (line 42). When a reader wishes to read, it calls monitor entry beginRead (lines 8-21); a reader that has finished calls endRead (lines 23-34). In beginRead, a new reader can proceed as long as no thread is writing and no writer thread is waiting to write (line 13). The latter condition is important for preventing indefinite postponement of waiting writers; it is tested by using the boolean function, queue, which determines whether or not threads are waiting on the condition variable specified in its argument. Note that procedure begi nRead ends by signaling canRead (line 20) to allow another waiting reader to begin reading. This causes the next reader in the queue of waiting readers to become active and, in turn, signal the next waiting reader to proceed. This "chain reaction," also called a cascade, will continue until all waiting readers have become active. While this chaining is progressing, all arriving threads are forced to wait because the monitor observes the rule that signaled threads are serviced before arriving threads. If arriving readers were allowed to proceed, then a continuous stream of arriving readers would cause indefinite postponement of waiting writers. Since the readers do not interfere with one another and can be run in parallel on multiprocessor systems, this is an efficient way to service these threads. When a thread is done reading, it calls monitor entry endRead (lines 23-34), which decrements the number of readers by 1 (line 26). Eventually, this decrementing causes the number of readers to become zero, at which point the thread signals 260 Concurrent Programming canWrite (line 31) to allow a waiting writer, if there is one, to proceed (preventing indefinite postponement of waiting writers). When a thread wishes to write, it calls monitor entry beginWrite (lines 3646). Because a writer must have exclusive access, if there are any readers or if there is an active writer (line 40), the new writer must wait on condition variable canWrite (line 42). When the writer is able to proceed (because canWrite is signaled in line 31 or line 60), writeLock is set to t r u e (line 45). This keeps out any other readers and writers. When a writer finishes, it calls monitor entry endWrite (lines 48-63).This procedure sets writeLock to f a l s e (line 51) so that either readers or another writer can become active. The monitor then must signal another waiting thread to proceed (lines 53-61). Should it give preference to a waiting reader or a waiting writer? If it gives preference to a waiting writer, then it will be possible for a steady stream of incoming writers to cause the indefinite postponement of waiting readers. Therefore, as a writer finishes, it first checks if there is a waiting reader (line 54). If so. then can Read is signaled (line 56) and the waiting reader proceeds (and this, of course, cascades in all waiting readers). If there is no waiting reader, then canWrite is signaled (line 60) and a waiting writer is allowed to proceed. Self Review 1. Figure 6.3 cascades in waiting readers, but never cascades in waiting writers. Why? 2. What safeguard is there against a steady stream of readers indefinitely postponing a writer? Ans: 1) Readers don't interfere

with one another, so many readers can safely read at once. Each writer, however, must have exclusive access, or indeterminate results could occur. 2) In monitory entry beginRead, if there are any writers waiting to write (line 13), then a thread that wishes to read must wait on condition variable canRead (line 15). 6.3 Java Monitors In the following sections, we present complete, working Java multithreaded solutions to common concurrent programming problems. Monitors are associated with every object created in Java. Monitors are also the primary mechanism to provide mutual exclusion and synchronization in multithreaded Java applications. The keyword synchronized imposes mutual exclusion on an object in Java. In this section, we explain how Java monitors differ from the pseudocode monitors discussed in the previous section. When a thread attempts to execute a method protected by a Java monitor (i.e., the method is declared synchronized), it must first enter that monitor's entry set (commonly referred to as an entry queue), which is a queue of threads awaiting entry into the monitor. If there is no contention for entry into the monitor, a thread will immediately enter the monitor. If a thread is already inside the monitor, other threads must remain in the entry set until the monitor becomes available. 6.4 Producer/Consumer Relationship in Java 261 Java monitors are typically referred to as signal-and-continue monitors.50 Recall that a signal-and-continue monitor allows a thread inside the monitor to signal that the monitor will soon become available, but still maintain a lock on the monitor until it exits. A thread can exit the monitor by waiting on a condition variable or by completing execution of the code protected by the monitor. A thread, executing in a monitor, that must wait on a condition variable issues the wait call. The wait method causes the thread to release the lock on the monitor and wait on an unnamed condition variable. After calling wai t, a thread is placed in the wait set (also commonly referred to as the wait queue), a set of threads waiting to reenter the monitor for the object. Threads remain in the wait set until sig--ed (notified) by another thread. Because the condition variable is implicit in Java, a thread may be signaled, reenter the monitor and find that the condition on which it waited has not been met. Consequently, a thread may be signaled several times before the condition on which it is waiting is met. Threads issue signals by calling method notify or notifyAll. The notify method wakes a single thread in the wait set. The algorithm that determines which thread will enter the monitor next varies, depending on the Java virtual machine (JVM) implementation. As a result, the programmer cannot rely on a particular queuing discipline when noti f y is called. Another pitfall of the notify method is that the order in which some JVM implementations remove threads from the entry and wait sets might introduce lengthy service delays for particular threads in these sets—creating the possibility of indefinite postponement. As a result, if more than two threads access a monitor, it is best to use the notifyAll method, which wakes all threads in the entry and wait sets. When all threads are awake, the thread scheduler determines which thread acquires the monitor. The thread scheduler employs a queue discipline that prevents indefinite postponement.51, 52 Because notifyAll wakes all threads waiting to enter the monitor (as opposed to notify, which wakes a single thread), notifyAll incurs more overhead than notify. In the simple case of two-thread synchronization, noti f y yields higher performance than notifyAll without suffering from indefinite postponement. Self Review 1. What is the difference between signal-and-continue monitors and signal-and-exit monitors? Which type does Java use? Ans: 1) Java uses signal-and-continue monitors. They allow a thread to signal that it will soon be exiting the monitor, yet retain control of the monitor. Signal-and-exit monitors require that a thread release its lock on the monitor immediately after signaling. 6.4 Java Multithreading Case Study, Part III: Producer/Consumer Relationship in Java In this section, we present a Java implementation of the producer/consumer relationship investigated in Section 5.2.1, Java Multithreading Case Study, Part II. The 262 Concurrent Programming application in Fig. 6.4 and Fig. 6.5 demonstrates a producer and a consumer accessing a single shared buffer, with synchronization provided by a monitor. In this case, the producer produces a new value only when the buffer is empty, making the buffer full; the consumer consumes a value only when the buffer is full, making the buffer empty. This example reuses interface Buffer (Fig. 5.1) and classes Producer (Fig. 5.2) and Consumer (Fig. 5.3) from the example of Section 5.2.1. Reusing these classes from the example without synchronization enables us to demonstrate that the threads accessing the shared object are unaware that they are being synchronized. The code that performs the synchronization is placed in the s e t and get methods of class SynchronizedBuffer (Fig. 6.4), which implements interface Buffer (line 4). Thus, the Producer's and Consumer's run methods simply call the shared object's s e t and get methods as in the example of Section 5.2.1. Class SynchronizedBuffer (Fig. 6.4) contains two fields—buffer (line 6) and occupiedBuffers (line 7). Method s e t (lines 9-43) and method get (lines 4579) are now synchronized methods (lines 10 and 46); thus, only one thread can enter any of these methods at a time on a particular SynchronizedBuffer object. Although occupiedBuffers (line 7) is logically a condition variable, it is of type i nt; there is no object or type in Java that directly represents a condition variable. The methods in SynchronizedBuffer use occupiedBuffers in conditional expressions (lines 16 and 52) to determine whether it is the producer's or the consumer's turn to perform a task. If occupiedBuffers is zero, buffer is empty and the producer can place a value into buffer (line 34). This condition (line 52) also means that the consumer must wait (line 59) in the get method to read the value of buffer (again, because it is empty). If occupi edBuffers is one, the consumer can read the value from buffer, because it contains new information. This condition (line 16) also means that the producer must wait to place a value into buffer, because it is currently full. 1 2 3 4 5 6 7 8 9 10 11 // Fig. 6.4: SynchronizedBuffer.Java // SynchronizedBuffer synchronizes access to a shared integer. public class SynchronizedBuffer implements Buffer { private int buffer = -1; // shared by producer and consumer private int occupiedBuffers = 0; // counts occupied buffers // place value into buffer public synchronized void set(int value) { 12 13 14 Figure 6.4 | (Part 1 of 3.) // for display, get name of thread that called this method String name = Thread.currentThread() .getName() ; SynchronizedBuffer synchronizes access to a shared integer. 6.4 Producer/Consumer Relationship in Java 15 16 // while there are no empty buffers, place thread in waiting state while (occupiedBuffers == 1) { 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 // output thread and buffer information, then wait try { System.err.println(name + " tries to write."); displayState("Buffer full. " + name + " waits."); wait(); } // end try // if waiting thread interrupted, print stack trace catch (InterruptedException exception) { exception.printStackTrace(); } // end catch } // end while buffer = value; // set new buffer value // indicate producer cannot store another value // until consumer retrieves current buffer value ++occupiedBuffers; displayStateC name + " writes " + buffer); notify(); // tell waiting thread to enter ready state } // end method set; releases lock on SynchronizedBuffer // return value from buffer public synchronized int get() { // for display, get name of thread that called this method String name = Thread.currentThread().getName(); // while no data to read, place thread in waiting state while (occupiedBuffers == 0) { // output thread and buffer information, then wait try { System.err.println(name + " tries to read."); displayStateC "Buffer empty. " + name + " waits."); Figure 6.4 | SynchronizedBuffer synchronizes access to a shared integer. (Part 2 of 3.) 263 264 Concurrent Programming 59 60 61 w a i t ( ) ; / / w a i t u n t i l b u f f e r contains new value } // end try // if waiting thread interrupted, print stack trace 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 / / i f waiting thread interrupted, print stack trace catch (InterruptedException exception) { exception.printStackTrace(); } // end catch } // end while // indicate that producer can store another value // because consumer just retrieved buffer value --occupiedBuffers; displayState(name + " reads " + buffer); notify(); // tell waiting thread to become ready return buffer; } // end method get; releases lock on SynchronizedBuffer // display current operation and buffer state public void displayState(String operation) { StringBuffer outputLine = new StringBuffer(operation); outputLine.setLength(40); outputLine.append(buffer + "\t\t" + occupiedBuffers); System.err.println(outputLine); System.err.println(); } // end method displayState } // end class SynchronizedBuffer Figure 6.4 | Synchronizedbuffer synchronizes access to a shared integer. (Part 3 of 3.) When the Producer thread's run method invokes synchronized method set (from line 26 of Fig. 5.2), the thread attempts to acquire a lock on the SynchronizedBuffer monitor object. [Note:When we talk about "acquiring a lock" on a Java monitor, we mean "gaining mutually exclusive access to a monitor" in the generic discussion of monitors in Section 6.2, Monitors.] If the lock is available, the Producer thread acquires the lock. Then the loop in method set (lines 16-32 of Fig. 6.4) determines whether occupiedBuffers is equal to one. If so, buffer is full, so line 21 outputs a message indicating that the Producer thread is trying to write a value, and line 22 invokes method displayState (lines 81-89) to output another message indicating that the buffer is full and that the Producer thread is in the waiting state. Note that method 6.4 Producer/Consumer Relationship in Java 265 method d i s p l a y S t a t e is not declared synchronized because it is called in method main (line 18 of Fig. 6.5) before the producer and consumer threads are created and thereafter only from inside Synchronized methods get and s e t (lines 10 and 46 of Fig.6.4). Only one thread can execute inside the SynchronizedBuffer's monitor at once and di spl a y S t a t e is accessed only from inside the monitor, so mutually exclusive access is enforced without needing to declare d i s p l a y S t a t e Synchronized. Line 23 invokes method wait (inherited from Object by SynchronizedBuffer; all Java classes inherit from Object directly or indirectly) to place the thread that called method s e t (i.e., the Producer thread) in the waiting state for the SynchronizedBuffer object. The call to wait causes the calling thread to release the lock on the SynchronizedBuffer object. This is important because the thread cannot currently perform its task—and holding the lock would prevent other threads from accessing the object. This would result in deadlock, because the condition variable on which the first thread was waiting would never change. After the producer thread calls wait in line 23, another thread can attempt to acquire the SynchronizedBuffer object's lock and invoke the object's s e t or get methods— in particular, a consumer thread can now empty the buffer, eventually allowing the waiting producer to proceed. The producer thread remains in the waiting state until it is notified by another thread that it may proceed—at which point the producer thread returns to the ready state and waits for a processor. When the producer thread returns to the running state, the thread implicitly attempts to reacquire the lock on the SynchronizedBuffer object. If the lock is available, the producer reacquires the lock and method s e t continues executing with the next statement after wai t. Because wai t is called in a loop (lines 15-32), the loop continuation condition is tested to determine whether the thread can proceed with its execution. If not, wai t is invoked again; otherwise, method s e t continues with the next statement after the loop. Line 34 in method s e t assigns the argument value to buffer. Line 38 increments occupiedBuffers to indicate that the buffer now contains a value (i.e., a consumer can read the value, but a producer cannot put another value there yet). Line 40 invokes method di spl a y S t a t e to output a line to the console window indicating that the producer is writing a new value into the buffer. Line 42 invokes method n o t i f y (inherited from Object). If the consumer thread is waiting, it enters the ready state, where it can attempt its task again (as soon as the thread is assigned a processor). Method n o t i f y returns immediately and method s e t returns to its caller. [Note: Invoking method notify works correctly in this program because only one thread calls method get at any time (the consumer thread). Programs that have multiple threads waiting on a condition should invoke notifyAll to ensure that multiple threads receive notifications properly.] When method s e t returns, it implicitly releases the lock on the SynchronizedBuffer object. Methods get and s e t are implemented similarly. When the consumer thread's run method invokes synchroni zed method get (from line 28 of Fig. 5.3), the thread attempts to acquire a lock on the SynchronizedBuffer object. When it acquires the 266 Concurrent Programming lock, the while loop in method get (lines 51-68 of Fig. 6.4) determines whether occupiedBuffers is equal to 0. If so, the buffer is empty, so line 57 outputs a message indicating that the consumer thread is trying to read a value, and line 58 invokes method d i s p l a y S t a t e to output another message indicating that the buffer is empty and that the consumer thread waits. Line 59 invokes method wait to place the consumer thread that called method get in the waiting state for the SynchronizedBuffer object. Again, the call to wait causes the calling thread to release the lock on the SynchronizedBuffer object, so another thread can attempt to acquire the lock and invoke the object's s e t or get method. If the lock on the SynchronizedBuffer is not available (e.g., if the ProducerThread has not yet returned from method s e t) the consumer thread is blocked until the lock becomes available. The consumer thread object remains in the waiting state until it is notified by the producer thread that it may proceed—at which point the consumer thread returns to the ready state and waits a processor. When it returns to the running state, the thread implicitly attempts to reacquire the lock on the SynchronizedBuffer object. If the lock is available, the consumer thread reacquires it and method get continues executing with the next statement after the wait. Because wait is called in a loop (lines 51-68), the loop continuation condition is tested to determine whether the thread can proceed. If not, wait is invoked again; otherwise, method get continues with the next statement after the loop. Line 72 decrements occupiedBuffers to indicate that buffer is now empty, line 74 outputs a line to the console window indicating the value the consumer just read and line 76 invokes method notify. If the producer thread is waiting for the lock on this SynchronizedBuffer object, it enters the ready state. As soon as it is assigned a processor, the thread will attempt to reacquire the lock and continue performing its task. Method n o t i f y returns immediately, then method get returns the value of buffer to its caller (line 78). [Note: Again, invoking method n o t i f y works correctly in this program because only one thread calls method set at any time (the producer thread).] When method get returns, the lock on the SynchronizedBuffer object is released. Class SharedBufferTest2 (Fig. 6.5) is similar to class SharedBufferTest (Fig. 5.5). SharedBufferTest2 contains method main (lines 6-27), which launches the application. Line 9 instantiates a shared SynchronizedBuffer and assigns its reference to SynchronizedBuffer variable sharedLocation. We use a Synchroni zedBuffer variable rather than a Buffer variable, so that mai n can invoke SynchronizedBuffer method d i s p l a y S t a t e , which is not declared in interface Buffer. The SynchronizedBuffer object stores the data that is shared between the producer and consumer threads. Lines 11-18 display the column heads for the output. Lines 21-22 create a Producer object and a Consumer object, respectively, and pass sharedLocation to each constructor, so each object is initialized with a reference to the same SynchronizedBuffer. Next, lines 24-25 invoke method s t a r t on the producer and consumer threads to place them in the ready state. This launches these threads and sets up the initial call to each thread's run method. Finally, method main terminates and the mai n thread of execution dies. 6.4 Producer/Consumer Relationship in JAVA 267 Study the three sample outputs in Fig. 6.5. Observe that every integer produced is consumed exactly once. The synchronization and the variable occupiedBuffers ensure that the producer and consumer can execute only when it is their turn. The producer must go first; the consumer must wait if the producer has not produced since the consumer last consumed; the producer must wait if the consumer has not yet consumed the value that the producer most recently produced. In the first and second sample outputs, notice the comments indicating when the producer and consumer must wait to perform their respective tasks. In the third sample output, notice that the producer and consumer were fortuitously able to perform their tasks without waiting. Self Review 1. Why does the displ ayState method not need to be declared as synchronized? 2. (T/F) If the noti f y method is never called in the get method, the producer will never finish producing because it will be indefinitely postponed at line 23 of Fig. 6.4. Ans: 1) Although occupi edBuffers is a shared resource, the only time that di spl ayState is accessed is inside synchronized methods or when the program contains only one thread. 2) False. It is possible that the buffer will never fill, in which case the producer will never execute line 23. This is the case in the third output of Fig. 6.5. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 // Fig. 6.5: SharedBufferTest2.Java // SharedBufferTest2creates producer and consumer threads. public class SharedBufferTest2 { public s t a t i c void main(String [] args) { // create shared object used by threads SynchronizedBuffer sharedLocation = new SynchronizedBuffer(); // Display column heads for output StringBuffer columnHeads = new StringBuffer("Operation"); columnHeads.setLength(40); columnHeads.appendC "Buffer\t\tOccupied Count"); System.err.println(columnHeads); System.err.println(); sharedLocation.displayStateC "Initial State"); // create producer and consumer objects Producer producer = new Producer(sharedLocation); Consumer consumer = new Consumer(sharedLocation); producer.start(); // start producer thread Figure 6.5 | Threads modifying a shared object with synchronization. (Part 1 of 4.) 268 Concurrent Programming 25 26 27 28 29 consumer.start(); // start consumer thread } // end main } // end class SharedBufferTest2 Sample Output 1: Operation Buffer Occupied Count Initial State -1 0 Consumer tries to read. Buffer empty. Consumer waits. -1 0 Producer writes 1 1 1 Consumer reads 1 1 0 Consumer tries to read. Buffer empty. Consumer waits. 1 Producer writes 2 2 0 0 1 Consumer reads 2 2 0 Producer writes 3 3 1 Consumer reads 3 3 0 Producer writes 4 4 1 Producer done producing. Terminating Producer. Consumer reads 4 4 0 Consumer read values totaling: 10. Terminating Consumer. Figure 6.5 | Threads modifying a shared object with synchronization. (Part 2 of 4.) 6.4 Producer/Consumer Relationship in Java 269 Sample Output 2: Operation Buffer Occupied Count Initial State -1 0 Consumer tries to read. Buffer empty. Consumer waits. -1 0 Producer writes 1 1 1 Consumer reads 1 1 0 Producer writes 2 2 1 Producer tries to write. Buffer full. Producer waits. 2 1 Consumer reads 2 2 0 Producer writes 3 3 1 Consumer reads 3 3 0 Producer writes 4 4 1 Producer done producing, Terminating Producer. Consumer reads 4 4 0 Initial State Consumer tries to read. Buffer empty. Consumer waits. Producer writes 1 Consumer read values totaling: 10. Terminating Consumer. Sample Output 3: Operation Buffer Occupied Count Initial State -1 0 Producer writes 1 1 1 Figure 6.5 | Threads modifying a shared object with synchronization. (Part 3 of 4.) 270 Concurrent Programming Sample Output 3 (Cont.): Operation Buffer Occupied Count Initial State -1 0 Producer writes 1 1 1 Consumer reads 1 1 0 Producer writes 2 2 1 Consumer reads 2 2 0 Producer writes 3 3 1 Consumer reads 3 3 0 Producer writes 4 4 1 Producer done producing. Terminating Producer. Consumer reads 4 4 0 Consumer read values totaling: 10. Terminating Consumer. Figure 6.5 | Threads modifying a shared object with synchronization (Part 4 of 4.) 6.5 Java Multithreading Case Study, Part IV: Circular Buffer in Java The program of Section 6.4 uses thread synchronization to guarantee that two threads manipulate data in a shared buffer correctly. However, the application may not perform optimally. If the two threads operate at different speeds, one of them will spend more (or most) of its time waiting. If the producer thread produces values faster than the consumer can consume them, the producer will spend most of its time waiting for the consumer. Similarly, if the consumer consumes faster than the producer can produce them, the consumer will spend most of its time waiting for the producer. Even when the threads operate at the same relative speeds, over a period of time, occasionally those threads may become "out of sync," causing one of them to wait for the other. We cannot and should not make assumptions about the relative speeds of asynchronous concurrent threads. Too many interactions occur 6.5 with the operating system, the network, the user and other system components that . can cause the threads to operate at different and unpredictable speeds. When this happens in the producer/consumer example, one thread must wait. When threads spend a significant portion of their time waiting, programs may become less efficient, the system may become less responsive to interactive users and applications may suffer long delays because the processor is not used efficiently. To minimize the waiting time for threads that share resources and operate at the same average speeds, we know from earlier in this chapter that we can implement a circular buffer that provides extra buffer slots into which the producer can place values at times when it is running faster than the consumer and from which the consumer can retrieve those values when it is running faster than the producer. The key to using a circular buffer is to provide it with sufficient buffer slots to handle the anticipated "extra" production. If, over a period of time, we determine that the producer often produces as many as three more values than the consumer can consume, we can provide a buffer of three or more slots to handle the extra production. If the number of buffer slots is too small, threads will wait more; if too large, that would waste memory. The Java program of Fig. 6.6-Fig. 6.7 demonstrates a producer and a consumer accessing a circular buffer (in this case, a shared array of three cells) with synchronization. In this version of the producer/consumer relationship, the consumer consumes a value only when the array is not empty and the producer produces a value only when the array is not full. Class Producer is slightly modified from the version presented in Fig. 5.2; this version produces values from 11 to 20 (rather than 1-4). Class Consumer is slightly modified from the version in Fig. 5.3; this version consumes ten (rather than four) values from the circular buffer. The significant changes of Fig. 6.4-Fig. 6.5 occur in Ci r c u l a r B u f f e r (Fig. 6.6), which replaces SynchronizedBuffer (Fig. 6.4). C i r c u l a r B u f f e r contains four fields. Array b u f f e r s (line 8) implements the circular buffer as a three-element integer array. Variable occupiedBuffers (line 11) is the condition variable that can be used to determine whether a producer can write into the circular buffer (i.e., when occupiedBuffers is less than the number of elements in buffers) and whether a consumer can read from the circular buffer (i.e., when occupiedBuffers is greater than 0). Variable readLocation (line 14) indicates the position in b u f f e r s from which the next value can be read by a consumer. Variable w r i t e L o c a t i o n (line 15) indicates the next location in buffers in which a value can be placed by a producer. 1 2 3 // Fig, 6.6: CircularBuffer.Java // CircularBuffer synchronizes access to an array of // shared buffers. Figure 6.6 | CircularBuffer controls access to a shared array of integers. (Part 1 of 5.) Circular Buffer in Java 271 272 Concurrent Programming 4 5 public class CircularBuffer implements Buffer { 6 7 8 9 10 11 12 13 14 15 16 17 18 // place value into buffer public synchronized void set(int value) { 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 // each array element is a buffer private int buffers[] = { -1, -1, -1 }; // occupiedBuffers maintains count of occupied buffers private int occupiedBuffers = 0; // variables that maintain read and write buffer locations private int readLocation = 0; private int writeLocation = 0; // get name of thread that called this method String name = Thread.currentThread().getName(); // while buffer full, place thread in waiting state while (occupiedBuffers == buffers.length) { // output thread and buffer information, then wait try { System.err.println("All buffers full. " + name + " waits."); w a i t ( ) ; / / w a i t u n t i l space i s a v a i l a b l e } // e n d t r y // if waiting thread interrupted, print stack trace catch (InterruptedException exception) { exception.printStackTrace(); } // end catch } // end while // place value in writeLocation of buffers buffers[writeLocation] = value; // output produced value System.err.println("\n" + name + " writes " + buffers[writeLocation] + " "); Figure 6.6 | SynchronizedBuffer controls access to a shared array of integers. (Part 2 of 5.) 6.5 Circular Buffer in Java 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 // indicate that one more buffer is occupied ++occupiedBuffers; // update writeLocation for future write operation writeLocation = (writeLocation + 1) % buffers.length; // display contents of shared buffers System.err.println(createStateOutput()); notify(); // return a waiting thread to ready state } // end method set // return value from buffer public synchronized int get() { // get name of thread that called this method String name = Thread.currentThread().getName(); // while buffer is empty, place thread in waiting state while (occupiedBuffers == 0) { // output thread and buffer information, then wait try { System.err.println("All buffers empty. " + name + " waits."); wait(); // wait until buffers contains new data } // end try // if waiting thread interrupted, print stack trace catch (InterruptedException exception) { exception.printStackTrace(); } // end catch } // end while // obtain value at current readLocation int readValue = buffers[readLocation]; // output consumed value System.err.println("\n" + name + " reads " + readValue + " "); // decrement occupied buffers value Figure 6.6 | SynchronizedBuffer controls access to a shared array of integers. (Part 3 of 5.) 273 274 Concurrent Programming 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 --occupiedBuffers; // update readLocation for future read operation readLocation = (readLocation + 1) % buffers.length; // display contents of shared buffers System.err.println(createStateOutput()); notify(); // return a waiting thread to ready state return readValue; } // end method get // create state output public String createStateOutput() { // first line of state information String output = "(buffers occupied: " + occupiedBuffers + ")\nbuffers: "; for (int i = 0; i < buffers.length; ++i) { output += " " + buffers[i] + " "; } // end for // second line of state information output += "\n "; for (int i = 0; i < buffers.length; ++i) { output += "---- "; } // end for // third line of state information output += "\n "; // append readLocation (R) and writeLocation (W) // indicators below appropriate buffer locations for (int i = 0; i < buffers.length; ++i) { if (i == writeLocation && writeLocation == readLocation) { output += " WR "; } // end if Figure 6.6 | Synchronizedbuffer controls access to a shared array of integers. (Part 4 of 5.) 6.5 Circular Buffer in Java 275 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 else if (i == writeLocation) { output += " W "; } // end if else if (i == readLocation) { output += " R "; } // end if else { output += " "; } // end else } // end for output += "\n"; return output; } // end method createStateOutput } // end class CircularBuffer Figure 6.6 | SynchronizedBuffer controls access to a shared array of integers. (Part 5 of 5.) Method s e t (lines 17-59) performs the same tasks as it did in Fig. 6.4, with a few modifications. The while loop in lines 23-40 determines whether the producer must wait (i.e., all buffers are full). If so, lines 29-30 output that the producer is waiting to perform its task. Then line 31 invokes method wait to place the producer thread in the waiting state for the Ci r c u l a r B u f f e r object. When execution eventually continues in line 43 after the whi l e loop, the value written by the producer is placed in the circular buffer at location writeLocation. Next, lines 46-47 output the value produced. Line 50 increments occupiedBuffers—there is now at least one value in the buffer that the consumer can read. Then, line 53 updates writeLocation for the next call to CircularBuffer method s e t . The output continues at line 56 by invoking method c r e a t e S t a t e O u t put (lines 107-157), which outputs the number of occupied buffers, the contents of the buffers and the current writeLocation and readLocation. Finally, line 58 invokes method n o t i f y to indicate that the consumer thread waiting on the CircularBuffer object (if indeed the consumer is waiting) should proceed. Method get (lines 61-105) of class C i r c u l a r B u f f e r also performs the same tasks as it did in Fig. 6.4, with a few minor modifications. The while loop in lines 67-84 determines whether the consumer must wait (i.e., all buffer slots are empty). If the consumer thread must wait, lines 73-74 output that the consumer is waiting to perform its task. Then, line 75 invokes method wait to place the consumer thread in the waiting state for the C i r c u l a r B u f f e r object. When execution eventually 276 Concurrent Programming continues at line 87 after the while loop, readValue is assigned the value at location readLocation in the circular buffer. Lines 90-91 output the consumed value. Line 94 decrements the occupiedBuffers—there is now at least one open position in the buffer in which the producer thread can place a value. Then, line 97 updates readLocation for the next call to C i r c u l a r B u f f e r method get. Line 100 invokes method c r e a t e S t a t e O u t p u t to output the number of occupied buffers, the contents of the buffers and the current w r i t e L o c a t i o n and readLocation. Finally, line 102 invokes method n o t i f y to indicate that the producer thread waiting for the C i r c u l a r B u f f e r object (if indeed the producer is waiting) should proceed, and line 104 returns the consumed value to the calling method. Note that because Java implements signal-and-continue monitors, this program does not require the output parameter discussed in Section 6.2.3, Monitor Example: Circular Buffer. Class C i r c u l a r B u f f e r T e s t (Fig. 6.7) contains the main method (lines 8—24) that launches the application. Line 13 creates the CircularBuffer object sharedLocation. Lines 19-20 create the producer and consumer threads, and lines 22-23 start them. The sample outputs include the current occupiedBuffers, the contents of the buffers and the current w r i t e L o c a t i o n and readLocation. In the output, the letters W and R represent the current w r i t e L o c a t i o n and readLocation. respectively. Notice that, after the third value is placed in the third element of the buffer, the fourth value is inserted back at the beginning of the array—this is the circular buffer effect. 1 2 3 4 5 6 // Fig. 6.7: CircularBufferTest.Java // CircularBufferTest shows two threads manipulating a // circular buffer. // set up the producer and consumer threads and start them public class CircularBufferTest 7 { 8 public s t a t i c void main (String args[]) 9 { 10 11 12 13 14 15 16 17 18 19 20 21 // create shared object for threads; use a reference // to a CircularBuffer rather than a Buffer reference // to invoke CircularBuffer method createStateOutput CircularBuffer sharedLocation = new CircularBuffer(); // display initial state of buffers in CircularBuffer System.err.println(sharedLocation.createStateOutput()); // set up threads Producer producer = new Producer(sharedLocation); Consumer consumer = new Consumer(sharedLocation); Figure 6.7 | CircularBufferTest Figure 6.7 | CircularBufferTest instantiates producer and consumer threads. (Part 1 of 6.) 6.5 Circular Buffer in Java» 277 22 23 24 25 26 producer.start(); // start producer thread consumer.start(); // start consumer thread } // end main } // end class CircularBufferTest instantiates producer and consumer threads. (Part 2 of 6.) CircularBufferTest instantiates producer and consumer threads. 278 Concurrent Programming Figure 6.7 | (Part 3 of 6.) CircularBufferTest instantiates producer and consumer threads. 6.5 Circular Buffer in Java 279 Figure 6.7 | CircularBufferTest instantiates producer and consumer threads. (Part 4 of 6.) 2-80 foncurreftt Prtyrammjng Figure 6.7 | CircularBufferTest instantiates producer and consumer threads. Web Resources 281 Sample Output (Cont.): Consumer read values totaling: 155. Terminating Consumer. Figure 6.7 | CircularBufferTest instantiates producer and consumer threads. (Part 6 of 6.) Self Review 1. What are the potential costs and benefits of making the circular buffer larger? 2. What would happen if line 102 of Fig. 6.6 was omitted? Ans: 1) The potential benefit is that the producer can produce more and block less if the consumer is temporarily slower than the producer. However, this could result in wasted memory if both the consumer and producer work at the same speed, if one works much faster than the other or if the producer rarely, if ever, gets far enough ahead of the consumer to take advantage of the extra space. 2) The consumer thread would complete without issuing the notify. If the producer fortuitously never waits for the consumer (indeed a rare occurrence), then the system would function correctly. More commonly, however, if the producer were waiting it would never be able to proceed, i.e., the producer could be deadlocked. WebResources developer.java.sun.com/developer/Books/ performance2/chap4.pdf This page from Sun's Web site provides a detailed description of how to use Java monitors. hissa.nist.gov/rbac/5277/titlerpc.html Compares several popular RPC implementations. java.sun.com/docs/books/tutorial/essential/ threads/synchronization.html This is Sun's example of implementing mutual exclusion in the consumer/producer relationship in Java. Summary Concurrent programs are harder to write, debug, modify, and prove correct than non-concurrent programs. Nonetheless, there has been a surge of interest in concurrent programming languages because they enable us to express more naturally solutions to problems that are inherently parallel. The proliferation of multiprocessing systems, distributed systems and massively parallel architectures has also fueled the surge. A monitor is an object that contains both the data and procedures needed to perform allocation of one or more serially reusable shared resources. Monitor data is accessible only within the monitor; there is no way for threads outside the monitor to access monitor data directly. To accomplish a resource allocation function using monitors, a thread must call a monitor entry routine. Many threads might want to enter the monitor at various times, but mutual exclusion is rigidly enforced at the monitor boundary. A thread that tries to enter the monitor when it is in use is made to wait by the monitor. Eventually, the thread that has the resource will call a monitor entry routine to return the resource. There might be threads waiting for the resource, so the monitor entry routine calls signal to allow one of the waiting threads to acquire the resource and enter the monitor. To avoid indefinite postponement, a monitor gives higher priority to waiting threads than to newly arrived threads. Before a thread can reenter the monitor, the thread calling signal must first exit the monitor. A signal-and-exit 282 Concurrent Programming monitor requires a thread to immediately exit the monitor upon signaling. Alternatively, a signal-and-continue monitor allows a thread inside the monitor to signal that the monitor will soon become available, but still maintain a lock on the monitor until the thread exits the monitor. A thread can exit the monitor by waiting on a condition variable or by completing execution of the code protected by the monitor. A thread currently inside a monitor may need to wait outside the monitor until another thread performs an action inside the monitor. A monitor associates a separate condition variable with each distinct situation that might cause a thread to have to wait. Every condition variable has an associated queue. A thread calling wai t on a particular condition variable is placed into the queue associated with that condition variable; a thread calling signal on a particular condition variable causes a thread waiting on that condition variable (if there is such a thread) to be removed from the queue associated with that condition variable and to enter the monitor. In the circular buffer implementation of the solution to the producer/consumer problem, the producer deposits data in the successive elements of the shared array. The consumer removes them in the order in which they were deposited (FIFO). The producer can be several items ahead of the consumer. Eventually, the producer fills the last element of the array. When it produces more data, it must "wrap around" and again begin depositing data in the first element of the array. Because of the fixed size of the circular buffer, the producer will occasionally find all the array elements full; in this case the producer must wait until the consumer empties an array element. Similarly, there will be times when the consumer wants to consume, but the array will be empty; in this case the consumer must wait until the producer deposits data into an array element. If the two threads operate at different speeds, one of them will spend more (or even most) of its time waiting. If the producer thread produces values faster than the consumer can consume them, the producer thread spends most of its time waiting for the consumer to remove the next value from the array. Similarly, if the consumer thread consumes values faster than the producer can produce them, the consumer thread spends most of its time waiting for the producer to place the next value into the array. In computer systems, it is common to have some threads (called writers) that modify data and others (called readers) that read data. In the readers and writers problem, many readers may access the database at once. But a writer can modify the data, so it must have exclusive access. A new reader can proceed as long as no thread is writing and no writer thread is waiting to write. Each new reader signals the next waiting reader to proceed. This causes a "chain reaction" that continues until all waiting readers have become active. While this chaining is in progress, all arriving threads are forced to wait. During the chaining, arriving readers cannot enter the monitor, because the monitor observes the rule that signaled threads are serviced before arriving threads. If arriving readers were allowed to proceed, then a continuous stream of arriving readers would indefinitely postpone waiting writers. When the last reader leaves the monitor, the thread signals a waiting writer to proceed. When a writer finishes, it first checks if there is a waiting reader. If there is, the waiting reader proceeds (again cascading in all waiting readers). If there is no waiting reader, a waiting writer is allowed to proceed. Monitors are the primary mechanism providing mutual exclusion and synchronization in multithreaded Java applications. The keyword synchronized imposes mutual exclusion on an object in Java. Java monitors are signal-and-continue monitors, allowing a thread to signal that the monitor will soon become available, but still maintain a lock on the monitor until the thread exits the monitor. In Java, the wai t method causes the calling thread to release the lock on the monitor and wait on an unnamed condition variable. After calling wai t, a thread is placed in the wait set. A thread remains in the wait set until signaled by another thread. Because the condition variable is implicit in Java, a thread may be signaled, reenter the monitor and find that the condition on which it waited has not been met. Threads issue signals by calling method notify or notifyAll. The noti f y method wakes a single thread in the wait set. If more than two threads may access a monitor, it is best to use the notifyAll method, which wakes all threads in the entry and wait sets. Because notifyAll wakes all threads attempting to enter the monitor (instead of a single thread), notify can yield higher performance in some applications. Key Terms bounded buffer—See circular buffer. circular buffer—In the producer/consumer relationship, a fixed-size region of shared memory that stores multiple Exercises 283 values produced by a producer. If the producer occasionally produces values faster than the consumer, a circular buffer reduces the time the producer spends waiting for a consumer to consume the values, when compared to a buffer that stores a single value. If the consumer temporarily consumes values faster than the producer, a circular buffer can similarly reduce the time a consumer spends waiting for the producer to produce values. notify—Java method that wakes one thread in a monitor's wait set. The thread that is awakened depends on the JVM implementation. condition variable—Variable that contains a value and an associated queue. When a thread waits on a condition variable inside a monitor, it exits the monitor and is placed in the condition variable's queue. Threads wait in the queue until signaled by another thread. serially reusable shared resource—Resource that can be used only by one thread at a time. entry queue—See entry set. entry set —In Java, a queue of threads waiting to enter a monitor after calling a synchronized method. information hiding—Software architectural technique that facilitates the development of more reliable software systems by preventing direct access to data within an object by outside objects. monitor—Concurrency construct that contains both the data and procedures needed to provide mutual exclusion while allocating a serially reusable shared resource or group of serially reusable shared resources. monitor entry routine—Monitor routine that can be called by any thread, but that can be executed by only one thread at a time. Unlike private monitor routines, which can be called only by threads executing inside the monitor, monitor entry routines enforce mutual exclusion. notifyAll —Java method that awakens all threads in a monitor's wait and entry sets. Method notifyAll ensures that waiting threads are not indefinitely postponed, but incurs more overhead than notify. signal-and-continue monitor—Monitor that allows a thread to signal that the monitor is available, but does not require the thread to release the lock until it exits the monitor, at which point a signaled thread may enter the monitor. signal-and-exit monitor—Monitor that requires a thread to release the lock on the monitor as soon as the thread signals another thread. Solo—Small operating system created by Per Brinch Hansen to demonstrate fail-safe concurrent programming. synchronization—Coordination between asynchronous concurrent threads to sequentialize their access to shared resources. synchronized—Java keyword that imposes mutual exclusive access to code inside an object. wait queue—See wait set. wait set—In Java, a set of threads waiting to reacquire the lock on a monitor. Exercises 6.1 Compare and contrast the use of monitors and semaphore operations. 6.2 When a resource is returned by a thread calling a monitor, the monitor gives priority to a waiting thread over a new requesting thread. Why? 6.3 How do condition variables differ from conventional variables? Does it make sense to initialize condition variables? 6.4 The text has stated repeatedly that no assumptions should be made about the relative speeds of asynchronous concurrent threads. Why? 6.5 What factors, do you suppose, would affect a designer's choice of the number of slots a circular buffer should have? 6.6 Why is it considerably more difficult to test, debug, and prove program correctness for concurrent programs than for sequential programs? 6.7 The text states that information hiding is a system structuring technique that contributes to the development of more reliable software systems. Why, do you suppose, is this so? 6.8 Refer to the monitor described in Fig. 6.2 and answer each of the following questions. 6.4 The text has stated repeatedly that no assumptions should be made about the relative speeds of asynchronous concurrent threads. Why? a. Which procedure places data into the circular buffer? 6.5 What factors, do you suppose, would affect a designer's choice of the number of slots a circular buffer should have? b. Which procedure removes data from the circular buffer? c. Which queuing discipline best describes the operation of the circular buffer? 284 Concurrent Programming d. Is this true: w r i t e r P o s i t i o n > = readerPosition? e. Which statements perform monitor initialization? f. Which statement(s) can "wake up" a thread waiting on a condition variable? g. Which statement(s) can put a thread "to sleep"? h. Which statement(s) ensure that the buffer "wraps around" i. Which statement(s) modify a shared critical variable to indicate that a slot in the buffer is available? 6.9 In the readers and writers monitor presented in Fig. 6.3 why does it make sense to cascade in all waiting readers? Could this cause indefinite postponement of waiting writers? Under what circumstances might you choose to limit the number of waiting readers you would initiate when reading is allowed? 6.10 (The Sleeping Barber Problem)53 A barbershop has a cutting room with one chair and a waiting room with n chairs. Customers enter the waiting room one at a time if space is available, otherwise they go to another shop. Each time the barber finishes a haircut the customer leaves to go to another store, and a waiting customer, if there is one, enters the cutting room and has a haircut. Customers may enter the waiting room one at a time, or waiting customers may enter the (empty) cutting room one at a time, but these events are mutually exclusive. If the barber discovers that the waiting room is empty, the barber falls asleep in the waiting room. An arriving customer, finding the barber asleep, wakes the barber and has a haircut; otherwise the arriving customer waits. Use a monitor to coordinate the operation of the barber and the clients. If you know Java, also implement your monitor in Java. 6.11 (The Cigarette Smokers Problem)54 [Note: One of the authors, HMD, worked with S. Patil and Jack Dennis in the Computation Structures Group at M.I.T.'s Project Mac] This has become one of the classic problems in concurrency control Three smokers are represented by threads S1, S2, and S3. Three vendors are represented by threads V1, V2, and V3. Each smoker requires tobacco, a wrapper, and a match to smoke; when these resources are available, the smoker smokes the cigarette to completion, then becomes eligible to smoke again. S1 has tobacco, S2 has wrappers, and S3 has matches. V1 supplies tobacco and wrappers, V2 supplies wrappers anc matches, and V3 supplies matches and tobacco. V1, V2, and V3 operate in mutual exclusion; only one of these threads can operate at a time and the next vendor cannot operate until resources supplied by the previous vendor have been consumed by a smoker. Use a monitor to coordinate the operation of the smoker and vendor threads. 6.12 Semaphores are at least as powerful as monitors. Show how to implement a monitor by using semaphores. 6.13 Use semaphores to solve the readers and writers problem. 6.14 Implement the readers and writers problem. Model your solution after the readers and writers solution in Fig. 6.2 and the Java-based circular-buffer solution in Fig. 6.6 and Fig. 6.7. 6.15 Should a waiting thread receive priority over a thread first attempting to enter a monitor? What priority scheme, if any, should be imposed on waiting threads? Suggested Project 6.16 Prepare a research paper on concurrent programming in video game consoles. Does the hardware provide mutual exclusion primitives to the application programmer? Suggested Simulation 6.17 Extend the producer/consumer solution without a circular buffer presented in Fig. 6.1 to handle multiple producers. Recommended Reading Solutions to problems in concurrent programming have been known for quite some time. Dijkstra55 was the first to truly address such concerns, and he was soon followed by many others.56, 57, 58, 59 Recent developments in mutual exclusion algorithms pertain to distributed systems and are beyond the scope of this chapter. However, the reader is encouraged to study Lamport's survey of the mutual exclusion problem and some powerful results.60, 61 Initial development of the concept of a monitor has been attributed to Brinch Hansen62 and Hoare.63 An excellent survey of monitors, which describes their history, taxonomy and usage, is presented by Buhr.64 For more detailed information about the Java programming language and concurrent programming, see Java How to Program65 and Concurrent. Programming in Java: Design Principles and Pattern, 2nd ed.66 The bibliography for this chapter is located on our Web site at www.deitel.com/books/os3e/Bibliography.pdf. Works Cited 285 Works Cited 1. "Rationale for the Design of the Ada Programming Language," ACM SIGPLAN Notices, Vol. 14, No. 6, June 1979, Part B. 2. Carvalho, O. S. F., and G. Roucairol, "On Mutual Exclusion in Computer Networks," Communications of the ACM, Vol. 26, No. 2, February 1983, pp. 146-147. 3. Liskov, B., and R. Scheifler, "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," ACM Transactions on Programming Languages and Systems, Vol. 5, No. 3, 1983, pp. 381-404. 4. Shatz, S. M., "Communication Mechanisms for Programming Distributed Systems," Computer, Vol. 17, No. 6, June 1984, pp. 21-28. 5. Fisher, D. A., and R. M. Weatherly, "Issues in the Design of a Distributed Operating System for Ada," Computer, Vol. 19, No. 5, May 1986, pp. 38-47. 6. Liskov, B. H.; M. Herlihy; and L. Gilbert, "Limitations of Synchronous Communication with Static Process Structure in Languages for Distributed Computing," Proceedings of the 13th ACM Symposium on Principles of Programming Languages, St. Petersburg, Florida, January 1986. 7. Shatz, S. M., and J. Wang, "Introduction to Distributed-Software Engineering," Computer, Vol. 20, No. 10, October 1987, pp. 23-32. 8. Roberts, E. S.; A. Evans, Jr.; C. R. Morgan; and E. M. Clarke, "Task Management in Ada—A Critical Evaluation for Real-Time Multiprocessors," Software—Practice and Experience, Vol. 11, No. 10, October 1981, pp. 1019-1051. 9. Nielsen, K. W., and K. Shumate, "Designing Large Real-Time Systems with Ada," Communications of the ACM, Vol. 30, No. 8, August 1987, pp. 695-715. 10. Ford, R., "Concurrent Algorithms for Real-Time Memory Management," IEEE Software,Vol. 5, No. 5, September 1988, pp. 10-24. 11. "Preliminary Ada Reference Manual," ACM SIGPLAN Notices, Vol. 14, No. 6, June 1979, Part A. 12. Brinch Hansen, P., "The Programming Language Concurrent Pascal," IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975, pp. 199-207. 13. Coleman, D.; Gallimore, R. M.; Hughes, J. W.; and M. S. Powell, "An Assessment of Concurrent Pascal," Software—Practice and Experience,Vol 9,1979, pp. 827-837. 14. Brinch Hansen, P., "Distributed Processes: A Concurrent Programming Concept," Communications of the ACM, Vol. 21, No. 11, November 1978, pp. 934-941. 15. Gehani, N H., and W. D. Roome, "Concurrent C," SoftwarePractice and Experience,Vol. 16, No. 9,1986, pp. 821-844. 16. Kieburtz, R. B., and A. Silberschatz, "Comments on 'Communicating Sequential Processes,' " ACM Transactions on Programming Languages and Systems, Vol. 1, No. 2,1979, pp. 218-225. 17. Hoare, C. A. R., Communicating Sequential Processes, Englewood Cliffs, NJ: Prentice-Hall, 1985. 18. Hoppe, J., "A Simple Nucleus Written in Modula-2: A Case Study," Software—Practice and Experience, Vol. 10, No. 9, September 1980, pp. 697-706. 19. Wirth, N., Programming in Modula-2, New York: Springer-Verlag, 1982. 20. Ogilvie, J. W. L., Modula-2 Programming, New York: McGraw-Hill, 1985. 21. McGraw, J. R., "The VAL Language: Description and Analysis," ACM Transactions on Programming Languages, Vol. 4, No. 1, January 1982, pp. 44-82. 22. Cook, R. P., "*MOD—A Language for Distributed Programming," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6,1980, pp. 563-571. 23. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Acta Informatica,Vol. 1,1971, pp. 115-138. 24. Brinch Hansen, P., "Structured Multiprogramming," Communications of the ACM, Vol. 15, No. 7, July 1972, pp. 574-578. 25. Brinch Hansen, P., Operating System Principles, Englewood Cliffs, NJ: Prentice Hall, 1973. 26. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557. Corrigendum, Communications of the ACM, Vol. 18, No. 2, February 1975, p. 95. 27. Brinch Hansen, P., "The Solo Operating System: Processes, Monitors, and Classes," Software—Practice and Experience, Vol. 6, 1976, pp. 165-200. 28. Howard, J. H., "Proving Monitors," Communications of the ACM, Vol. 19, No. 5, May 1976, pp. 273-279. 29. Howard, J. H., "Signaling in Monitors," Second International Conference on Software Engineering, San Francisco, October 1976, pp. 47-52. 30. Lister, A. M., and K. J. Maynard, "An Implementation of Monitors," Software—Practice and Experience, Vol. 6, No. 3, July 1976, pp. 377-386. 31. Kessels, J. L. W., "An Alternative to Event Queues for Synchronization in Monitors," Communications of the ACM, Vol. 20, No. 7, July 1977, pp. 500-503. 32. Keedy, J., "On Structuring Operating Systems with Monitors," Australian Computer Journal, Vol. 10, No. 1, February 1978, pp. 23-27, reprinted in Operating Systems Review, Vol. 13, No. 1, January 1979, pp. 5-9. 33. Lampson, B. W., and D. D. Redell, "Experience with Processes and Monitors in MESA," Communications of the ACM, Vol. 23, No. 2, February 1980, pp. 105-117. 34. Wegner, P., and S. A. Smolka, "Processes, Tasks and Monitors: A Comparative Study of Concurrent Programming Primitives," IEEE Transactions on Software Engineering, Vol. SE-9, No. 4, July 1983, pp. 446-462. 286 Concurrent Programming 35. Buhr, P. A.; Fortier, M.; and Coffin, M., "Monitor Classification," ACM Computing Surveys, Vol. 27, No. 1, March 1995, pp. 63-107. 36. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557. 37. Buhr, P. A.; Fortier, M.; and Coffin, M., "Monitor Classification," ACM Computing Surveys, Vol. 27, No. 1, March 1995, pp. 63-107. 38. Brinch Hansen, P., "The Programming Language Concurrent Pascal," IEEE Transactions on Software Engineering, No. 2 June 1975,199-206. 39. Syracuse University, "Biography." . 40. Syracuse University, "Biography." . 41. Brinch Hansen, P., Operating Systems Principles, Englewood Cliffs, NX: Prentice Hall, 1973. 42. Brinch Hansen, P., "The Purpose of Concurrent Pascal." Proceedings of the International Conference on Reliable Software, pp. 305-309,1975. 43. Syracuse University, "Biography". . 44. Brinch Hansen, P., "Java's Insecure Parallelism," SIGPLAN Notices, Vol. 34, No. 4, pp. 38-45, April 1999. 45. Brinch Hansen, P., "The Solo Operating System," Software—Practice and Experience,Vol. 6, No. 2, pp. 141-205, April-June 1976. 46. Brinch Hansen, P., "Joyce—A Programming Language for Distributed Systems," Software—Practice and Experience, Vol. 17, No. 1, January 1987, pp. 29-50. 47. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557. Corrigendum, Communications of the ACM, Vol. 18, No. 2, February 1975, p. 95. 48. Courtois, P. X; Heymans, F.; and D. L. Parnas, "Concurrent Control with Readers and Writers," Communications of the ACM, Vol. 14, No. 10, October 1971, pp. 667-668. 49. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557. Corrigendum, Communications of the ACM, Vol. 18, No. 2, February 1975, p. 95. 50. Hartley, S. X, "Concurrent Programming Using the Java Language," Modified: 30 December 1997, . 51. Venners, B. "Thread Synchronization", Chapter 20 of Inside the Java Virtual Machine, last updated April 18, 2003, . 52. Christopher, T. W., and G. K. Thiruvathukal, High-Performance Java Platform Computing, Multithreaded and Networked Programming, Upper Saddle River, NJ: Prentice Hall PTR, February 2001, pp. 89-93. 53. Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control," Communications of the ACM, Vol. 8, No. 5, September 1965,p.569. 54. Patil, S. S., "Limitations and Capabilities of Dijkstra's Semaphores Primitives for Coordination among Processes," M.I.T. Project MAC Computational Structures Group Memo 57, February 1971. 55. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Acta Informatica, Vol. 1,1971, pp. 115-138. 56. Knuth, D., "Additional Comments on a Problem in Concurrent Programming Control," Communications of the ACM,Vol. 9, No. 5, May 1966, pp. 321-322. 57. Bernstein, A. J. "Program Analysis for Parallel Processing," IEEE Transactions on Computers, Vol. 15, No. 5, October 1966, pp. 757-762. 58. Courtois, P. X; Heymans, E; and D. L. Parnas, "Concurrent Control with Readers and Writers," Communications of the ACM Vol. 14, No. 10, October 1971, pp. 667-668. 59. Lamport, L., "Concurrent Reading and Writing," Communications of the ACM, Vol. 20, No. 11, November 1977, pp. 806-811. 60. Lamport, L., "The Mutual Exclusion Problem: Part I —A Theory of Interprocess Communication," Journal of the ACM (JACM), Vol.33 No.2,April 1986,pp.313-326. 61. Lamport, L., "The Mutual Exclusion Problem: Part II —Statement and Solutions," Journal of the ACM (JACM),Vol. 33 No. 2, April 1986, pp. 327-348. 62. Brinch Hansen, P., "Structured Multiprogramming," Communications of the ACM, Vol. 15, No. 7, July 1972, pp. 574-578. 63. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557. Corrigendum, Communications of the ACM, Vol. 18, No. 2, February 1975, p. 95. 64. Buhr, P. A.; Fortier, M.; and M. Coffin, "Monitor Classification," ACM Computing Surveys,Vol 27, No. 1, March 1995, pp. 63-107. 65. Deitel, H. M. and P. J. Deitel, Java How to Program, 5th ed., Upper Saddle River, NJ: Prentice Hall, 2003. 66. Lea, D., Concurrent Programming in Java: Design Principles and Pattern, 2nd ed., Reading, MA: Addison-Wesley, 1999. We had better wait and see. -H.H.Asquith- Every man is the center of a circle, whose fatal circumference he cannot pass. —John James Ingalls— Hold me but safe within the bond Of one immortal look. — Robert Browning — Detection is, or ought to be, an exact science, and should be treated in the same cold and Unemotional manner. — Sir Arthur Conan Doyle — Delays have dangerous ends. —William Shakespeare — Chapter 7 Deadlock and Indefinite Postponement Objectives After reading this chapter, you should understand: • the problem of deadlock. • the four necessary conditions for deadlock to exist. • the problem of indefinite postponement. • the notions of deadlock prevention, avoidance, detection and recovery. • algorithms for deadlock avoidance and detection. • how systems can recover from deadlocks. Chapter Outline 7.1 Introduction 7.7.2 Denying the "No-Preemption" Condition 7.2 7.7.3 Denying the "Circular-Wait" Condition Examples of Deadlock 7.2.1 Traffic Deadlock Anecdote: One-lane Bridge 7.2.2 Simple Resource Deadlock Operating Systems Thinking: Waiting, Deadlock and Indefinite Postponement 7.2.3 Deadlock in Spooling Systems 7.2.4 Example: Dining Philosophers 7.3 Related Problem: Indefinite Postponement 7.4 Resource Concepts 7.5 Four Necessary Conditions for Deadlock 7.6 Deadlock Solutions 7.7 Deadlock Prevention Anecdote: No Nuts, Bolts or Screws Allowed 7.7.1 Denying the "Wait-For" Condition 7.8 Deadlock Avoidance with Dijkstra's Bankers Algorithm Anecdote: Acronyms 7.8.1 Example of a Safe State 7.8.2 Example of an Unsafe State 7.8.3 Example of Safe-State-to-UnsafeState Transition 7.8.4 Banker's Algorithm Resource Allocation 7.8.5 Weaknesses in the Banker's Algorithm 7.9 Deadlock Detection 7.9.1 Resource-Allocation Graphs 7.9.2 Reduction of Resource-Allocation Graphs 7.10 Deadlock Recovery 7.11 Deadlock Strategies in Current and Future Systems Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited 290 7.1 Introduction 291 7.1 Introduction In the previous four chapters, we discussed asynchronous, concurrent processes and threads. A multiprogrammed system provides many benefits, but as discussed in Chapter 6, Concurrent Programming, multiprogramming also introduces additional complexity. One problem that arises in multiprogrammed systems is deadlock. A process or thread is in a state of deadlock (or is deadlocked) if it is waiting for a particular event that will not occur. In a system deadlock, one or more processes are deadlocked. 1 '2 The remainder of this chapter will focus on processes, but most of the discussion applies to threads as well. In multiprogrammed computing systems, resource sharing is one of the primary goals. When resources are shared among a set of processes, each process maintaining exclusive control over particular resources allocated to it, deadlocks can develop in which some processes will never be able to complete execution. The result can be loss of work and reduced system throughput and system failure. This chapter discusses the problem of deadlock and summarizes the four major areas of deadlock research, namely deadlock prevention, avoidance, detection and recovery. It also considers the closely related problem of indefinite postponement, also called starvation, in which a process that is not deadlocked could wait for an event that might never occur or might occur unpredictably far in the future because of biases in the systems resource-scheduling policies. In some cases, the price for making a system free of deadlock is high. For some systems, such as those that are mission critical, the price must be paid no matter how high, because allowing a deadlock to develop could be catastrophic, especially if it puts human life at risk.This chapter also discusses solutions to the problems of deadlock and indefinite postponement in terms of trade-offs between their overhead and their anticipated benefits. Self Review 1. A system designer decides to try to avoid any possibility of deadlock by creating a multiprogrammed system that does not share resources. What is wrong with this idea? 2. Compare and contrast deadlock with indefinite postponement. Ans: 1) It would be highly inefficient, because each process would need its own set of resources. Also, asynchronous concurrent programs often require shared resources such as semaphores. 2) Deadlock and indefinite postponement are similar in that they occur when processes wait for an event. Deadlock occurs because the event will never occur; indefinite postponement occurs because it is uncertain when or even whether the event will ever occur (due to biases in a system's resource-scheduling policies). 7.2 Examples of Deadlock Deadlocks can develop in many ways. If a process is given the task of waiting for an event to occur, and if the system includes no provision for signaling that event, then we have a one-process deadlock. 3 Such deadlocks are extremely difficult to detect. Deadlocks in real systems often involve multiple processes competing for multiple resources of multiple types. Let us consider several common examples. 292 Deadlock and Indefinite Postponement 7.2.1 Traffic Deadlock Figure 7.1 illustrates a kind of deadlock that occasionally develops in cities. A number of automobiles are attempting to drive through a busy neighborhood, and the traffic is completely snarled. The police have to unwind the jam by slowly and carefully backing cars out of the congested area. Eventually the traffic begins to flow normally, but not without much annoyance, effort, and loss of time (see the Anecdote, One-Lane Bridge). Self Review 1. Assuming that there are no cars beyond the ellipses in Fig. 7.1, what minimum number of cars would have to back up to relieve the deadlock and which car(s) would they be? 2. If cars could be removed by airlifting in Fig. 7.1, what minimum number of cars, and which one(s), would have to be removed to relieve the deadlock? Figure 7.1 | Traffic deadlock example.4 7.2 Examples of Deadlock 293 Ans: 1) In Fig. 7.1, only two cars would need to back up to allow every other car to eventually move-any one of the cars abutting an ellipsis, then the car ahead of that one in the intersection. 2) Only one car has to be removed—namely, any one of the four cars in the intersections. 7.2.2 Simple Resource Deadlock Most deadlocks in operating systems develop because of the normal contention for dedicated resources (i.e., resources that may be used over time by many processes but by only one process at a time, sometimes called serially reusable resources). For example, a printer can print jobs for only one process at a time (otherwise pieces of many print jobs would be interwoven). A simple example of a resource deadlock is illustrated in Fig. 7.2. This resource-allocation graph shows two processes as rectangles and two resources as circles. An arrow from a resource to a process indicates that the resource belongs to, or has been allocated to, the process. An arrow from a process to a resource indicates that the process is requesting, but has not yet been allocated, the resource. The diagram illustrates a deadlocked system: Process P1 holds resource R 1 and needs resource R 2 to continue. Process P 2 holds resource R 2 and needs resource R1 to continue. Each process is waiting for the other to free a resource that the other process will not free. This circular wait is characteristic of deadlocked systems (see the Operating Systems Thinking feature, Waiting, Deadlock and Indefinite Postponement). Holding resources tenaciously in this way, a is sometimes referred to as a deadly embrace. Anecdote One-lane Bridge A former operating systems stu- drivers were approaching a one-said, "I don't back up for idiots." dent of HMD's was one of the original designers of the IBM personal computer. When we studied lane bridge. They reached the middle of the bridge at the same time, forcing a confrontation. The other driver calmly got back in his car and started backing up. He said, "No problem—I do." deadlock in class, he shared a deadlock anecdote w i t h me. Two They got out of their cars and went nose to nose. One driver Lesson to operating system designers: Processes are neither stubborn nor cooperative, so deadlock is a painful experience at best. Usually, one or more processes will have to be "backed out" of the deadlock, losing some or all of their work. 294 Deadlock and Indefinite Postponement Figure 7.2 | Resource deadlock example. This System is deadlocked because each process holds a resource being requested by the o t h e r process and neither process is willing to release the resource it holds. Self Review 1. If the traffic deadlock in Fig. 7.1 were to be represented using a resource-allocation grapr what would the processes be? What would the resources be? 2. Suppose several arrows are pointing from a process to resources in a resource-allocatk'graph. What does that mean? How does it affect the possibility of deadlock? Ans: l) The cars would be the processes. The resources would be the sections of the street th.. the cars occupy. Each car currently holds the section of the street directly below it and is requesting the section in front of it. 2) It means that the process is requesting several resourcesThe possibility of deadlock depends on whether those resources are allocated to other pro cesses, some of which, in turn, are requesting resources held by the process we are discussing. 7.2.3 Deadlock in Spooling Systems A spooling system improves system throughput by disassociating a program from slow devices such as printers. For example, if a program sending lines to the printer must wait for each page to be printed before it can transmit the next page, then it will execute slowly. To speed the program's execution, a spooling system routes output pages to a much faster device, such as a hard disk, where they are temporarily stored until they can be printed. Spooling systems can be prone to deadlock. Some spooling systems require that the complete output from a program be available before printing can begin. Several partially completed jobs generating pages to a spool file can become deadlocked if the disk's available space fills before any job completes. The user or system administrator may kill one or more jobs to make sufficient spooling space available for the remaining jobs to complete. 7.2 Examples of Deadlock 295 Typically, the system administrator specifies the amount of space for spooling files. One way to make deadlocks less likely is to provide considerably more space for spooling files than is likely to be needed. This solution may not be feasible if space is at a premium. A more common solution is to restrain the input spoolers so that they do not accept additional print jobs when the spooling files begin to reach some saturation threshold, such as 75 percent full. This may reduce system throughput, but it is the price paid to reduce the likelihood of deadlock. Today's systems are more sophisticated than this. They might allow printing to begin before the job is completed so that a full, or nearly full, spooling file can be emptied or partially cleared even while a job is still executing. This concept has been applied to streaming audio and video clips, where the audio and video begin to play before the clips down-load fully. In many systems spooling space allocation has been made more dynamic, so that if existing space starts to fill, then more space may be made available. Operating Systems Thinking Waiting, Deadlock and Indefinite Postponement Operating systems must manage many different types of waiting scenarios, which rely on an abundant use of queues to do so. Pro- mation, so it must wait for the consumer to empty some data; cease progressing; this can be fatal in mission-critical systems cesses and threads must wait for a processor to become available before they can execute; they often wait for I/O requests to producer to generate more data. the consumer must wait for the We will see extensive exam- (also called starvation) typically occurs when a process is waiting for a complete and for requested resources to become available. I/O pies of waiting scenarios throughout this book. Operating systems must manage waiting carefully to avoid a 'n o ' 'n serious problems, some of which are indefinite postponement and deadlock. Processes and threads sharing data may need to wait for a producer to get in line in front of the waiting process. A steady able. Processes and threads sharing data that finishes and leaves a waiting thread to be deadlocked, the process is of course, deadlock. In a deadlock scenario, processes stream of arriving high-priority processes is said to indefinitely postpone the lower-priority process. This could be as dangerous critical section. In the producer/ consumer relationship, the producer passes information to the and threads are waiting for events that will never happen; a simple two-process deadlock has as deadlock in a mission-critical system. So take waiting seriously—if it is mismanaged, system consumer via a shared region of each process waiting on a failures could occur. memory; when that region resource that only one process at 296 Deadlock and Indefinite Postponement Self Review 1. Suppose a spooling system has a saturation threshold of 75 percent and

limits the maximum size of each file to 25 percent of the total spooling file size. Could deadlock occur in this system? 2. Suppose a spooling system has a saturation threshold of 75 percent and limits the maximum size of each file to 25 percent of the total spooling file size. Describe a simple way to ensure that deadlock will never occur in the system. Explain how this could lead to inefficient resource allocation. Ans: 1) Yes, deadlock can still occur in this system. For instance, several jobs can begin transferring their outputs. When the spooling file reaches the 75 percent threshold, new jobs are not allowed. However, jobs that have begun are allowed to continue spooling, which may result in deadlock if there is insufficient space in the spooling file. 2) A simple adjustment would be to allow only one job to continue spooling data when the file reaches the threshold. This would be inefficient because it would limit the maximum job size to much less than the available spooling space. 7.2.4 Example: Dining Philosophers Dijkstra's problem of the Dining Philosophers,5, 6 illustrates many of the subtle problems inherent in concurrent programming. The problem is this: Five philosophers sit around a circular table. Each leads a simple life alternating between thinking and eating spaghetti. In front of each philosopher is a dish of spaghetti that is constantly replenished by a dedicated wait staff. There are exactly five forks on the table, one between each adjacent pair of philosophers. Eating spaghetti (in the most proper manner) requires that a philosopher use both adjacent forks (simultaneously). Develop a concurrent program free of deadlock and indefinite postponement that models the activities of the philosophers. If the solution to this problem were not free of deadlock and indefinite postponement, one or more philosophers would starve. The program must, of course, enforce mutual exclusion—two philosophers cannot use the same fork at once. A typical philosopher behaves as shown in Fig. 7.3. The problems of mutual exclusion, deadlock and indefinite postponement lie in the implementation of method eat. Consider the simple, yet dangerous implementation in Fig. 7.4. Because the philosophers operate asynchronously and con- 1 2 3 4 5 6 7 8 9 void typicalPhilosopher() { while (true) { think(); eat(); } // end while } // end typicalPhilospher Figure 7.3 | Dining philosopher behavior. 7.2 Examples of Deadlock 297 1 void eat() 2 { 3 pickUpLeftFork(); 4 pickUpRightFork(); 5 eatForSomeTime(); 6 putDownRightFork(); 7 putDownLeftFork(); 8 } // eat Figure 7.4 | Implementation of method eat currently, it is possible for each philosopher to execute line 3 before any philosopher executes line 4. In this case, each philosopher will hold exactly one fork, and no forks will remain available on the table. The philosophers will all deadlock and starve. One way to break the deadlock when each philosopher holds a left fork is to force one or more philosophers to put their left fork down so that another philosopher may grab it as a right fork. To implement this rule, pickllpRightForkO can specify that a philosopher puts down the left fork if that philosopher cannot obtain the right fork. However, in this case, it is possible (albeit unlikely) that each philosopher will pick up and put down its left fork repeatedly in tandem without ever obtaining the two forks it needs to eat spaghetti. In this case, the philosophers are not deadlocked, but "livelocked," suffering from indefinite postponement, and they will still starve. The solution must prevent deadlock (by forcing philosophers to put down forks) but also prevent livelock (indefinite postponement) by guaranteeing that each philosopher will obtain both forks from time to time. In the exercises, you will be asked to develop a complete solution to the Dining Philosophers problem. Self Review 1. Could the implementation of the eat method in Fig. 7.4 allow the philosophers to live in harmony without starving? 2. Consider the eat method in Fig. 7.4. Suppose lines 3 and 4 were replaced with pickUpBothForksAtOnce(); would that implementation prevent deadlock? Would it prevent the philosophers from starving? Ans: 1) Yes. We cannot predict the relative speeds of asynchronous concurrent processes. It is possible that not all five philosophers will pick up their left fork at once. Various philosophers could acquire the two forks they need, then release them to give the other philosophers a chance to eat. 2) Yes, it would prevent deadlock, assuming that pickUpBothForksAtOnce() can be performed atomically, but it would still allow indefinite postponement. This could occur, for example, if two of the philosophers were to continually pick up two forks, eat, put down the forks and pick them up again before any of the other philosophers could grab them; those other philosophers would starve. 298 Deadlock and Indefinite Postponement 7.3 Related Problem: Indefinite Postponement In any system that requires processes to wait as a result of resource-allocation and process scheduling decisions, a process may be delayed indefinitely while other processes receive the system's attention. This situation, called indefinite postponement, indefinite blocking, or starvation, can be as devastating as deadlock. Indefinite postponement may occur because of biases in a system's resourcescheduling policies. When resources are scheduled on a priority basis, it is possible for a given process to wait for a resource indefinitely, as processes with higher priorities continue to arrive. Waiting is a fact of life and is certainly an important aspect of what goes on inside computer systems, so systems should be designed to manage waiting processes fairly and efficiently. Some systems prevent indefinite postponement by increasing a process's priority gradually as it waits for a resource —this technique is called aging. Eventually, the waiting process's priority will exceed the priorities of all processes, and the it will be serviced. Self Review 1. Dekker's Algorithm and Peterson's Algorithm prevented a process from being indefinitely postponed from entering its critical section. Describe how indefinite postponement was prevented. How is this related to aging? 2. Suppose an interactive process appears to be "dead in the water." Does this mean that the process is definitely deadlocked? Could it be indefinitely postponed? Are there other possibilities? Ans: 1) Both Dekker's Algorithm and Peterson's Algorithm made the waiting process the favored process when the resource was available again. This is similar to aging, because the waiting process earns a higher priority by waiting. 2) This is an interesting question. From the user's perspective, an interactive process would appear "dead in the water" if the system simply stopped responding to the interactive user's requests. The process could be deadlocked, but it does not have to be. It could be suffering from indefinite postponement, but it does not have to be. It could also be waiting for an event that may happen shortly, it could be locked in an infinite loop or it could be doing a lengthy calculation that will soon finish. Anyone who has experienced a system that appears to "hang" knows how frustrating this can be. What services should the operating system provide to help the user in such situations? We address these issues throughout the book. 7.4 Resource Concepts As a resource manager, the operating system is responsible for the allocation of a vast array of resources of various types; this is part of what makes operating system design so interesting. We consider resources that are preemptible, such as processors and main memory. Processors are perhaps the most frequently preempted resources on a computer system. Processors must be rapidly switched (i.e., multiplexed) among all active processes competing for system service to ensure that these processes progress at a reasonable rates. Whenever a particular process reaches a point at which it cannot effectively use a processor (such as during a long 7.5 Four Necessary Conditions for Deadlock 299 wait for input/output completion), the operating system dispatches another process to that processor. As we will see in Chapters 9,10 and 11, a user program currently occupying a particular range of locations in main memory may be removed or preempted by another program. Preemption is thus extremely critical to the success of multiprogrammed computer systems. Certain resources are nonpreemptible; they cannot be removed from the pro----es to which they are assigned until the processes voluntarily release them. For example, tape drives and optical scanners are normally assigned to a particular process for periods of minutes or hours. Some resources may be shared among several processes, while others are dedicated to a single process at a time. Although a single processor can normally belong to only one process at a time, its multiplexing among many processes creates the illusion of simultaneous sharing. Disk drives are sometimes dedicated to single processes, but typically they contain files that can be accessed by many processes. Disks can be multiplexed among many processes requesting I/O. Data and programs certainly are resources that the operating system must control and allocate. On multiprogramming systems, many users may simultaneously want to use an editor program. If the operating system maintained in main memory a separate copy of the editor for each program, there would be a significant amount of redundant data, wasting memory. A better technique is for the operating system to load one copy of the code in memory and to make the copy available to each user. If a process were allowed to modify this shared code, other processes might behave unpredictably. As a result, this code must be reentrant, meaning the code is not modified as it executes. Code that may be changed but is reinitialized each time it is used is said to be serially reusable. Reentrant code may be shared by several processes simultaneously, whereas serially reusable code may be used correctly by only one process at a time. When we call particular resources shared, we must be careful to state whether they may be used by several processes simultaneously or by only one process at a time. The latter kind—serially reusable resources —are the ones that tend to become involved in deadlocks. Self Review 1. (T/F) Processes do not deadlock as a result of contending for the processor. 2. (T/F) Nonpreemptible resources must be hardware. Ans: 1) True. The processor is a preemptible resource that can easily be taken from a process, assigned to other processes and returned to the original process, enabling it to proceed normally. 2) False. Certain software resources are nonpreemptible, e.g., monitors. 7.5 Four Necessary Conditions for Deadlock Coffman, Elphick and Shoshani 7 proved that the following four conditions are necessary for deadlock to exist: 300 Deadlock and Indefinite Postponement 1. A resource may be acquired exclusively by only one process at a time (mutual exclusion condition). 2. A process that has acquired an exclusive resource may hold that resource while the process waits to obtain other resources (wait-for condition, also called the hold-and-wait condition). 3. Once a process has obtained a resource, the system cannot remove it from the process's control until the process has finished using the resource (nopreemption condition). 4. Two or more processes are locked in a "circular chain" in which each process is waiting for one or more resources that the next process in the chain. is holding (circular-wait condition). Because these are necessary conditions, the existence of a deadlock implies that each of them must be in effect. As we will see later, this observation helps us develop schemes to prevent deadlocks. Taken together, all four conditions are necessary and sufficient for deadlock to exist (i.e., if they are all in place, the system is deadlocked). Self Review 1. Describe how the four necessary conditions for deadlock apply to spooling systems. 2. Which of the four conditions would be violated if a user could remove jobs from a spooling system? Ans: 1) No two jobs can simultaneously write data to the same location in the spooling file. Partially spooled jobs remain in the spooling file until more space is available. Jobs cannot remove other jobs from the spooling file. Finally, when the spooling file is full, each job waits for all of the other jobs to free up space. 2) This would violate the "no-preemption" condition. 7.6 Deadlock Solutions Deadlock has been one of the more productive research areas in computer science and operating systems. There are four major areas of interest in deadlock research —deadlock prevention, deadlock avoidance, deadlock detection and deadlock recovery. In deadlock prevention our concern is to condition a system to remove any possibility of deadlocks occurring. Prevention is a clean solution as far as deadlock itself is concerned, but prevention methods can often result in poor resource utilization. In deadlock avoidance the goal is to impose less stringent conditions than in deadlock prevention in an attempt to get better resource utilization. Avoidance methods do not precondition the system to remove all possibility of deadlock. Instead, they allow the possibility to loom, but whenever a deadlock is approached, it is carefully sidestepped. Deadlock detection methods are used in systems in which deadlocks can occur. The goal is to determine if a deadlock has occurred, and to identify the processes and resources that are involved. 7.7 Deadlock Prevention Deadlock recovery methods are used to clear deadlocks from a system so that it may operate free them, and so that the deadlocked processes may complete their execution and free their resources. Recovery is a messy problem at best, typically requiring that one or more of the deadlocked processes be flushed from the system. The flushed processes are normally restarted from the beginning when sufficient resources are available, with much or all of the work done by these processes being lost. Self Review 1. Compare and contrast deadlock prevention and deadlock avoidance. 2. Some systems ignore the problem of deadlock. Discuss the costs and benefits of this approach. Ans: 1.) Deadlock prevention makes deadlock impossible but results in lower resource utilization. With deadlock avoidance, when the threat of deadlock approaches, it is sidestepped and resource utilization is higher. Systems using either deadlock prevention or deadlock avoidance will be free of deadlocks. 2) Systems that ignore deadlock may fail when deadlock occurs. This is an unacceptable risk in mission-critical systems, but it may be appropriate in other systems where deadlocks rarely occur and the "cost" of dealing with an occasional deadlock is lower than the costs of implementing deadlock prevention or avoidance schemes. 7.7 Deadlock Prevention This section considers various methods of deadlock prevention and examines the effects on both users and systems, especially from the standpoint of performance.8-9-1011, 12, 13 Havender,14 observing that a deadlock cannot occur if a system denies any of the four necessary conditions, suggested the following deadlock prevention strategies: • Each process must request all its required resources at once and cannot proceed until all have been granted. • If a process holding certain resources is denied a further request, it must release its original resources and, if necessary, request them again together with the additional resources. • A linear ordering of resources must be imposed on all processes; i.e., if a process has been allocated certain resources, it may subsequently request only those resources later in the ordering. In the sections that follow, we consider each strategy independently and discuss how each denies one of the necessary conditions (see the Anecdote, No Nuts, Bolts or Screws Allowed). Note that Havender presents three strategies, not four. The first necessary condition, namely that processes claim exclusive use of the resources they require, is not one that we want to break, because we specifically want to allow dedicated (i.e., serially reusable) resources. Self Review 1. What is the basic premise of Havender's research on deadlock prevention? Ans: Deadlock cannot occur in systems in which any of the necessary conditions for deadlock are precluded. 301 302 Deadlock and Indefinite Postponement 7.7.1 Denying the "Wait-For" Condition Havender's first strategy requires that all of the resources a process needs to complete its task must be requested at once. The system must grant them on an "all or none" basis. If all the resources needed by a process are available, then the system may grant them all to the process at once, and the process may continue to execute. If they are not all available, then the process must wait until they are. While the process waits, however, it may not hold any resources. Thus the "wait-for" condition is denied, and deadlocks cannot occur. Although this strategy successfully prevents deadlock, it wastes resources. For example, a program requiring four tape drives at one point in its execution must request, and receive, all four before it begins executing. If all four drives are needed throughout the execution of the program, then there is no serious waste. But, suppose the program needs only one tape drive to begin execution (or worse yet, none Anecdote No Nuts, Bolts or Screws Allowed Some years ago, HMD had the privilege of visiting one of the automated factories of one of the largest computer vendors. At the time, the factory—run almost exclusively by robots—was producing small inexpensive printers for personal computers. The head of the factory made a strong point. He observed that printers have a significant number of moving parts and that the company's engineers had traced failures of previous lines of printers to nuts, bolts and screws that had loosened as a consequence of the mechanical motion. So when they designed the printer that was being produced at this factory, the design team made a pact that all the parts of this printer (there were about 80 of them including the casing) would be designed to snap into place—no nuts, bolts or screws would be allowed in the design. The head of the factory then proceeded to assemble one of the printers by hand from the pieces sitting on his table. At the end of the demo, he plugged the printer into a wall socket and printed several pages to show that it was up and running. He mentioned that the "mean-timeto-failure" of this type of printer was proven to be miniscule cornpared to previous lines. Lesson to operating systems designers: One way to minimize the chance of system failure is to eliminate as many "points of failure" as possible at the design stage. This works well with another design heuristic—KIS (keep it simple). Both of these heuristics are appealing, but can be difficult to implement. As you read this book, keep this anecdote in mind, searching for points of failure that you may be able to eliminate in future operating systems designs. 7.7 at all) and then does not need the remaining tape drives for several hours. The requirement that the program must request, and receive, all four tape drives before execution begins means that substantial resources will sit idle for several hours. One approach to getting better resource utilization in these circumstances is to divide a program into several threads that run relatively independently of one another. Then resource allocation can be controlled by each thread rather than for the entire process. This can reduce waste, but it involves a greater overhead in application design and execution. This Havender strategy could cause indefinite postponement if, for example, it favored waiting processes with small resource needs over those attempting to accumulate many resources. A steady stream of processes arriving with small resource needs could indefinitely postpone a process with more substantial needs. One way to avoid this is to handle the needs of the waiting processes in first-come-firstserved order. Unfortunately, accumulating the full complement of resources for a process with substantial needs would cause considerable waste, as the gradually accumulating resources would sit idle until all were available. In large mainframe computer environments with expensive resources, there is some controversy over whom to charge for these unused resources. Because the resources are being accumulated for a specific user, some designers feel this user should pay for them, even while they sit idle. Other designers say this would destroy the predictability of resource charges; if the user tried to run the process on a busy day, the charges would be much higher than when the machine was lightly loaded. Self Review 1. Explain how denying the "wait-for" condition can reduce the degree of multiprogramming in a system. 2. Describe under what conditions a process can be indefinitely postponed when using Havender's scheme to deny the "wait-for" condition. Ans: 1) If processes in a system require more resources than are available to concurrently execute, the system must force some of those processes to wait until others finish and return their resources. In the worst case, the system may be limited to executing only one process at a time. 2) A process could request a substantial number of a system's resources. If the operating system gives higher priority to processes that require fewer of system resources, then the process requesting all of the system's resources will be postponed indefinitely while processes that require fewer resources proceed. 7.7.2 Denying the "No-Preemption" Condition Havender's second strategy denies the "no preemption" condition. Suppose a system does allow processes to hold resources while requesting additional resources. As long as sufficient resources remain available to satisfy all requests, the system cannot deadlock. But consider what happens when a request for additional resources cannot be satisfied. Now a process holds resources that a second process Deadlock Prevention 303 304 Deadlock and Indefinite Postponement may need in order to proceed, while the second process may hold resources needed by the first process—a two-process deadlock. Havender's second strategy requires that when a process holding resources is denied a request for additional resources, it must release the resources it holds and, if necessary, request them again together with the additional resources. This strategy effectively denies the "no-preemption" condition—resources can indeed be removed from the process holding them prior to the completion of that process. Here, too, the means for preventing deadlock can be costly. When a process releases resources, it may lose all of its work to that point. This may seem to be a high price to pay, but the real question is, "How often does this price have to be paid?" If this occurs infrequently, then this strategy provides a relatively low-cost means of preventing deadlocks. If it occurs frequently, however, then the cost is substantial and the effects are disruptive, particularly when high-priority or deadline processes cannot be completed on time because of repeated preemptions. Could this strategy lead to indefinite postponement? It depends. If the system favors processes with small resource requests over those requesting substantial resources, that alone could lead to indefinite postponement. Worse yet, as the process requests additional resources, this Havender strategy requires the process to give up all the resources it has and request an even larger number. So indefinite postponement can be a problem in a busy system. Also, this strategy requires all resources to be preemptible, which is not always the case (e.g., printers should not be preempted while processing a print job). Self Review 1. What is the primary cost of denying the "no-preemption" condition? 2. Which of Ha vender's first two deadlock strategies do you suppose people find more palatable? Why? Ans: 1) A process may lose all of its work up to the point that its resources were preempted. Also, the process could suffer indefinite postponement, depending on the system's resourceallocation strategy. 2) Most people probably would prefer the first strategy, namely, requiring a process to request all the resources it will need in advance. The second strategy requires the process to give up the resources it already has, possibly causing wasteful loss of work. Interestingly, the first strategy could cause waste as well, as processes gradually acquire resources they cannot yet use. 7.7.3 Denying the "Circular-Wait" Condition Havender's third strategy denies the possibility of a circular wait. In this strategy. we assign a unique number to each resource (e.g., a disk drive, printer, scanner, file) that the system manages and we create a linear ordering of resources. A process must then request its resources in a strictly ascending order. For example, if a process requests resource R3 (where the subscript, 3, is the resource number), then the process can subsequently request only resources with a number greater than 3. 7.7 Deadlock Prevention 305 Because all resources are uniquely numbered, and because processes must request resources in ascending order, a circular wait cannot develop (Fig. 7.5). [Note: A proof of of this property is straightforward. Consider a system that enforces a linear ordering of resources in which Ri and Rj are resources numbered by integers i and j, respectively (i ≠ j). If the system has a circular wait characteristic of deadlock, then according to Fig. 7.5, at least one arrow (or set of arrows) leads upward from Ri to Rj and an arrow (or set of arrows) exists leading down from Rj to Ri. However, the linear ordering of resources with the requirement that resources be requested in ascending order implies that no arrow can ever lead from Rj to Ri if j > i, Therefore deadlock cannot occur in this system.] Denying the "circular-wait" condition has been implemented in a number of legacy operating systems, but not without difficulties.15, 16, 17, 18 One disadvantage of this strategy is that it is not as flexible or dynamic as we might desire. Resources must be requested in ascending order by resource number. Resource numbers are assigned for the computer system and must be "lived with" for long periods (i.e., months or even years). If new resources are added or old ones removed at an instal lation, existing programs and systems may have to be rewritten. Another difficulty is determining the ordering of resources in a system. Clearly, the resource numbers should be assigned to reflect the order in which most processes actually use the resources. For processes matching this ordering, more efficient operation may be expected. But for processes that need the resources in a different order than that specified by the linear ordering, resources must be acquired and held, possibly for long periods of time, before they are actually used. This can result in poor performance. An important goal in today's operating systems is to promote software portability across multiple environments. Programmers should be able to develop their applications without being impeded by awkward hardware and software restrictions. Havender's linear ordering truly eliminates the possibility of a circular wait, yet it diminishes a programmer's ability to freely and easily write application code that will maximize an application's performance. Self Review 1. How does a linear ordering for resource allocation reduce application portability? 2. (T/F) Imposing a linear ordering for resource requests yields higher performance than denying the "no-preemption" condition. Ans: 1) Different systems will normally have different sets of resources and may order resources differently, so an application written for one system may need to be modified to run effectively on another. 2) False. There are situations where each solution results in higher performance because the solution requires insignificant overhead. If a system's processes each request disjoint sets of resources, denying "no preemption" is quite efficient. If a process uses resources in an order corresponding to the system's linear ordering, then denying the "circular wait" condition can result in higher performance. 306 Deadlock and Indefinite Postponement Process P1 has obtained resources R3, R4, R6 and R7, and is requesting resource R8 (as indicated by the dotted line). No circular wait can develop because all arrows must point upward. Figure 7.5 | Havender's linear ordering of of resources for preventing deadlock. 7.8 Deadlock Avoidance with Dijkstra's Banker's Algorithm For some systems, it is impractical to implement the deadlock prevention strategies we discussed in the preceding section. However, if the necessary conditions for a 7.8 Dealock Avoidance with Dijkstra's Banker's Algorithm 307 deadlock to occur are in place, it is still possible to avoid deadlock by carefully allo--ting system resources. Perhaps the most famous deadlock avoidance algorithm is Dijkstra's Banker's Algorithm, so named because the strategy is modeled after a banker who makes loans from a pool of capital and receives payments that are returned to that pool.19, 20, 21, 22, 23 We paraphrase the algorithm here in the context of operating systems resource allocation. Subsequently, much work has been done in deadlock avoidance.24, 25, 26, 27, 28, 29, 30, 31 The Banker's Algorithm defines how a particular system can prevent deadlock by carefully controlling how resources are distributed to users (see the Anecdote, Acronyms). A system groups all the resources it manages into resource types. Each resource type corresponds to resources that provide identical functionality. To simplify our presentation of the Banker's Algorithm, we limit our discussion to a system that manages only one type of resource. The algorithm is easily extendable to pools of resources of various types; this is left to the exercises. The Banker's Algorithm prevents deadlock in operating systems that exhibit the following properties: • The operating system shares a fixed number of resources, t, among a fixed number of processes, n. • Each process specifies in advance the maximum number of resources that it requires to complete its work. • The operating system accepts a process's request if that process's maximum need does not exceed the total number of resources available in the system, t (i.e., the process cannot request more than the total number of resources available in the system). • Sometimes, a process may have to wait to obtain an additional resource, but the operating system guarantees a finite wait time. • If the operating system is able to satisfy a process's maximum need for resources, then the process guarantees that the resource will be used and released to the operating system within a finite time. Anecdote Acronyms Most computer folks are familiar which nicely describe the relation- what I want when I want it." The with the acronym WYSIWYG for ship between the process request- operating system responds, " what you see is what you get." ing resources and the operating "You'll get what I've got when I Few people have seen these two - system attempting to allocate get it." IWWIWWIWI and YGWIGWIGI, them. The process says, "I want 308 Deadlock and Indefinite Postponement The system is said to be in a safe state if the operating system can guarantee that all current processes can complete their work within a finite time. If not, then the system is said to be in an unsafe state. We also define four terms that describe the distribution of resources among processes. • Let max(Pi) be the maximum number of resources that process Pi requires during its execution. For example, if process P3 never requires more than two resources, then max(P3) = 2. • Let loan(Pi) represent process P i 's current loan of a resource, where its loan is the number of resources the process has already obtained from the system. For example, if the system has allocated four resources to process P 5 , then loan(P5) = 4. • Let claim(Pi) be the current claim of a process, where a process's claim is equal to its maximum need minus its current loan. For example, if process P7 has a maximum need of six resources and a current loan of four resources, then we have claim(P7) = max(P7) - loan(P7) = 6 - 4 = 2 • Let a be the number of resources still available for allocation. This is equivalent to the total number of resources (t) minus the sum of the loans to all the processes in the system, i.e., So, if the system has a total of three processes and 12 resources, and the system has allocated two resources to process P 1 , one resource to process P 2 and four resources to process P3, then the number of available resources is a = 12 - (2 + 1 + 4) = 12 - 7 = 5 Dijkstra's Banker's Algorithm requires that resources be allocated to processes only when the allocations result in safe states. In that way, all processes remain in safe states at all times, and the system will never deadlock. Self Review 1. (T/F) An unsafe state is a deadlocked state. 2. Describe the restrictions that the Banker's Algorithm places on processes. Ans: 1) False. A process in an unsafe state might eventually deadlock, or it might complete its execution without entering deadlock. What makes the state unsafe is simply that the operating system cannot guarantee that from this state all processes can complete their work. From an unsafe state, it is possible but not guaranteed that all processes could complete their work, so a system in an unsafe state could eventually deadlock. 2) Each process, before it runs, is required to specify the maximum number of resources it may require at any point during its execution. Each process cannot request more than the total number of resources in 7.8 Deadlock Avoidance with Dijkstra's Banker's Algorithm 309 the system. Each process must also guarantee that once allocated a resource, the process will eventually return that resource to the system within a finite time. 7.8.1 Example of a Safe State Suppose a system contains 12 equivalent resources and three processes sharing the resources, as in Fig. 7.6. The second column contains the maximum need for each process, the third column the current loan for each process and the fourth column the current claim for each process. The number of available resources, a, is two; this is computed by subtracting the sum of the current claims from the total number of resources, t= 12. This state is "safe" because it is possible for all three processes to finish. Note that process P2 currently has a loan of four resources and will eventually need a maximum of six, or two additional resources. The system has 12 resources, of which 10 are currently in use and two are available. If the system allocates these two available resources to P 2 , fulfilling P 2 's maximum need, then P 2 can run to completion. Note that after P2 finishes, it will release six resources, enabling the system to immediately fulfill the maximum needs of P1 (3) and P3 (3), enabling both of those processes to finish. Indeed, all processes can eventually finish from the safe sate of Fig. 7.6. Self Review 1. In Fig. 7.6, once P2 runs to completion, must P1 and P3 run sequentially one after the other, or could they run concurrently? 2. Why did we focus initially on allowing P2 to finish rather than focusing on either P1 or P 3 ? Ans: 1) P 1 and P 3 each need three more resources to complete. When P 2 finishes, it will release six resources, which is enough to allow P1 and P3 to run concurrently. 2) Because P2 is the only process whose current claim can be satisfied by the two available resources. 7.8.2 Example of an Unsafe State Assume a system's 12 resources are allocated as in Fig. 7.7. We sum the values of the third column and subtract from 12 to obtain a value of one for a. At this point, no matter which process requests the available resource, we cannot guarantee that all three processes will finish. In fact, suppose process P1 requests and is granted the last available resource. A three-way deadlock could occur if indeed each process loan (Pi) (current loan) claim(Pi) (current claim) Process max(Pi) (maximum need) P1 4 1 3 P2 6 4 2 P3 8 5 3 Total resources, X, = 12 Figure 7.6 | Safe State. Available resources, a, = 2 310 Deadlock and Indefinite Postponement max(Pi) loan(Pi) (maximum need) (current Process claim (Pi) loan) (current claim) P1 10 8 2 p2 5 2 3 p3 3 1 2 Total resources, t, = 12 Available resources, a, = 1 Figure 7.7 | Unsafe state. needs to request at least one more resource before releasing any resources to the pool. It is important to note here that an unsafe state does not imply the existence of deadlock, nor even that deadlock will eventually occur. What an unsafe state does imply is simply that some unfortunate sequence of events might lead to a deadlock. 1. Why is deadlock possible, but not guaranteed, when a system enters an unsafe state? 2. What minimum number of resources would have to be added to the system of Fig. 7.7 to make the state safe? Ans: 1) Processes could give back their resources early, increasing the number of available resources to the point that the state of the system was once again safe and all other processes could finish. 2) By adding one resource, the number of available resources becomes two, enabling P1 to finish and return its resources, enabling both P2 and P3 to finish. Hence the new state is safe. 7.8.3 Example of Safe-State-to-Unsafe-State Transition Our resource-allocation policy must carefully consider all resource requests before granting them, or a process in a safe state could enter an unsafe state. For example, suppose the current state of a system is safe, as shown in Fig. 7.6. The current value of a is 2. Now suppose that process P3 requests an additional resource. If the system were to grant this request, then the new state would be as in Fig. 7.8. Now, the current value of a is 1, which is not enough to satisfy the current claim of any process, so the state is now unsafe. Process max(Pi) loan(Pi (maximum need) (current) claim(Pi loan) (current P1 4 1 3 P2 6 4 2 P3 8 6 2 Total resources, t, = 12 Available resources, a, = 1 Figure 7.8 | Safe-state-to-unsafe-state transition.) claim) 7.8 Deadlock Avoidance with Dijkstra's Banker's Algorithm 311 Self Review 1. If process P2 requested one additional resource in Fig. 7.6, would the system be in a safe or an unsafe state? 2. (TIF) A system cannot transition from an unsafe state to a safe state. Ans: 1) The system would still be in a safe state because there is a path of execution that will not result in deadlock. For example, if P2 requests another resource, then P2 can complete execution. Then there will be six available resources, which is enough for P 1 and P 3 to finish. 2) False. As processes release resources, the number of available resources can become sufficiently large for the state of the system to transition from unsafe to safe. 7.8.4 Banker's Algorithm Resource Allocation Now it should be clear how resource allocation operates under Dijkstra's Banker's Algorithm. The "mutual exclusion," "wait-for," and "no-preemption" conditions are allowed—processes are indeed allowed to hold resources while requesting and waiting for additional resources, and resources may not be preempted from a process holding those resources. As usual, processes claim exclusive use of the resources they require. Processes ease onto the system by requesting one resource at a time; the system may either grant or deny each request. If a request is denied, that process holds any allocated resources and waits for a finite time until that request is eventually granted. The system grants only requests that result in safe states. Resource requests that would result in unsafe states are repeatedly denied until they can eventually be satisfied. Because the system is always maintained in a safe state, sooner or later (i.e., in a finite time) all requests can be satisfied and all users can finish. Self Review 1. (T/F) The state described by Figure 7.9 is safe. 2. What minimum number of additional resources, when added to the system of Fig. 7.9, would make the state safe? Ans: 1) False. There is no guarantee that all of these processes will finish. P2 will be able to finish by using up the two remaining resources. However, once P2 is done, there are only three available resources left. This is not enough to satisfy either P 1 's claim of 4 or P 3 's claim of five. 2) By adding one more resource, we could allow P2 to finish and return three resources. These plus the added resource would enable P1 to finish, returning five resources and enabling P3 to finish. Process max(Pi) loan(Pi) claim(Pi P1 5 1 4 P2 P3 3 1 2 10 5 5 a=2 Figure 7.9 | State description of three processes.) 312 Deadlock and Indefinite Postponement 7.8.5 Weaknesses in the Banker's Algorithm The Banker's Algorithm is compelling because it allows processes to proceed that might have had to wait under a deadlock prevention situation. But the algorithm has a number of weaknesses. • It requires that there be a fixed number of resources to allocate. Because resources frequently require service, due to breakdowns or preventive maintenance, we cannot count on the number of resources remaining fixed. Similarly, operating systems that support hot swappable devices (e.g., USB devices) allow the number of resources to vary dynamically. • The algorithm requires that the population of processes remains fixed. This, too, is unreasonable. In today's interactive and multiprogrammed systems. the process population is constantly changing. • The algorithm requires that the banker (i.e., the system) grant all requests within a "finite time." Clearly, much better guarantees than this are needed in real systems, especially real-time systems. • Similarly, the algorithm requires that clients (i.e., processes) repay all loans (i.e., return all resources) within a "finite time." Again, much better guarantees than this are needed in real systems. • The algorithm requires that processes state their maximum needs in advance. With resource allocation becoming increasingly dynamic, it is becoming more difficult to know a process's maximum needs. Indeed, one main benefit of today's high-level programming languages and "friendly" graphical user interfaces is that users are not required to know such lowlevel details as resource use. The user or programmer expects the system to "print the file" or "send the message" and should not need to worry about what resources the system might need to employ to honor such requests. For the reasons stated above, Dijkstra's Banker's Algorithm is not implemented in today's operating systems. In fact, few systems can afford the overhead incurred by deadlock avoidance strategies. Self Review 1. Why does the Banker's Algorithm fail in systems that support hot swappable devices? Ans: The Banker's Algorithm requires that the number of resources of each type remain fixed. Hot swappable devices can be added and removed from the system at any time, meaning that the number of resources of each type can vary. 7.9 Deadlock Detection We have discussed deadlock prevention and avoidance —two strategies for ensuring that deadlocks do not occur in a system. Another strategy is to allow deadlocks to occur, locate them and remove them, if possible. Deadlock detection is the process of determining that a deadlock exists and identifying the processes and resources 7.9 Deadlock Detection 313 involved in the deadlock . 3 2 , 33, 34, 35, 36, 37, 38, 39, 40 Deadlock detection algorithms generally focus on determining if a circular wait exists, given that the other necessary conditions for deadlock are in place. Deadlock detection algorithms can incur significant runtime overhead. Thus we again face the trade-offs so prevalent in operating systems—is the overhead involved in deadlock detection justified by the potential savings from locating and breaking up deadlocks? For the moment, we shall ignore this issue and concentrate instead on the development of algorithms capable of detecting deadlocks. 7.9.1 Resource-Allocation Graphs To facilitate the detection of deadlocks, a popular notation (Fig. 7.10) is used in which a directed graph indicates resource allocations and requests.41 Squares represent pro----es and large circles represent classes of identical resources. Small circles drawn inside large circles indicate the separate identical resources of each class. For example, a large circle labeled "R 1 " containing three small circles indicates that there are three equivalent resources of type R1 available for allocation in this system. Figure 7.10 illustrates the relationships that may be indicated in a resourceallocation and request graph. In Fig. 7.10(a), process P1 is requesting a resource of type R1. The arrow from P1 touches only the extremity of the large circle, indicating that the resource request is under consideration. In Fig. 7.10(b), process P2 has been allocated a resource of type R2 (of which there are two).The arrow is drawn from the small circle within the large circle R2 to the square P 2 , to indicate that the system has allocated a specific resource of that type to the process. Figure 7.10(c) indicates a situation somewhat closer to a potential deadlock. Process P3 is requesting a resource of type R 3 , but the system has allocated the only R3 resource to process P 4 . Figure 7.10(d) indicates a deadlocked system in which process P5 is requesting a resource of type R 4 , the only one of which the system has allocated to process P6. Process P6, is requesting a resource of type R 5 , the only one of which the system has allocated to process P 5 . This is an example of the "circular wait" necessary for a deadlocked system. Resource-allocation and request graphs change as processes request resources, acquire them and eventually release them to the operating system. Self Review 1. Suppose a process has control of a resource of type R1. Does it matter which small circle points to the process in the resource-allocation graph? 2. What necessary condition for deadlock is easier to identify in a resource-allocation graph than it is to locate by analyzing the resource-allocation data of all the system's processes? Ans: 1) No; all resources of the same type must provide identical functionality, so it does not matter which small circle within the circle R1 points to the process. 2) Resource-allocation graphs make it easier to identify circular waits. 314 Deadlock and Indefinite Postponement Figure 7.10 | Resource-allocation and request graphs. 7.9.2 Reduction of Resource-Allocation Graphs One technique useful for detecting deadlocks involves graph reductions, in which the processes that may complete their execution, if any, and the processes that will remain deadlocked (and the resources involved in that deadlock), if any, are determined. 42 If a process's resource requests may be granted, then we say that a graph may be reduced by that process. This reduction is equivalent to showing how the graph would look if the process was allowed to complete its execution and return its resources to the system. We reduce a graph by a process by removing the arrows to that process from resources (i.e., resources allocated to that process) and by removing arrows from that process to resources (i.e., current resource requests of that process). If a graph can be reduced by all its processes, then there is no deadlock. If a graph cannot be reduced by all its processes, then the irreducible processes constitute the set of deadlocked processes in the graph. Figure 7.11 shows a series of graph reductions demonstrating that a particular set of processes is not deadlocked. Figure 7.10(d) shows an irreducible set of processes that constitutes a deadlocked system. It is important to note here that the order in 7.9 Deadlock Detection 315 Figure 7.11 | Graph reductions determining that no deadlock exists which the graph reductions are performed does not matter: The final result will always be the same. We leave the proof of this result to the exercises (see Exercise 7.29). 316 Deadlock and Indefinite Postponement Self Review 1. Why might deadlock detection be a better policy than either deadlock prevention or deadlock avoidance? Why might it be a worse policy? 2. Suppose a system attempts to reduce deadlock detection overhead by performing deadlock detection only when there are a large number of processes in the system. What is one drawback to this strategy? Ans: 1) In general, deadlock detection places fewer restrictions on resource allocation, thereby increasing resource utilization. However, it requires that the deadlock detection algorithm be performed regularly, which can incur significant overhead. 2) Because deadlock can occur between two processes, the system might not ever detect some deadlocks if the number of processes in the system is small. 7.10 Deadlock Recovery Once a system has become deadlocked, the deadlock must be broken by removing one or more of the four necessary conditions. Usually, several processes will lose some or all of the work they have accomplished. However, this may be a small price to pay compared with leaving a system in a state where it cannot use some of its resources. Recovery from deadlock is complicated by several factors. First, it may not be clear that the system has become deadlocked. For example, most systems contain processes that wake periodically, perform certain tasks, then go back to sleep. Because such processes do not terminate until the system is shut down, and because they rarely enter the active state, it is difficult to determine if they are deadlocked. Second, most systems do not provide the means to suspend a process indefinitely, remove it from the system and resume it (without loss of work) at a later time. Some processes, in fact, such as real-time processes that must function continuously, are simply not amenable to being suspended and resumed. Assuming that effective suspend/resume capabilities did exist, they would most certainly involve considerable overhead and might require the attention of a highly skilled system administrator. Such an administrator is not always available. Finally, recovery from deadlock is complicated because the deadlock could involve many processes (tens, or even hundreds). Given that recovering from deadlocks with even a small number of processes could require considerable work, dealing with deadlock among many hundreds (or even more) processes could be a monumental task. In current systems, recovery is ordinarily performed by forcibly removing a process from the system and reclaiming its resources.43, 44 The system ordinarily loses the work that the removed process has performed, but the remaining processes may now be able to complete. Sometimes it is necessary to remove several processes until sufficient resources have been reclaimed to allow the remaining processes to finish. Recovery somehow seems like an inappropriate term here, because some processes are in fact "killed" for the benefit of the others. Processes may be removed according to some priority order. Here, too, we face several difficulties. For example, the priorities of the deadlocked processes may 7.10 Deatlock Recovery 317 not exist, so the system may need to make an arbitrary decision. The priorities also may be incorrect or somewhat muddled by special considerations, such as deadline scheduling, in which a relatively low-priority process has a high priority temporarily because of an impending deadline. Furthermore, it may require considerable effort to determine the right processes to remove. The suspend/resume mechanism allows the system to put a temporary hold on a process (temporarily preempting its resources), and, when it is safe to do so, to resume the held process without loss of work. Research in this area is important for reasons other than deadlock recovery. For example, a suspend/resume mechanism can be applied to a system as a whole, allowing a user to shut down the entire system and start it later without loss of work. Such technology has been incorporated into many laptop systems, where limited battery life requires users to minimize power consumption. The Advanced Configuration and Power Interface (ACPI), a popular specification for power management, defines a sleeping state in which the contents of memory, registers and other system state information are written to a nonvolatile medium (such as the hard disk) and the system is powered off. In Windows XP, this feature is known as "suspend to disk" or "hibernate." When the system starts up again, it resumes from the point at which it entered the sleeping state without loss of work.45 Checkpoint/rollback, the precursor to suspend/resume, is widely used in current database systems. Checkpoint/rollback copes with system failures and deadlocks by attempting to preserve as much data as possible from each terminated process. Checkpoint/rollback facilitates suspend/resume capabilities by limiting the loss of work to the time at which the last checkpoint (i.e., saved state of the system) was taken. When a process in a system terminates (by accident or intentionally as the result of a deadlock recovery algorithm), the system performs a rollback by undoing every operation related to the terminated process that occurred since the last checkpoint. When databases may have many resources (perhaps millions or more) that must be accessed exclusively, there can be a risk of deadlock. To ensure that data in the database remains in a consistent state when deadlocked processes are terminated, database systems typically perform resource allocations using transactions. The changes specified by a transaction are made permanent only if the transaction completes successfully. We discuss transactions in more detail in Chapter 13, File and Database Systems. Deadlocks could have horrendous consequences in certain real-time systems. A real-time process control system monitoring a gasoline refinery must function without interruption for the refinery's safe and proper operation. A computerized heart pacemaker must literally not "miss a beat." Deadlocks cannot be risked in such environments. What would happen if a deadlock did develop? Clearly, it would have to be detected and removed instantaneously. But is this always possible? These are some of the considerations that keep operating systems designers from restful sleep. 318 Deadlock and Indefinite Postponement Self Review 1. (T/F) A system can eliminate deadlock by choosing at random an irreducible process in a resource-allocation graph. 2. Why might a system that restarts a process it "kills" to break a deadlock suffer from poor performance? Ans: 1) False. There may be multiple circular waits within the system by the time a deadlock is detected. 2) First, because killing a process causes loss of work. Second, because the restarted process will execute the same code that caused the initial deadlock, and if the system state has not changed, the system can become deadlocked repeatedly. 7.11 Deadlock Strategies in Current and Future Systems In personal computer systems and workstations, deadlock has generally been viewed as a limited annoyance. Some systems implement the basic deadlock prevention methods suggested by Havender while others ignore the problem—these methods seem to be satisfactory. While ignoring deadlocks may seem dangerous, this approach can actually be rather efficient. Consider that today's systems can contain thousands or millions of serially reusable objects on which processes and threads can deadlock. The time required to execute a deadlock detection algorithm can rise exponentially with the number of serially reusable objects in the system. If deadlock is rare, then the processor time devoted to checking for deadlocks significantly reduces system performance. In systems that are neither mission critical nor business critical, the choice of ignoring deadlock to favor performance often outweighs the need for a solution to the occasional deadlock. Although some systems ignore deadlock that occurs due to user processes, it is far more important to prevent deadlock in the operating system. Systems such as Microsoft Windows provide debugger support, allowing developers to thoroughly test drivers and applications to ensure that they acquire resources without causing deadlock (e.g., they do not attempt to acquire locks in recursive routines, or they specify lock acquisition in a certain order). 46 Interestingly, once such programs are released, these testing mechanisms are often disabled to improve efficiency.47 In real-time, mission-critical or business-critical systems the possibility of deadlock cannot be tolerated. Researchers have developed techniques that handle deadlock while minimizing data loss and maintaining good performance. For example, deadlock is commonly addressed in distributed database systems, which could provide concurrent access to billions of records for millions of users over thousands of sites.48 Due to the large size of distributed database systems, they often do not employ deadlock prevention or deadlock avoidance algorithms. Instead, they rely on deadlock detection and recovery via checkpoint/rollback (using transactions).49 These techniques are beyond the scope of this chapter; see Chapter 17, Introduction to Distributed Systems, for an introduction to such methods. Given current trends, deadlock will continue to be an important area of research for several reasons: Web Resources 319 • Many large-scale systems are oriented more toward asynchronous parallel operations than toward the serial operations of the past. Multiprocessing is common, and parallel computation will be prevalent. Networks and distributed systems are ubiquitous. There are, quite simply, more operations going on concurrently, more conflicts over resources and hence more chances of deadlock. Consequently, research on deadlock detection and recovery in distributed systems has become quite active. • With the increasing tendency of operating systems designers to view data as a resource, the number of resources that operating systems must manage is increasing dramatically. This is particularly evident in Web servers and database systems, which require high resource utilization and performance. This makes most deadlock prevention techniques impractical and deadlock recovery algorithms more important. Researchers have developed advanced transaction-based algorithms that ensure high resource utilization while maintaining a low cost of deadlock recovery.50 • Hundreds of millions of computers are incorporated into common devices, particularly small, portable ones such as cell phones, PDAs and navigation systems. These systems, increasingly characterized as Systems-on-a-Chip (SoC), are limited by a small set of resources and the demands of real-time tasks.51, 52 Deadlock-free resource allocation in such systems is essential, because users cannot rely on an administrator to detect and rid the system of the deadlock. Self Review 1. Why is deadlock prevention not a primary concern for many operating systems? 2. Why might deadlock in distributed systems be more difficult to detect than in a single computer? Ans: 1) Deadlock is rare and is often considered a minor annoyance for most personal computer users, who are often more concerned with operating system performance and features. 2) Deadlock in distributed systems can be difficult to detect because each computer is managed by a different operating system, requiring each operating system to collect information from other computers in order to construct its resource-allocation graph. Web Resources www.diee.unica.it/~giua/PAPERS/CONF/03smc_b.pdf Discusses an efficient approach to deadlock prevention in a real-time system. www.db.fmi.uni-passau.de/publications/techreports/ dda.html Describes a solution to deadlock detection in distributed systems. www.linux-mag.com/2001-03/compi1e_01.html This Compile Time article describes how to deal with deadlock, using the Dining Philosophers example. The beginning of the article also reviews mutual exclusion primitives (e.g., semaphores) and concurrent programming. 320 Deadlock and Indefinite Postponement Summary One problem that arises in multiprogrammed systems is deadlock. A process or thread is in a state of deadlock (or is deadlocked) if the process or thread is waiting for a particular event that will not occur. In a system deadlock, one or more processes are deadlocked. Most deadlocks develop because of the normal contention for dedicated resources (i.e., resources that may be used by only one user at a time). Circular wait is characteristic of deadlocked systems. One example of a system that is prone to deadlock is a spooling system. A common solution is to restrain the input spoolers so that, when the spooling files begin to reach some saturation threshold, they do not read in more print jobs. Today's systems allow printing to begin before the job is completed so that a full, or nearly full, spooling file can be emptied or partially cleared even while a job is still executing. This concept has been applied to streaming audio and video clips, where the audio and video begin to play before the clips are fully downloaded. In any system that keeps processes waiting while it makes resource-allocation and process scheduling decisions, it is possible to delay indefinitely the scheduling of a process while other processes receive the system's attention. This situation, variously called indefinite postponement, indefinite blocking, or starvation, can be as devastating as deadlock. Indefinite postponement may occur because of biases in a system's resource scheduling policies. Some systems prevent indefinite postponement by increasing a process's priority as it waits for a resource—this technique is called aging. Resources can be preemptible (e.g., processors and main memory), meaning that they can be removed from a process without loss of work, or nonpreemptible meaning that they (e.g., tape drives and optical scanners), cannot be removed from the processes to which they are assigned. Data and programs certainly are resources that the operating system must control and allocate. Code that cannot be changed while in use is said to be reentrant. Code that may be changed but is reinitialized each time it is used is said to be serially reusable. Reentrant code may be shared by several processes simultaneously, whereas serially reusable code may be used by only one process at a time. When we call particular resources shared, we must be careful to state whether they may be used by several processes simultaneously or by only one of several processes at a time. The latter kind—serially reusable resources —are the ones that tend to become involved in deadlocks. The four necessary conditions for deadlock are: a resource may be acquired exclusively by only one process at a time (mutual exclusion condition); a process that has acquired an exclusive resource may hold it while waiting to obtain other resources (wait-for condition, also called the hold-and-wait condition); once a process has obtained a resource, the system cannot remove the resource from the process's control until the process has finished using the resource (no-preemption condition); and two or more processes are locked in a "circular chain" in which each process in the chain is waiting for one or more resources that the next process in the chain is holding (circular-wait condition). Because these are necessary conditions for a deadlock to exist, the existence of a deadlock implies that each of them must be in effect. Taken together, all four conditions are necessary and sufficient for deadlock to exist (i.e., if all these conditions are in place, the system is deadlocked). The four major areas of interest in deadlock research are deadlock prevention, deadlock avoidance, deadlock detection, and deadlock recovery. In deadlock prevention our concern is to condition a system to remove any possibility of deadlocks occurring. Havender observed that a deadlock cannot occur if a system denies any of the four necessary conditions. The first necessary condition, namely that processes claim exclusive use of the resources they require, is not one that we want to break, because we specifically want to allow dedicated (i.e., serially reusable) resources. Denying the "wait-for" condition requires that all of the resources a process needs to complete its task be requested at once, which can result in substantial resource underutilization and raises concerns over how to charge for resources. Denying the "no-preemption" condition can be costly, because processes lose work when their resources are preempted. Denying the "circular-wait" condition uses a linear ordering of resources to prevent deadlock. This strategy can increase efficiency over the other strategies, but not without difficulties. In deadlock avoidance the goal is to impose less stringent conditions than in deadlock prevention in an attempt to get better resource utilization. Avoidance methods allow the possibility of deadlock to loom, but whenever a deadlock is approached, it is carefully sidestepped. Dijkstra's Banker's Algorithm is an example of a deadlock avoidance algorithm. In the Banker's Algorithm, the system ensures that a process's maximum resource need does not exceed the number of available resources. The system is said to be in a safe state if the operating system can guarantee that all current processes can complete their work within a finite time. If not, then the system is said to be in an unsafe state. Key Dijkstra's Banker's Algorithm requires that resources be allocated to processes only when the allocations result in safe states. It has a number of weaknesses (such as requiring a fixed number of processes and resources) that prevent it from being implemented in real systems. Deadlock detection methods are used in systems in which deadlocks can occur. The goal is to determine if a deadlock has occurred, and to identify those processes and resources involved in the deadlock. Deadlock detection algorithms can incur significant runtime overhead. To facilitate the detection of deadlocks, a directed graph indicates resource allocations and requests. Deadlock can be detected using graph reductions. If a process's resource requests may be granted, then we say that a graph may be reduced by that process. If a graph can be reduced by all its processes, then there is no deadlock. If a graph cannot be reduced by all its processes, then the irreducible processes constitute the set of deadlocked processes in the graph. Deadlock recovery methods are used to clear deadlocks from a system so that it may operate free of the deadlocks, and so that the deadlocked processes may complete their execution and free their resources. Recovery typically requires that one or more of the deadlocked processes be flushed from the system. The suspend/resume mechanism allows the system to put a temporary hold on a process (temporarily preempting its resources), and, when it is safe to do so, resume the held process without loss of work. Terms 321 Checkpoint/rollback facilitates suspend/resume capabilities by limiting the loss of work to the time at which the last checkpoint (i.e., saved state of the system) was taken. When a process in a system terminates (by accident or intentionally as the result of a deadlock recovery algorithm), the system performs a rollback by undoing every operation related to the terminated process that occurred since the last checkpoint. To ensure that data in the database remains in a consistent state when deadlocked processes are terminated, database systems typically perform resource allocations using transactions. In personal computer systems and workstations, deadlock has generally been viewed as a limited annoyance. Some systems implement the basic deadlock prevention methods suggested by Havender, while others ignore the problem—these methods seem to be satisfactory. While ignoring deadlocks may seem dangerous, this approach can actually be rather efficient. If deadlock is rare, then the processor time devoted to checking for deadlocks significantly reduces system performance. However, given current trends, deadlock will continue to be an important area of research as the number of concurrent operations and number of resources become large, increasing the likelihood of deadlock in multiprocessor and distributed systems. Also, many real-time systems, which are becoming increasingly prevalent, require deadlock-free resource allocation. Key Terms Advanced Configuration and Power Interface (ACPI)—Power management specification supported by many operating systems that allows a system to turn off some or all of its devices without loss of work. aging—Method of preventing indefinite postponement by increasing a process's priority gradually as it waits. checkpoint—Record of a state of a system so that it can be restored later if a process must be prematurely terminated (e.g., to perform deadlock recovery). checkpoint/rollback—Method of deadlock and system recovery that undoes every action (or transaction) of a terminated process since the process's last checkpoint. circular wait—Condition for deadlock that occurs when two or more processes are locked in a "circular chain," in which each process in the chain is waiting for one or more resources that the next process in the chain is holding. circular-wait necessary condition for deadlock—One of the four necessary conditions for deadlock; states that if a deadlock exists, there will be two or more processes in a circular chain such that each process is waiting for a resource held by the next process in the chain. deadline scheduling—Scheduling a process or thread to complete by a definite time; the priority of the process or thread may need to be increased as its completion deadline approaches. deadlock—Situation in which a process or thread is waiting for an event that will never occur. deadlock avoidance—Strategy that eliminates deadlock by allowing a system to approach deadlock, but ensuring that deadlock never occurs. Avoidance algorithms can achieve higher performance than deadlock prevention algorithms. (See also Dijktra's Banker's Algorithm.) deadlock detection—Process of determining whether or not a system is deadlocked. Once detected, a deadlock can be removed from a system, typically resulting in loss of work. deadlock prevention—Process of disallowing deadlock by eliminating one of the four necessary conditions for deadlock. 322 Deadlock and Indefinite Postponement deadlock recovery—Process of removing a deadlock from a system. This can involve suspending a process temporarily (and preserving its work) or sometimes killing a process (thereby losing its work) and restarting it. no-preemption necessary condition for deadlock—One of the four necessary conditions for deadlock; states that deadlock can occur only if resources cannot be forcibly removed from processes. deadly embrace—See deadlock. nonpreemptible resource—Resource that cannot be forcibly removed from a process, e.g., a tape drive. Such resources are the kind that can become involved in deadlock. dedicated resource—Resource that may be used by only one process at a time. Also known as a serially reusable resource. Dijkstra's Banker's Algorithm—Deadlock avoidance algorithm that controls resource allocation based on the amount of resources owned by the system, the amount of resources owned by each process and the maximum amount of resources that the process will request during execution. Allows resources to be assigned to processes only when the allocation results in a safe state. (See also safe state and unsafe state.) Dining Philosophers—Classic problem introduced by Dijkstra that illustrates the problems inherent in concurrent programming, including deadlock and indefinite postponement. The problem requires the programmer to ensure that a set of n philosophers at a table containing n forks, who alternate between eating and thinking, do not starve while attempting to acquire the two adjacent forks necessary to eat. graph reduction—Altering a resource-allocation graph by removing a process if that process can complete. This also involves removing any arrows leading to the process (from the resources allocated to the process) or away from the process (to resources the process is requesting). A resource-allocation graph can be reduced by a process if all of that process's resource requests can be granted, enabling that process to run to completion and free its resources. Havender's linear ordering—See linear ordering. linear ordering (Havender)—Logical arrangement of resources that requires that processes request resources in a linear order. This method denies the circular-wait necessary condition for deadlock. maximum need (Dijkstra's Banker's Algorithm) —Characteristic of a process in Dijkstra's Banker's Algorithm that describes the largest number of resources (of a particular type) the process will need during execution. mutual exclusion necessary condition for deadlock—One of the four necessary conditions for deadlock; states that deadlock can occur only if processes cannot claim exclusive use of their resources. necessary condition for deadlock—Condition that must be true for deadlock to occur. The four necessary conditions are the mutual exclusion condition, no-preemption condition, wait-for condition and circular-wait condition. preemptible resource—Resource that may be removed from a process such as a processor or memory. Such resources cannot be involved in deadlock. reentrant code—Code that cannot be changed while in use and therefore can be shared among processes and threads. resource allocation graph—Graph that shows processes and resources in a system. An arrow pointing from a process to a resource indicates that the process is requesting the resource. An arrow pointing from a resource to a process indicates that the resource is allocated to the process. Such a graph helps determine if a deadlock exists and. if so, helps identify the processes and resources involved in the deadlock. resource type —Grouping of resources that perform a common task. safe state—State of a system in Dijkstra's Banker's Algorithm in which there exists a sequence of actions that

will allow every process in the system to finish without the system becoming deadlocked. saturation threshold—A level of resource utilization above which the resource will refuse access. Designed to reduce deadlock, it also reduces throughput. serially reusable code—Code that can be modified but is reinitialized each time it is used. Such code can be used by only one process or thread at a time. serially reusable resource—See dedicated resource. shared resource—Resource that can be accessed by more than one process. starvation—Situation in which a thread waits for an event that might never occur, also called indefinite postponement. sufficient conditions for deadlock—The four conditionsmutual exclusion, no-preemption, wait-for and circularwait-which are necessary and sufficient for deadlock. suspend/resume—Method of halting a process, saving its state, releasing its resources to other processes, then restoring its resources after the other processes have released them. transaction—Atomic, mutually exclusive operation that either completes or is rolled back. Modifications to database entries are often performed as transactions to enable high performance and reduce the cost of deadlock recovery. Exercises 323 unsafe state—State of a system in Dijkstra's Banker's Algorithm that might eventually lead to deadlock because there might not be enough resources to allow any process to finish. wait-for condition—One of the four necessary conditions for deadlock; states that deadlock can occur only if a process is allowed to wait for a resource while it holds another. Exercises 7.1 Define deadlock. 7.2 Give an example of a deadlock involving only a single process and a single resource. 7.3 Give an example of a simple resource deadlock involving three processes and three resources. Draw the appropriate resource-allocation graph. 7.4 What is indefinite postponement? How does indefinite postponement differ from deadlock? How is indefinite postponement similar to deadlock? 7.5 Suppose a system allows for indefinite postponement of certain entities. How would you as a systems designer provide a means for preventing indefinite postponement? 7.6 Discuss the consequences of indefinite postponement in each of the following types of systems. a. batch processing b. timesharing c. real-time 7.7 A system requires that arriving processes must wait for service if the needed resource is busy. The system does not use aging to elevate the priorities of waiting processes to prevent 1 2 3 4 5 7.8 In a system of n processes, a subset of m of these processes is currently suffering indefinite postponement. Is it possible for the system to determine which processes are being indefinitely postponed? 7.9 (The Dining Philosophers) One of Dijkstra's more delightful contributions is his problem of the Dining Philosophers. 53, 54 It illustrates many of the subtle problems inherent in concurrent programming. Your goal is to devise a concurrent program (with a monitor) that simulates the behavior of the philosophers. Your program should be free of deadlock and indefinite postponement—otherwise one or more philosophers might starve. Your program must, of course, enforce mutual exclusion—two philosophers cannot use the same fork at once. Figure 7.12 shows the behavior of a typical philosopher. Comment on each of the following implementations of a typical philosopher. a. See Fig. 7.13. b. See Fig. 7.14. c. See Fig. 7.15. d. See Fig. 7.16. typical Philosopher() { while (true) { think(); 6 7 8 indefinite postponement. What other means might the system use to prevent indefinite postponement? eat(); } // end w h i l e } // end t y p i c a l Philosopher Figure 7.12 | Typical philosopher behavior for Exercise 7.9. 1 2 3 4 typicalPhilosopher() { while (true) { Figure 7.13 | Philosopher behavior for Exercise 7.9(a) (Part 1 of 2.) 324 Deadlock 5 and Indefinite Postponement think(); 6 7 8 9 10 11 12 13 14 15 16 pickUpLeftFork(); pickUpRightFork(); eat(); putDownLeftFork(); putDownRightFork(); } // end while } // end typical Philosopher Figure 7.13 | Philosopher behavior for Exercise 7.9(a). (Part 2 of 2.) 1 2 3 4 5 6 7 8 9 10 11 12 13 14 typicalPhilosopher() { while (true) { think(); pickUpBothForksAtOnce(); eat(); putDownBothForksAtOnce(); } // end while } // end typical Philosopher Figure 7.14 | Phjlosopher behavior for Exercise 7.9(b). 1 2 3 4 5 6 7 8 9 10 11 12 13 typicalPhilosopher() { while (true) { think(); w h i l e (notHoldingBothForks) { pickUpLeftFork(); if (r i g h t F o r k N o t A v a i l a b l e) { putDownLeftFork(); Figure 7.15 | Philosopher behavior for Exercise 7.9(c). (Part 1 of 2.) £Exercise 14 15 16 17 18 19 20 21 22 23 24 25 26 27 } / / end if else { pickUpRightFork(); } // end w h i l e } // end else eat(); putDownLeftFork() ; putDownRightFork(); } // end while } // end t y p i c ä l Philosopher Figure 7.15 | Philosopher behavior for Exercise 7.9(c). (Part 2 of 2.) 1 2 3 4 5 6 7 8 typicalPhilosopher() { while (true) { think(); if (philosopherID mod 2 == 0) { 9 10 11 pickUpLeftFork(); pickUpRightFork(); 12 13 eat(); 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 putDownLeftFork(); putDownRightFork(); } // end if else { pickUpRightFork(); pickUpLeftFork(); eat(); putDownRightFork(); putDownLeftFork(); } /,/ end else } // end while } // end typical Philosopher Figure 7.16 | Philosopher behavior for Exercise 7.9(d). 325 326 Deadlock and Indefinite Postponement 7.10 Define and discuss each of the following resource concepts. a. preemptible resource 7.19 Explain the intuitive appeal of deadlock avoidance over deadlock prevention. 7.20 In the context of Dijkstra's Banker's Algorithm discuss whether each of the states described in Fig. 7.17 and Fig. 7.18 is safe or unsafe. If a state is safe, show how it is possible for all processes to complete. If a state is unsafe, show how it is possible for deadlock to occur. b. nonpreemptible resource c. shared resource d. dedicated resource e. reentrant code f. serially reusable code g. dynamic resource allocation 7.11 State the four necessary conditions for a deadlock to exist. Give a brief intuitive argument for the necessity of each individual condition. 7.12 In the context of the traffic deadlock, illustrated in Fig. 7.1, discuss each of the necessary conditions for deadlock. 7.21 The fact that a state is unsafe does not necessarily imply that the system will deadlock. Explain why this is true. Give an example of an unsafe state and show how all of the processes could complete without a deadlock occurring. 7.22 Dijkstra's Banker's Algorithm has a number of weaknesses that preclude its effective use in real systems. Comment on why each of the following restrictions may be considered a weakness in the Banker's Algorithm. a. The number of resources to be allocated remains fixed. 7.13 What are the four areas of deadlock research mentioned in the text? Discuss each briefly. b. The population of processes remains fixed. 7.14 Havender's method for denying the "wait-for" condition requires that processes must request all of the resources they will need before the system may let them proceed. The system grants resources on an "all-or-none" basis. Discuss the pros and cons of this method. 7.15 Why is Havender's method for denying the "no-preemption" condition not a popular means for preventing deadlock? 7.16 Discuss the pros and cons of Havender's method for denying the "circular-wait" condition. 7.17 How does Havender's linear ordering for denying the "circular-wait" condition prevent cycles from developing in resource-allocation graphs? 7.18 A process repeatedly requests and releases resources of types R1 and R 2 , one at a time and in that order. There is exactly one resource of each type. A second process also requests and releases these resources one at a time repeatedly. Under what circumstances could these processes deadlock? If so, what could be done to prevent deadlock? Process max(Pi) loan(Pi c. The operating system guarantees that resource requests will be serviced in a finite time. d. Users guarantee that they will return held resources within a finite time. e. Users must state maximum resource needs in advance. 7.23 (Banker's Algorithm for Multiple Resource Types) Consider Dijkstra's Banker's Algorithm as discussed in Section 7.8, Deadlock Avoidance with Dijkstra's Banker's Algorithm. Suppose that a system using this deadlock avoidance scheme has n processes and m different resource types; assume that multiple resources of each type may exist and that the number of resources of each type is known. Develop a version of the Banker's Algorithm that will enable such a system to avoid deadlock. [Hint: Under what circumstances could a particular process be guaranteed to complete its execution, and thus return its resources to the pool?] 7.24 Could the Banker's Algorithm still function properly if resources could be requested in groups? Explain your answer carefully.) claim(Pi 4 1 3 P2 6 4 2 P3 8 5 3 P4 2 0 2 P1 a =1 Figure 7.17 | Resource description for State A.) Exercises Process max(Pi) loan(Pi) claim(Pi 8 4 4 p2 8 3 5 P3 8 5 3 P1 327) a=2 Figure 7.18 | Resource description for State B. 7.25 A system that uses Banker's-Algorithm deadlock avoidance has five processes (1, 2, 3, 4, and 5) and uses resources of four different types (A, B, C, and D). There are multiple resources of each type. Is the state of the system depicted by Fig. 7.19 and Fig. 7.20 safe? Explain your answer. If the system is safe, show how all the processes could complete their execution successfully. If the system is unsafe, show how deadlock might occur. 7.26 Suppose a system with n processes and m identical resources uses Banker's Algorithm deadlock avoidance. Write a function boolean i s S a f e S t a t e l (i n t [ ] [ ] maximumNeed, i n t [ ] [ ] loans. int[] a v a i l a b l e) that determines whether the system is in a safe state. 7.27 Suppose a system uses the Banker's Algorithm with n processes, m resource types, and multiple resources of each type. Write a function boolean i s S a f e S t a t e 2 (i n t [] [] maximumNeed, i n t [ ] [] l o a n s , i n t [ ] a v a i l a b l e) that determines whether the system is in a safe state. Process Current loan 7.28 In a system in which it is possible for a deadlock to occur, under what circumstances would you use a deadlock detection algorithm? 7.29 In the deadlock detection algorithm employing the technique of graph reductions, show that the order of the graph reductions does not matter, the same final state will result. [Hint: No matter what the order, after each reduction, the available resource pool increases.] 7.30 Why is deadlock recovery such a difficult problem? 7.31 Why is it difficult to choose which processes to "flush" in deadlock recovery? 7.32 One method of recovering from deadlock is to kill the lowest-priority process (or processes) involved in the deadlock. This (these) process(es) could then be restarted and once again allowed to compete for resources. What potential problem might develop in a system using such an algorithm? How would you solve this problem? Maximum Need Current Claim A B C D A B C D A B C C 1 1 0 2 0 3 2 4 2 2 2 2 2 2 0 3 1 2 3 5 1 2 3 2 0 0 3 2 4 5 1 2 7 7 5 0 3 2 4 5 5 0 8 2 5 0 2 1 4 2 0 0 1 4 3 0 0 6 5 4 2 1 3 6 2 Figure 7.19 | System state describing current loan, maximum need and current claim. Total Resources Resources Available A B C D A B C D 13 13 9 13 3 4 0 1 Figure 7.20 | System state describing total number of resources and available resources. 328 Deadlock and Indefinite Postponement 7.33 Why will deadlock probably be a more critical problem in future operating systems than it is today? 7.34 Do resource malfunctions that render a resource unusable generally increase or decrease the likelihood of deadlocks and indefinite postponement? Explain your answer. 7.35 The vast majority of computer systems in use today do allow at least some kinds of deadlock and indefinite postponement situations to develop, and many of these systems provide no automatic means of detecting and recovering from these problems. In fact, many designers believe that it is virtually impossible to certify a system as absolutely free of the possibilities of deadlock and indefinite postponement. Indicate how these observations should affect the design of "mission-critical" systems. 7.36 The table in Fig. 7.21 shows a system in an unsafe state. Explain how all of the processes may manage to finish execution without the system becoming deadlocked. 7.37 A system has three processes and four identical resources. Each process requires at most two of the resources at any given time. a. Can deadlock occur in this system? Explain. b. If there are m processes, and each could request up to n resources, how many resources must be available in the system to ensure that deadlock will never occur? c. If there are m processes and r resources in the system, what maximum number of resources, n, could each process request, if all processes must have the same maximum? Suggested Projects 7.40 Determine how Web servers and other business-critical systems address the problem of deadlock. 7.38 Prepare a research paper on how current operating systems deal with deadlock. 7.39 Research how real-time systems ensure that deadlock never occurs. How do they manage to eliminate deadlock yet maintain performance? Suggested Simulations 7.41 (Deadlock Detection and Recovery Project) Write a simulation program to determine whether a deadlock has occurred in a system of n identical resources and m processes. Have each process generate a set of resources that it wants (e.g., 3 of resource A, 1 of resource B and 5 of resource C). Then, from each set, request the resources one type at a time in a random order and with random pauses between types. Have each process hold onto all the resources that it has acquired until it can get all of them. Deadlocks should start developing. Now have another thread check for deadlocks every few seconds. It should report when a deadlock has occurred and start killing threads involved in the deadlock. Try different heuristics for choosing processes to kill and see which type of heuristic results in the best average time between deadlocks. 7.42 (Deadlock Prevention Simulation Project) Write a simulation program to compare various deadlock prevention schemes discussed in Section 7.7, Deadlock Prevention. In particular, compare deadlock prevention by denying the "wait-for" condition (Section 7.7.1, Denying the "Wait-For" Condition) with deadlock prevention by denying the "no-preemption" condition (Section 7.7.2, Denying the "No-Preemption" Condition). Your program should generate a sample user population, user arrival times, user resource needs (assume the system has n identical resources) in terms both of maximum needs and of when the resources are actually required, and so on. Each simulation should accumulate statistics on job turnaround times, resource utilization, number of jobs progressing at once (assume that jobs may progress when they have the portion of the n resources they currently need), and the like. Observe the results of your simulations and draw conclusions about the relative effectiveness of these deadlock prevention schemes. Process loan(Pi) max(Pi) claim(Pi) P1 1 5 4 P2 1 3 2 P3 5 10 5 a= 1 Figure 7.21 | Example of a system in an unsafe state. Recommended Reading 329 7.43 (Deadlock Avoidance Simulation Project) Write a simulation program to examine the performance of a system of n identical resources and m processes operating under banker'salgorithm resource allocation. Model your program after the one you developed in Exercise 7.42. Run your simulation and compare the results with those observed in your deadlock prevention simulations. Draw conclusions about the relative effectiveness of deadlock avoidance vs. the deadlock prevention schemes studied. 7.44 (Deadlock Prevention and Avoidance Comparison) Create a program that simulates arriving jobs with various resource needs (listing each resource that will be needed and the time at which it will be needed). This can be a randomnumber-based driver program. Use your simulation to determine how deadlock prevention and deadlock avoidance strategies yield higher resource utilization. Recommended Reading Though many operating systems simply ignore the problem of deadlock, there has been considerable research in the field to create effective algorithms to solve the problem. Dijkstra, Isloor and Marsland 55 and Zobel 56 characterize the problem of deadlock. Dijkstra was one of the first to document the problem of deadlock in multiprogrammed systems, presenting both the Dining Philosophers problem and the banker's algorithm for deadlock avoidance.57, 58 Coffman, Elphick and Shoshani later provided a framework for research in this area by classifying the four necessary conditions for deadlock.59 Holt 60, 61 developed resource-allocation graphs to aid in deadlock pre-ention, while Habermann 62 presented a family of deadlock avoidance techniques. In discussing the UNIX System V operating system, Bach indicates how deadlocks may develop in areas such as interrupt handling.63 Kenah et al. discuss the detection and handling of deadlocks in Digital Equipment Corporation's VAX/VMS operating system.64 Today's systems rarely implement deadlock prevention or avoidance mechanisms. In fact, in distributed systems, conventional deadlock prevention and avoidance mechanisms are not feasible, because each computer in a distributed system may be managed by a different operating system. Researchers have developed deadlock detection mechanisms65 coupled with checkpoint/rollback capabilities66 to reduce loss of work. Research into deadlock is currently focused on developing efficient deadlock detection algorithms and applying them to distributed systems, networks and other deadlock-prone environments.67 The bibliography for this chapter is located on our Web site at www.deitel.com/books/os3e/Bibliography.pdf. Works Cited 1. Isloor, S. S., and T. A. Marsland, "The Deadlock Problem: An Overview," Computer, Vol. 13, No. 9, September 1980, pp. 58-78. 2. Zobel, D., "The Deadlock Problem: A Classifying Bibliography," Operating Systems Review, Vol. 17, No. 4, October 1983, pp. 6-16. 3. Holt, R. G, "Some Deadlock Properties of Computer Systems," ACM Computing Surveys,Vol. 4, No. 3, September 1972, pp. 179196. 4. Coffman, E. G., Jr.; Elphick, M. J.; and A. Shoshani, "System Deadlocks," Computing Surveys,Vol. 3,No. 2, June 1971,p. 69. 5. Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, Netherlands, 1965, reprinted in F. Genuys, ed., Programming Languages, New York: Academic Press, 1968. 6. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Acta Informatica,Vol. 1,1971, pp. 115-138. 7. Coffman, E. G, Jr.; Elphick, M. J.; and A. Shoshani, "System Deadlocks," Computing Surveys, Vol. 3, No. 2, June 1971, pp. 6778. 8. Habermann, A. N., "Prevention of System Deadlocks," Communications of the ACM,Vol. 12, No. 7, July 1969, pp. 373-377,385. 9. Holt, R. C, "Comments on the Prevention of System Deadlocks," Communications of the ACM, Vol. 14, No. 1, January 1971, pp. 36-38. 10. Holt, R. C, "On Deadlock Prevention in Computer Systems," Ph.D. Thesis, Ithaca, NY.: Cornell University, 1971. 11. Parnas, D. L., and A. N. Haberman, "Comment on Deadlock Prevention Method," Communications of the ACM, Vol. 15, No. 9, September 1972, pp. 840-841. 12. Newton, G., "Deadlock Prevention, Detection, and Resolution: An Annotated Bibliography," ACM Operating Systems Review, Vol. 13, No. 2, April 1979, pp. 33-44. 13. Gelernter, D., "A DAG Based Algorithm for Prevention of Storeand-Forward Deadlock in Packet Networks," IEEE Transactions on Computers,Vol C-30,No. 10, October 1981, pp. 709-715. 14. Havender, J. W., "Avoiding Deadlock in Multitasking Systems," IBM Systems Journal, Vol. 7, No. 2,1968, pp. 74-84. 330 Deadlock and Indefinite Postponement 15. Brinch Hansen, P., Operating System Principles, Englewood Cliffs, NJ: Prentice Hall, 1973. 16. Scherr, A. L., "Functional Structure of IBM Virtual Storage Operating Systems, Part II: OS/VS2-2 Concepts and Philosophies," IBM Systems Journal, Vol. 12, No. 4,1973, pp. 382-400. 17. Auslander, M. A.; Larkin, D. C; and A. L. Scherr, "The Evolution of the MVS Operating System," IBM Journal of Research and Development, Vol. 25, No. 5,1981, pp. 471-482. 18. Kenah, L. J.; Goldenberg, R. E.; and S. F. Bate, VAX/VMS Internals and Data Structures, Version 4.4, Bedford, MA: Digital Press, 1988. 19. Brinch Hansen, P., Operating System Principles, Englewood Cliffs, NJ: Prentice Hall, 1973. 20. Dijkstra, E. W., Cooperating Sequential Processes, Technological University, Eindhoven, The Netherlands, 1965. 21. Dijkstra, E. W., Cooperating Sequential Processes, Technological University, Eindhover, Netherlands, 1965, Reprinted in F. Genuys, ed., Programming Languages, New York: Academic Press, 1968. 22. Habermann, A. N, "Prevention of System Deadlocks," Communications of the ACM, Vol. 12, No. 7, July 1969, pp. 373-377,385. 23. Madduri, H, and R. Finkel, "Extension of the Banker's Algorithm for Resource Allocation in a Distributed Operating System," Information Processing Letters, Vol. 19, No. 1, July 1984, pp. 1-8. 24. Havender, J. W., "Avoiding Deadlock in Multitasking Systems," IBM Systems Journal.Vol. 1, No. 2,1968, pp. 74-84. 25. Fontao, R. O., "A Concurrent Algorithm for Avoiding Deadlocks," Proc. Third ACM Symposium on Operating Systems Principles, October 1971, pp. 72-79. 26. Frailey, D. J., "A Practical Approach to Managing Resources and Avoiding Deadlock," Communications of the ACM, Vol. 16, No. 5, May 1973, pp. 323-329. 27. Devillers, R., "Game Interpretation of the Deadlock Avoidance Problem," Communications of the ACM, Vol. 20, No. 10, October 1977, pp. 741-745. 28. Lomet, D. B., "Subsystems of Processes with Deadlock Avoidance," IEEE Transactions on Software Engineering, Vol. SE-6, No. 3, May 1980, pp. 297-304. 29. Merlin, P. M., and P. J. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks—I: Store and Forward Deadlock," IEEE Transactions on Communications, Vol. COM-28, No. 3, March 1980, pp. 345-354. 30. Merlin, P. M., and P. J. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks—II: Other Deadlock Types," IEEE Transactions on Communications, Vol. COM-28, No. 3, March 1980, pp. 355-360. 31. Minoura, T, "Deadlock Avoidance Revisited," Journal of the ACM, Vol. 29, No. 4, October 1982, pp. 1023-1048. 32. Murphy, J. E., "Resource Allocation with Interlock Detection in a Multitask System," AFIPS FJCC Proc., Vol. 33, No. 2,1968, pp. 1169-1176. 33. Newton, G,, "Deadlock Prevention, Detection, and Resolution: An Annotated Bibliography," ACM Operating Systems Review, Vol. 13, No. 2, April 1979, pp. 33-44. 34. Gligor, V., and S. Shattuch, "On Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980, pp. 435-440. 35. Ho, G. S., and C. V. Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems," IEEE Transactions on Software Engineering, Vol. SE-8, No. 6, November 1982, pp. 554-557. 36. Obermarck, R., "Distributed Deadlock Detection Algorithm." ACM Transactions on Database Systems,Vol. 1, No. 2, June 1982. pp. 187-208. 37. Chandy, K. M., and J. Misra, "Distributed Deadlock Detection." ACM Transactions on Computer Systems,Vol. 1, No. 2, May 1983. pp. 144-156. 38. Jagannathan, J. R., and R. Vasudevan, "Comments on 'Protocols for Deadlock Detection in Distributed Database Systems'." IEEE Transactions on Software Engineering, Vol. SE-9, No. 3. May 1983, p. 371. 39. Kenah, L. I; Goldenberg, R. E.; and S. F. Bate, VAX/VMS Internals and Data Structures, version 4.4, Bedford, MA: Digital Press, 1988. 40. Pun H. Shiu, YuDong Tan, Vincent J. Mooney, "A Novel Parallel Deadlock Detection Algorithm and Architecture" Proceedings of the Ninth International Symposium on Hardware/Software Codesign, April 2001. 41. Holt, R. C, "Some Deadlock Properties of Computer Systems." ACM Computing Surveys, Vol. 4, No. 3, September 1972, pp. 179196. 42. Holt, R. C, "Some Deadlock Properties of Computer Systems." ACM Computing Surveys, Vol. 4, No. 3, September 1972, pp. 179196. 43. Kenah, L. J.; R. E. Goldenberg; and S. F. Bate, VAX/VMS Internals and Data Structures, version 4.4, Bedford, MA: Digital Press, 1988. 44. Thomas, K., Programming Locking Applications, IBM Corporation, 2001, . 45. Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. "Advanced Configuration and Power Management," rev. 2.0b, October 11,2002, p. 238 46. "Driver Development Tools: Windows DDK, Deadlock Detection," Microsoft MSDN Library, June 6, 2003. . 47. "Kernel-Mode Driver Architecture: Windows DDK, Preventing Errors and Deadlocks While Using Spin Locks," Microsoft MSDN Library, June 6, 2003, . 48. Krivokapic, N; Kemper, A.; and E. Gudes, "Deadlock Detection in Distributed Database Systems: A New Algorithm and Com- Works Cited parative Performance Analysis," The VLDB Journal— The International Journal on Very Large Data Bases, Vol. 8, No. 2,1999, pp. 79-100. 49. Krivokapic, N.; Kemper, A.; and E. Gudes, "Deadlock Detection in Distributed Database Systems: A New Algorithm and Comparative Performance Analysis," The VLDB Journal— The International Journal on Very Large Data Bases, Vol. 8, No. 2,1999, pp. 79-100. 50. Krivokapic, N.; Kemper, A.; and E. Gudes, "Deadlock Detection in Distributed Database Systems: A New Algorithm and Comparative Performance Analysis," The VLDB Journal— The International Journal on Very Large Data Bases, Vol. 8, No. 2,1999, pp. 79-100. 51. Magarshack, P., and P. Paulin, "System-on-chip Beyond the Nanometer Wall," Proceedings of the 40th Conference on Design Automation, Anaheim, CA: ACM Press, 2003, pp. 419-424. 52. Benini, L.; Macci, A.; and M. Poncino, "Energy-Aware Design of Embedded Memories: A Survey of Technologies, Architectures, and Techniques," ACM Transactions on Embedded Computer Systems (TECS), Vol. 2, No. 1,2003, pp. 5-32. 53. Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control," Communications of the ACM, Vol. 8, No. 5, September 1965, p. 569. 54. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Acta Informatica,Vol. 1,1971, pp. 115-138. 55. Isloor, S. S., and T. A. Marsland, "The Deadlock Problem: An Overview," Computer, Vol. 13, No. 9, September 1980, pp. 58-78. 56. Zobel, D, "The Deadlock Problem: A Classifying Bibliography," Operating Systems Review, Vol. 17, No. 4, October 1983, pp. 6-16. 57. Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, Netherlands, 1965, reprinted in E 331 Genuys, ed., Programming Languages, New York: Academic Press, 1968. 58. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Acta Informatica, Vol. 1,1971, pp. 115-138. 59. Coffman, E. G., Jr.; Elphick, M. J.; and A. Shoshani, "System Deadlocks," Computing Surveys, Vol. 3, No. 2, June 1971, pp. 6778. 60. Holt, R. C, "Comments on the Prevention of System Deadlocks," Communications of the ACM, Vol. 14, No. 1, January 1971, pp. 36-38. 61. Holt, R. G, "On Deadlock Prevention in Computer Systems," Ph.D. Thesis, Ithaca, NY.: Cornell University, 1971. 62. Habermann, A. N., "Prevention of System Deadlocks," Communications of the ACM, Vol. 12, No. 7, July 1969, pp. 373-377,385. 63. Bach, M. J., The Design of the UNIX Operating System, Englewood Cliffs, NJ: Prentice Hall, 1986. 64. Kenah, L. J.; Goldenberg, R. E.; and S. F. Bate, VAX/VMS Internals and Data Structures, version 4.4, Bedford, MA: Digital Press, 1988. 65. Obermarck, R., "Distributed Deadlock Detection Algorithm," ACM Transactions on Database Systems, Vol. 7, No. 2, June 1982, pp. 187-208. 66. Krivokapic, N.; Kemper, A.; and E. Gudes, "Deadlock Detection in Distributed Database Systems: A New Algorithm and Comparative Performance Analysis," The VLDB lournal— The International lournal on Very Large Data Bases, Vol. 8, No. 2,1999, pp. 79-100. 67. Shiu, P. H.; Tan, Y; and V J. Mooney, "A Novel Parallel Deadlock Detection Algorithm and Architecture," Proceedings of the Ninth International Symposium on Hardware/Software Codesign, April 2001. The heavens themselves, the planets and this center observe degree, priority, and place, ... —William Shakespeare — Nothing in progression can, rest on its original plan. We may as well think of rocking a grows man in the cradle of an infant. —Edmund Burke — For every problen there is one solution which is simple, neat, and wrong. —H. L. Mencken— There is nothing more requisite in business than dispatch. —Joseph Addison— Chapter 8 Processor Scheduling Objectives After reading this chapter, you should understand: • the goals of processor scheduling. • preemptive vs. nonpreemptive scheduling. • the role of priorities in scheduling. • scheduling criteria. • common scheduling algorithms. • the notions of deadline scheduling and real-time scheduling. • Java thread scheduling. Chapter Outline 8.1 Introduction 8.2 Scheduling levels 8.3 Preemptive vs. Nonpreemptpive Scheduling Operating Systems Thinking: Overhead 8.4 Priorities 8.5 Scheduling Objectives Operating Systems Thinking: Predictability 8.7.2 Round-Robin (RR) Scheduling 8.7.3 Shortest-Process-First (SPF) Scheduling 8.7.4 Highest-Response-Ratio-Next (HRRN) Scheduling 8.7.5 Shortest-Remaining-Time (SRT) Scheduling 8.7.6 Multilevel Feedback Queues 8.7.7 Fair Share Scheduling 8.8 Deadline Scheduling Operating Systems Thinking: Intensity of Resource Management vs. Relative Resource Value Operating Systems Thinking: Fairness 8.9 8.6 Scheduling Criteria Mini Case Study: Real-Time Operating Systems 8.7 Scheduling Algorithms 8.7.1 First-In-First-Out (FIFO) Scheduling 8.10 Java Thread Scheduling Real-Time Scheduling Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works C i t e d 334 8.1 Introduction 335 8.1 Introduction We have discussed how multiprogramming enables an operating system to use its resources more efficiently. When a system has a choice of processes to execute, it must have a strategy —called a processor scheduling policy (or discipline) —for deciding which process to run at a given time. A scheduling policy should attempt to satisfy certain performance criteria, such as maximizing the number of processes that complete per unit time (i.e., throughput), minimizing the time each process waits before executing (i.e., latency), preventing indefinite postponement of processes, ensuring that each process completes before its stated deadline, or maximizing processor utilization. Some of these goals, such as maximizing processor utilization and throughput, are complementary; others conflict with one another —a system that ensures that processes will complete before their deadlines may not achieve the highest throughput, fn this chapter, we discuss the problems of determining when processors should be assigned, and to which processes. Although we focus on processes, many of the topics we describe apply to jobs and threads as well. Self Review 1. When might a system that ensures that processes will complete before their deadlines not achieve the highest throughput? 2. Which performance criterion is the most important in an operating system? Why is this a hard question to answer? Ans: 1) This occurs, for example, when several short processes are delayed while the system dispatches a long process that must meets its deadline. 2) No one performance criterion is more important than the others for every operating system. It depends on the goals of the system. For example, in real-time systems, giving processes and threads immediate, predictable service is more important than high processor utilization. In supercomputers that perform lengthy calculations, processor utilization is typically more important than minimizing latency. 8.2 Scheduling levels In this section we consider three levels of scheduling (Fig. 8.1). High-level scheduling—also called job scheduling or long-term scheduling—determines which jobs the system allows to compete actively for system resources. This level is sometimes called admission scheduling, because it determines which jobs gain admission to the system. Once admitted, jobs are initiated and become processes or groups of processes. The high-level scheduling policy dictates the degree of multiprogramming— the total number of processes in a system at a given time. 1 Entry of too many processes into the system can saturate the system's resources, leading to poor performance. In this case, the high-level scheduling policy may decide to temporarily prohibit new jobs from entering until other jobs complete. After the high-level scheduling policy has admitted a job (which may contain one or more processes) to the system, the intermediate-level scheduling policy determines which processes shall be allowed to compete for processors. This policy 336 Processor Scheduling Figure 8.1 | Scheduling levels. responds to short-term fluctuations in system load. It temporarily suspends and resumes processes to achieve smooth system operation and to help realize certain systemwide performance goals. The intermediate-level scheduler acts as a buffer between the admission of jobs to the system and the assignment of processors to the processes representing these jobs. A system's low-level scheduling policy determines which active process the system will assign to a processor when one next becomes available. In many of today's systems, the low- and intermediate-level schedulers are the only schedulers. 8.3 Preemptive vs. Nonpreemtpive Scheduling (In this case, job initiation is performed by the intermediate-level scheduler.) Highlevel schedulers are often limited to large mainframe systems that perform batch processing. Low-level scheduling policies often assign a priority to each process, reflecting its importance —the more important a process, the more likely the scheduling policy is to select it to execute next. We discuss priorities in Section 8.4, Priorities, and throughout this chapter. The low-level scheduler (also called the dispatcher) also assigns (i.e., dispatches) a processor to the selected process. The dispatcher operates many times per second and must therefore reside in main memory at all times. In this chapter we discuss many low-level scheduling policies. We present each in the context of certain scheduling objectives and criteria (which we discuss in Section 8.5, Scheduling Objectives, and Section 8.6, Scheduling Criteria), and we describe how they relate to one another. Coffman and Kleinrock discuss popular scheduling policies and indicate how users who know which one the system employs can actually achieve better performance by taking appropriate measures. 2 Ruschitzka and Fabry give a classification of scheduling algorithms, and they formalize the notion of priority.3 Self Review 1. How should the intermediate scheduler respond to fluctuations in system load? 2. Which level of scheduler should remain resident in main memory? Why? Ans: 1) The intermediate scheduler can prohibit processes from proceeding to the low-level scheduler when the system becomes overloaded and can allow these processes to proceed when the system load returns to normal. 2) The low-level scheduler should remain resident in main memory because it executes frequently, requiring it to respond quickly to reduce scheduling overhead. 8.3 Preemptive vs. Nonpreemtpive Scheduling Scheduling disciplines are either preemptive or nonpreemptive. A scheduling discipline is nonpreemptive if, once the system has assigned a processor to a process, the system cannot remove that processor from that process. A scheduling discipline is preemptive if the system can remove the processor from the process it is running. Under a nonpreemptive scheduling discipline, each process, once given a processor, runs to completion or until it voluntarily relinquishes its processor. Under a preemptive scheduling discipline, the processor may execute a portion of a process's code and then perform a context switch. Preemptive scheduling is useful in systems in which high-priority processes require rapid response. In real-time systems (discussed in Section 8.9, Real-Time Scheduling), for example, the consequences of not responding to an interrupt could be catastrophic. 4, 5, 6, 7 In interactive timesharing systems, preemptive scheduling helps guarantee acceptable user response times. Preemption is not without c o s t context switches incur overhead (see the Operating Systems Thinking feature, 337 338 Processor Scheduling Overhead). To make preemption effective, the system must maintain many processes in main memory, so that the next process is ready when a processor becomes available. As we will see in Chapter 10, Virtual Memory Organization, only a portion of each process typically is in main memory at any time; the less active portions typically reside on disk. In nonpreemptive systems, short processes can experience lengthy service delays while longer processes complete, but turnaround times are more predictable, because incoming high-priority processes cannot displace waiting processes. Because a nonpreemptive system cannot remove a process from a processor until it completes, errant programs that never complete (e.g., by entering an infinite loop) may never relinquish control of the system. Also, in a nonpreemptive system, executing unimportant processes can make important processes wait. To prevent users from monopolizing the system (either maliciously or accidentally), a preemptive system can take the processor away from a process. As discussed in Chapter 3, Process Concepts, this is typically implemented by setting an interrupting clock or interval timer that periodically generates an interrupt, which allows the operating system to execute. Once a processor is assigned to a process, the process executes until it voluntarily releases its processor, or until the clock interrupt or some other interrupt occurs. The operating system may then decide whether the running process should continue or some other "next" process should execute. The interrupting clock helps guarantee reasonable response times to interactive users, prevents the system from getting hung up on a user in an infinite loop. Operating Systems Thinking Overhead Ultimately, computer systems exist to run applications for users. Although operating systems certainly perform important tasks, they consume valuable system resources in doing so; that resource consumption is said to be overhead, because the resources are not directly used by user applications to perform useful work. Operating systems designers seek to minimize that over- head while maximizing the portion of the systems resources that can be allocated to user applications. As we will see, overhead can improve performance by improving resource utilization; it can also reduce performance to provide a higher level of protection and security. As computer power continues to increase with costs decreasing, the resource consumption lost to overhead can become less of an issue. However, designers should be aware of a system's workload when considering overhead. For example, a large overhead for a small load may become relatively small under a heavy load; a small overhead for a small load may become relatively large under a heavy load. 8.4 and allows processes to respond to time-dependent events. Processes that need to run periodically depend on the interrupting clock. In designing a preemptive scheduling mechanism, one must carefully consider the arbitrariness of priority schemes. It does not make sense to build a sophisticated mechanism to faithfully implement a priority preemption scheme when the priorities themselves are not meaningfully assigned. Self R e v i e w 1. When is nonpreemptive scheduling more appropriate than preemptive scheduling? 2. Can a program that enters an infinite loop monopolize a preemptive system? Ans: 1) Nonpreemptive scheduling provides predictable turnaround times, which is important for batch processing systems that must provide users with accurate job-completion times. 2) This depends on the priority of the process and the scheduling policy. In general, a preemptive system containing a process executing an infinite loop will experience reduced throughput but will still be able to execute other processes periodically. A high-priority process that enters an infinite loop, however, may execute indefinitely if all other processes in the system have a lower priority. In general, preemptive systems are less affected by such programs than nonpreemptive systems. Operating systems typically deal with such situations by limiting the maximum time a process can use a processor. 8.4 Priorities Schedulers often use priorities to determine how to schedule and dispatch processes. Priorities may be statically assigned or may change dynamically. Priorities quantify the relative importance of processes. Static priorities remain fixed, so static-priority-based mechanisms are relatively easy to implement and incur relatively low overhead. Such mechanisms are not, however, responsive to changes in environment, even those that could increase throughput and reduce latency. Dynamic priority mechanisms are responsive to change. For example, the system may want to increase the priority of a process that holds a key resource needed by a higher-priority process. After the first process relinquishes the resource, the system lowers the priority, so that the higher-priority process may execute. Dynamic priority schemes are more complex to implement and have greater overhead than static schemes. Hopefully, the overhead is justified by the increased responsiveness of the system. In multiuser systems, an operating system must provide reasonable service to a large community of users but should also provide for situations in which a member of the user community needs special treatment. A user with an important job may be willing to pay a premium, i.e., to purchase priority, for a higher level of service. This extra charge is merited because resources may need to be withdrawn from other paying customers. If there were no extra charge, then all users would request the higher level of service. Priorities 339 340 Processor Scheduling Self Review 1. Why is it worthwhile to incur the higher cost and increased overhead of a dynamic priority mechanism? 2. Why would a dynamic scheduler choose to favor a low-priority process requesting an underutilized resource? Ans: 1) A carefully designed dynamic priority mechanism could yield a more responsive system than would a static priority mechanism. 2) The underutilized resource is likely to be available, allowing the low-priority process to complete and exit the system sooner than a higher-priority process waiting for a saturated resource. 8.5 Scheduling Objectives A system designer must consider a variety of factors when developing a scheduling discipline, such as the type of system and the users' needs. For example, the scheduling discipline for a real-time system should differ from that for an interactive desktop system; users expect different results from these kinds of systems. Depending on the system, the user and designer might expect the scheduler to: • Maximize throughput. A scheduling discipline should attempt to service the maximum number of processes per unit time. • Maximize the number of interactive processes receiving "acceptable" response times. • Maximize resource utilization. The scheduling mechanisms should keep the resources of the system busy. • Avoid indefinite postponement. A process should not experience an unbounded wait time before or while receiving service. • Enforce priorities. If the system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes. • Minimize overhead. Interestingly, this is not generally considered to be one of the most important objectives. Overhead often results in wasted resources. But a certain portion of system resources effectively invested as overhead can greatly improve overall system performance. • Ensure predictability. By minimizing the statistical variance in process response times, a system can guarantee that processes will receive predictable service levels (see the Operating Systems Thinking feature, Predictability). A system can accomplish these goals in several ways. In some cases, the scheduler can prevent indefinite postponement of processes through aging—gradually increasing a process's priority as the process waits for service. Eventually, its priority becomes high enough that the scheduler selects that process to run. The scheduler can increase throughput by favoring processes whose requests can be satisfied quickly, or whose completion frees other processes to run. One such strategy favors processes holding key resources. For example, a low-priority process may be holding a key resource that is required by a higher-priority process. If the 8.5 Scheduling Objectives 341 resource is nonpreemptible, then the scheduler should give the low-priority process more execution time than it ordinarily would receive, so that it will release the key resource sooner. This technique is called priority inversion, because the relative pri-----es of the two processes are reversed so that the high-priority one will obtain the resource it requires to continue execution. Similarly, the scheduler may choose to favor a process that requests underutilized resources, because the system will be more likely to satisfy this process's requests in a shorter period of time. Many of these goals conflict with one another, making scheduling a complex problem. For example, the best way to minimize response times is to have sufficient resources available whenever they are needed. The price for this strategy is that overall resource utilization will be poor. In real-time systems, fast, predictable responses are crucial, and resource utilization is less important. In other types of systems, the economics often makes effective resource utilization imperative. Despite the differences in goals among systems, many scheduling disciplines exhibit similar properties: • Fairness. A scheduling discipline is fair if all similar processes are treated the same, and no process can suffer indefinite postponement due to scheduling issues (see the Operating Systems Thinking feature, Fairness). • Predictability. A given process always should run in about the same amount of time under similar system loads. • Scalability. System performance should degrade gracefully (i.e., it should not immediately collapse) under heavy loads. Operating Systems Thinking Predictability Predictability is as important in your everyday life as it is in computers. What would you do if you stopped for a red traffic light, and the light stayed red for a long time, much longer than any red light delay you had ever experienced. You would probably become annoyed and fidgety, and after waiting for what you thought was a reasonable amount of time; you might be inclined to go through the light, a potentially dangerous action. Similarly, you have a sense of how long common tasks should take to perform on your computer. It is challenging for operating systems to ensure predictability, espedaily given that the load on a system can vary considerably based on the system load and the nature of the tasks being performed. Predictability is especially important for interactive users who demand prompt, consistent response times. Predictability is also important for real-time jobs, where human lives may be at stake (we discuss realtime scheduling in Section 8.9, Real-Time Scheduling). We discuss predictability issues throughout the book. 342 Processor Scheduling Self Review 1. How do the goals of reducing the variance in response times and enforcing priorities conflict? 2. Is scheduling overhead always "wasteful"? Ans: 1) In preemptive systems, high-priority processes can preempt lower-priority ones ai any time, thereby increasing the variance in response times. 2) No, the overhead incurred bj effective scheduling operations can increase resource utilization. 8.6 Scheduling Criteria To realize a system's scheduling objectives, the scheduler should consider process behavior. A processor-bound process tends to use all the processor time that the system allocates for it. An I/O-bound process tends to use the processor only briefly before generating an I/O request and relinquishing it. Processor-bound processes spend most of their time using the processor; I/O-bound processes spend most of their time waiting for external resources (e.g., printers, disk drives, network connections, etc.) to service their requests, and only nominal time using processors. A scheduling discipline might also consider whether a process is batch or interactive. A batch process contains work for the system to perform without interacting with the user. An interactive process requires frequent inputs from the user. The system should provide good response times to an interactive process, whereas a batch process generally can suffer reasonable delays. Similarly, a scheduling discipline should be sensitive to the urgency of a process. An overnight batch process Operating Systems Thinking Fairness How many times have you expressed annoyance saying, "That's not fair!" You have probably heard the expression, "Life is not fair." We all know what fairness is and most people would agree that it is a "good t h i n g " but not universally achieved. We discuss fairness issues in many chapters throughout this book. We will see priority-based scheduling strategies that yield good performance for most users, processes, threads, I/O requests and the like, but do so at the expense of other users, processes, threads and I/O requests that wind up being mistreated. "That's not fair," you say, yet it is hard to ignore how these strategies achieve the system's objectives, such as improving overall system performance. Operating systems must keep fairness in mind, but always in the context of other considerations. Fairness issues are particularly important in designing resource scheduling strategies, as we will see when we investigate processor scheduling in this chapter and disk scheduling in Chapter 12, Disk Performance Optimization. 8.6 does not require immediate responses. A real-time process control system that monitors a gasoline refinery must be responsive, possibly to prevent an explosion. When the second edition of this book was published, users interacted with processes by issuing trivial requests using a keyboard. In this environment, a scheduler could favor an interactive process with little effect on other processes, because the time required to service interactive processes (e.g., by displaying text) was nominal. AS computers became more powerful, system designers and application programmers included features such as graphics and GUIs to improve userfriendliness. Although some systems still use text-based interfaces, most of today's users interact via GUIs, using a mouse to perform actions such as opening, resizing, dragging and closing windows. Users expect systems to respond quickly so that these actions produce smooth movement. Unlike text display, this can be a computationally intensive task requiring the system to redraw the screen many times per second. Favoring these interactive processes can significantly reduce the level of service provided to other processes in the system. In the case of a batch process, this temporary reduction in service may be acceptable, though perhaps not for processes that execute in real time (e.g., multimedia applications). In a system that employs priorities, the scheduler should favor processes with higher priorities. Schedulers can base their decisions on how frequently a higher priority process has preempted a lower-priority one. Under some disciplines, frequently preempted processes receive less favored treatment. This is because the process's short runtime before preemption does not justify the overhead incurred by a context switch each time the process is dispatched. One can argue to the contrary that such processes should receive more favored treatment to make up for previous "mistreatment." Preemptive scheduling policies often maintain information about how much real execution time each process has received. Some designers believe a process that has received little execution time should be favored. Others believe a process that has received much execution time could be near completion and should be favored to help it reach completion, free its resources for other processes to use and exit the system as soon as possible. Similarly, a scheduler may maintain an estimate of how much time remains before a process completes. It is easy to prove that average waiting times can be minimized by running those processes first that require the minimum runtime until completion. Unfortunately, a system rarely knows exactly how much more time each process needs to complete. Self Review 1. Are interactive processes generally processor bound or I/O bound? How about batch processes? 2. The scheduler rarely knows exactly how much more time each process needs to complete. Consider a system that schedules processes based on this estimation. How can processes abuse this policy? Scheduling Criteria 343 344 Processor Scheduling Ans: 1) Interactive processes often wait for input from users, so they are generally I/O bound, but an interactive process could certainly enter a phase during which it is primarily processor bound. Batch processes do not interact with users and are often processor bound. Batch processes that require frequent access to disk or other I/O devices are I/O bound. 2) Processes would tend to underestimate their times to completion to receive favored treatment from the scheduler. 8.7 Scheduling Algorithms In previous sections we discussed scheduling policies, which specify the goal of the scheduler (e.g., maximizing throughput or enforcing priorities). In the subsections that follow we discuss scheduling algorithms that determine at runtime which process runs next. These algorithms decide when and for how long each process runs; they make choices about preemptibility, priorities, running time, time-to-completion, fairness and other process characteristics. As we will see, some systems require the use of a particular type of scheduler (e.g., real-time systems typically require preemptive, priority-based schedulers). Others rely on process behavior when making scheduler decisions (e.g., favoring I/O-bound processes). 8.7.1 First-In-First-Out (FIFO) Scheduling Perhaps the simplest scheduling algorithm is first-in-first-out (FIFO), also called first-come-first-served (FCFS) (Fig. 8.2). Processes are dispatched according to their arrival time at the ready queue. FIFO is nonpreemptive — once a process has a processor, the process runs to completion. FIFO is fair in that it schedules processes according to their arrival times, so all processes are treated equally, but somewhat unfair because long processes make short processes wait, and unimportant processes make important processes wait. FIFO is not useful in scheduling interactive processes because it cannot guarantee short response times. FIFO is rarely used as a master scheme in today's systems, but it is often found within other schemes. For example, many scheduling schemes dispatch processes according to priority, but processes with the same priority are dispatched in FIFO order. Figure 8.2 | First-in-first out scheduling. 8.7 Scheduling Algorithms 345 Self Review 1. Can indefinite postponement occur in a system that uses a FIFO scheduler? Assume that all processes eventually run to completion (i.e., no process enters an infinite loop). 2. (T/F) FIFO scheduling is rarely found in today's systems. Ans: 1) No, indefinite postponement cannot occur, because arriving processes must enter at the back of the queue, meaning they cannot prevent already waiting processes from executing. 2) False. FIFO can be found within many of today's scheduling algorithms (e.g., prioritybased schedulers that dispatch processes with the same priority in FIFO order). 8.7.2 Round-Robin (RR) Scheduling In round-robin (RR) scheduling (Fig. 8.3), processes are dispatched FIFO but are given a limited amount of processor time called a time slice or a quantum.8 If a process does not complete before its quantum expires, the system preempts it and gives the processor to the next waiting process. The system then places the preempted process at the back of the ready queue. In Fig. 8.3, process P1 is dispatched to a processor, where it executes either until completion, in which case it exits the system, or until its time slice expires, at which point it is preempted and placed at the tail of the ready queue. The scheduler then dispatches process P 2 . Round-robin is effective for interactive environments in which the system needs to guarantee reasonable response times. The system can minimize preemption overhead through efficient context-switching mechanisms and by keeping waiting processes in main memory. Like FIFO, round-robin is commonly found within more sophisticated processor scheduling algorithms but is rarely the master scheme. As we will see throughout this section, many more sophisticated scheduling algorithms degenerate to either FIFO or round-robin when all processes have the same priority. For this reason, FIFO and round-robin are two of the three scheduling algorithms required by the POSIX specification for real-time systems (we discuss real-time scheduling in Section 8.9, Real-Time Scheduling).9 Figure 8.3 | Round-robin Scheduling. 346 Processor Scheduling Selfish Round-Robin Kleinrock discussed a variant of round-robin called selfish round-robin (SRR) that uses aging to gradually increase process priorities over time.10 In this scheme, as each process enters the system, it first resides in a holding queue, where it ages until the its priority reaches the level of processes in the active queue. At this point, it is placed in the active queue and scheduled round-robin with other processes in the queue. The scheduler dispatches only processes in the active queue, meaning that older processes are favored over those that have just entered the system. In SRR. a process's priority increases at a rate a while in the holding queue, and at a rate b, where b ≤ a , in the active queue. When b < a, processes in the hold ing queue age at a higher rate than those in the active queue, so they will eventually enter the active queue and contend for the processor. Tuning the parameters a and b impacts how a process's age affects average latency and throughput. For example, as a becomes much larger than b, then processes that enter the system will spend lit tle, if any, time in the holding queue. If b c? 2. The text describes an optimal quantum value that enables each I/O-bound process to execute just long enough to generate an I/O request, then block. Why is this difficult to implement? 348 Processor Scheduling Ans: 1) This point would be the longest period of computation between I/O requests for any processes in the system. Increasing the value of q past c does not affect any processes. because each process blocks before its quantum expires, so the processes cannot take advantage of the additional processor time allocated to them. 2) In the general case, it is impossible to predict the path of execution a program will take, meaning that the system cannot accurately determine when a process will generate I/O. Therefore, the optimal quantum size is difficult to determine, because it is different for each process and can vary over time. 8.7.3 Shortest-Process-First (SPF) Scheduling Shortest-process-first (SPF) is a nonpreemptive scheduling discipline in which the scheduler selects the waiting process with the smallest estimated run-time-to-completion. SPF reduces average waiting time over FIFO. 14 The waiting times, however, have a larger variance (i.e., are more unpredictable) than FIFO, especially for large processes. SPF favors short processes at the expense of longer ones. Many designers advocate that the shorter the process, the better the service it should receive. Other designers disagree, because this strategy does not incorporate priorities (as measured by the importance of a process). Interactive processes, in particular, tend to be "shorter" than processor-bound processes, so this discipline would seem to still provide good interactive response times. The problem is that it is nonpreemptive, so, in general, arriving interactive processes will not receive prompt service. SPF selects processes for service in a manner ensuring the next one will complete and leave the system as soon as possible. This tends to reduce the number of waiting processes and also the number of processes waiting behind large processes. As a result, SPF can minimize the average waiting time of processes as they pass through the system. A key problem with SPF is that it requires precise knowledge of how long a process will run, and this information usually is not available. Therefore, SPF must rely on user- or system-supplied run-time estimates. In production environments where the same processes run regularly, the system may be able to maintain reasonable runtime heuristics. In development environments, however, the user rarely knows how long a process will execute. Another problem with relying on user process duration estimates is that users may supply small (perhaps inaccurate) estimates so that the system will give their programs higher priority. However, the scheduler can be designed to remove this temptation. For example, if a process runs longer than estimated, the system could terminate it and reduce the priority of that user's other processes, even invoking penalties. A second method is to run the process for the estimated time plus a small percentage extra, then "shelve" it (i.e., preserve it in its current form) so that the system may restart it at a later time.15 SPF derives from a discipline called short job first (SJF), which might have worked well scheduling jobs in factories but clearly is inappropriate for low-level scheduling in operating systems. SPF, like FIFO, is nonpreemptive and thus not suitable for environments in which reasonable response times must be guaranteed. 8.7 Scheduling Algorithms 349 Self Review 1. Why is SPF more desirable than FIFO when system throughput is a primary system objective? 2. Why is SPF inappropriate for low-level scheduling in today's operating systems? Ans: 1) SPF reduces average wait times, which increases throughput. 2) SPF does not provide processes with fast response times, which is essential in today's user-friendly, multiprogrammed, interactive systems. 8.7.4 Highest-Response-Ratio-Next (HRRN) Scheduling Brinch Hansen developed the highest-response-ratio-next (HRRN) policy that corrects some of the weaknesses in SPF, particularly the excessive bias against longer processes and the excessive favoritism toward short processes. HRRN is a nonpreemptive scheduling discipline in which each process's priority is a function not only of its service time but also of its time spent waiting for service.16 Once a process obtains it, the process runs to completion. HRRN calculates dynamic priorities according to the formula Because the service time appears in the denominator, shorter processes receive preference. However, because the waiting time appears in the numerator, longer processes that have been waiting will also be given favorable treatment. This technique is similar to aging and prevents the scheduler from indefinitely postponing processes. Self Review 1. (T/F) With HRRN scheduling, short processes are always scheduled before long ones. 2. Process P1 has declared a service time of 5 seconds and has been waiting for 20 seconds. Process P2 has declared a service time of 3 seconds and has been waiting for 9 seconds. If the system uses HRRN, which process will execute first? Ans: 1) False. The longer a process waits, the more likely it will be scheduled before shorter processes. 2) In this case, process P1 has a priority of 5 and P2 has a priority of 4, so the system executes P1 first. 8.7.5 Shortest-Remaining-Time (SRT) Scheduling Shortest-remaining-time (SRT) scheduling is the preemptive counterpart of SPF that attempts to increase throughput by servicing small arriving processes. SRT was effective for job-processing systems that received a stream of incoming jobs, but it is no longer useful in most of today's operating systems. In SRT, the scheduler selects the process with the smallest estimated run-time-to-completion. In SPF, once a process begins executing, it runs to completion. In SRT, a newly arriving process with a shorter estimated run-time preempts a running process with a longer run-time-to-completion. Again, SRT requires estimates of future process behavior to be effective, and the designer must account for potential user abuse of this system scheduling strategy. 350 Processor Scheduling The algorithm must maintain information about the elapsed service time of the running process and perform occasional preemptions. Newly arriving processes with short run-times execute almost immediately. Longer processes, however, have an even longer mean waiting time and variance of waiting times than in SPF. These factors contribute to a larger overhead in SRT than SPF. The SRT algorithm offers minimum wait times in theory, but in certain situations, due to preemption overhead, SPF might perform better. For example, consider a system in which a running process is almost complete and a new process with a small estimated service time arrives. Should the running process be preempted? The SRT discipline would perform the preemption, but this may not be the optimal choice. One solution is to guarantee that a running process is no longer preemptible when its remaining run time reaches a low-end threshold. A similar problem arises when a newly arriving process requires slightly less time to complete than the running process. Although the algorithm would correctly preempt the running process, this may not be the optimal policy. For example, if the preemption overhead is greater than the difference in service times between the two processes, preemption results in poorer performance. As these examples illustrate, the operating systems designer must carefully weigh the overhead of resource-management mechanisms against the anticipated benefits. Also we see that relatively simple scheduling policies can yield poor performance for subtle reasons. Self Review 1. Is SRT an effective processor scheduling algorithm for interactive systems? 2. Why is SRT an ineffective scheduling algorithm for real-time processes? Ans: 1) At first glance, SRT may seem to be an effective algorithm for interactive processes if the tasks performed before issuing I/O are short in duration. However, SRT determines priority based on the run-time-to-completion, not the run-time-to-I/O. Some interactive processes, such as a shell, execute for the lifetime of the session, which would place the shell at the lowest priority level. 2) SRT can result in a large variance of response times, whereas real-time processes require a small variance in response times to ensure that they will always complete their tasks within a particular period of time. 8.7.6 Multilevel Feedback Queues When a process obtains a processor, especially when the process has not as yet had a chance to establish a behavior pattern (e.g., how long it typically runs before generating an I/O request, or which portions of memory the process is currently favoring), the scheduler cannot determine the precise amount of processor time the process will require. I/O-bound processes normally use the processor only briefly before generating an I/O request. Processor-bound processes might use the processor for hours at a time if the system makes it available on a nonpreemptible basis. A scheduling algorithm should typically favor short processes, favor I/O-bound processes to get good I/O device utilization and good interactive response times and should determine the nature of a process as quickly as possible and schedule the process accordingly. 8.7 Scheduling Algorithms 351 Multilevel feedback queues (Fig. 8.4) help accomplish these goals.17 A new process enters the queuing network at the tail of the highest queue. The process progresses through that queue in FIFO order until the process obtains a processor. If the process completes its execution, or if it relinquishes the processor to wait for I/O completion or the completion of some other event, exits the queuing network. If a process's quantum expires before the process voluntarily relinquishes the processor, the system places the process at the tail of the next lower-level queue. As long as the process uses the full quantum provided at each level, it continues to move to the tail of the next lower queue. Usually there is some bottom-level queue through which the Figure 8.4 | Multilevel feedback queues. 352 Processor Scheduling process circulates round-robin until it completes. The process that next gets a processor is the one at the head of the highest nonempty queue in the multilevel feedback queue. A running process is preempted by one arriving in a higher queue. In this system, a process in a lower queue can suffer indefinite postponement, if a higher queue always contains at least one process. This can occur in systems that have a high rate of incoming processes to be serviced, or in which there are several I/O-bound processes consuming their quanta. In many multilevel feedback schemes, the scheduler increases a process's quantum size as the process moves to each lower-level queue. Thus, the longer a process has been in the queuing network, the larger the quantum it receives each time it obtains the processor. We will soon see why this is appropriate. Let us examine the treatment that processes receive by considering how this discipline responds to different process types. Multilevel feedback queues favor I/Obound processes and other processes that need only small bursts of processor time, because they enter the network with high priority and obtain a processor quickly. The discipline chooses for the first queue a quantum large enough so that the vast majority of I/O-bound processes (and interactive processes) issue an I/O request before that quantum expires. When a process requests I/O, it leaves the queuing network, having received the desired favored treatment. The process reenters the network when it next becomes ready. Now consider a processor-bound process. The process enters the network with high priority, and the system places it in the highest-level queue. At this point the queuing network does not "know" whether the process is processor bound or I/O bound—the network's goal is to decide this quickly. The process obtains the processor quickly, uses its full quantum, its quantum expires, and the scheduler moves the process to the next lower queue. Now the process has a lower priority, and incoming processes obtain the processor first. This means that interactive processes will still continue to receive good response times, even as many processor-bound processes sink lower in the queuing network. Eventually, the processor-bound process does obtain the processor, receives a larger quantum than in the highest queue and again uses its full quantum. The scheduler then places the process at the tail of the nextlower queue. The process continues moving to lower queues, waits longer between time slices and uses its full quantum each time it gets the processor (unless preempted by an arriving process). Eventually, the processor-bound process arrives at the lowest-level queue, where it circulates round-robin with other processor-bound processes until it completes. Multilevel feedback queues are therefore ideal for separating processes into categories based on their need for the processor. When a process exits the queuing network, it can be "stamped" with the identity of the lowest-level queue in which it resided. When the process reenters the queuing network, the system can place it directly in the queue in which it last completed operation—the scheduler here is employing the heuristic that a process's recent-past behavior is a good indicator of its near-future behavior. This technique allows the scheduler to avoid placing a 8.7 returning processor-bound process into the higher-level queues, where it would interfere with service to high-priority short processes or to I/O-bound processes. Unfortunately, if the scheduler always places a returning process in the lowest queue it occupied the last time it was in the system, the scheduler is unable to respond to changes in a process's behavior (e.g., the process may be transitioning from being processor bound to being I/O bound). The scheduler can solve this problem by recording not only the identity of the lowest-level queue in which the process resided, but also the amount of its quantum that was unused during its last execution. If the process consumes its entire quantum, then it is placed in a lowerlevel queue (if one is available). If the process issues an I/O request before its quantum expires, it can be placed in a higher-level queue. If the process is entering a new phase in which it will change from being processor bound to I/O bound, initially it may experience some sluggishness as the system determines that its nature is changing, but the scheduling algorithm will respond to this change. Another way to make the system responsive to changes in a process's behavior is to allow the process to move up one level in the feedback queuing network each time it voluntarily relinquishes the processor before its quantum expires. Similarly, the scheduler—when assigning a priority—can consider the time a process has spent waiting for service. The scheduler can age the process by promoting it and placing it in the

next higher queue after it has spent a certain amount of time waiting for service. One common variation of the multilevel feedback queuing mechanism is to have a process circulate round-robin several times through each queue before it moves to the next lower queue. Also, the number of cycles through each queue may be increased as the process moves to the next lower queue. This variation attempts to further refine the service that the scheduler provides to I/O-bound versus processor-bound processes. Multilevel feedback queuing is a good example of an adaptive mechanism, i.e., one that responds to the changing behavior of the system it controls.18, 19 Adaptive mechanisms generally require more overhead than nonadaptive ones, but the resulting sensitivity to changes in the system makes the system more responsive and helps justify the increased overhead. As we discuss in Section 20.5.2, Process Scheduling, the Linux process scheduler employs an adaptive mechanism borrowing from multilevel feedback queues. [Note: In the literature, the terms "process scheduling" and "processor scheduling" have been used equivalently] Self Review 1. What scheduling objectives should be evaluated when choosing the number of levels to use in a multilevel feedback queue? 2. Why are adaptive mechanisms desirable in today's schedulers? Ans: 1) One major objective is the variance in response times. Increasing the number of levels can cause processor-bound processes to wait longer, which increases the variance in response times. Another objective to consider is resource utilization. As the number of levels Scheduling Algorithms 353 354 Processor Scheduling increases, many processes can execute and issue I/Os before their quanta expire, resulting in effective use of both processor time and I/O devices. 2) In today's computers, many applications can be both computationally intensive and I/O bound. For example, a streaming video player must perform I/O to retrieve video clip data from a remote server, then must perform computationally intensive operations to decode and display the video images. Similarly, an interactive game such as a flight simulator must respond to user input (e.g., joystick movements) while rendering complicated 3D scenes. Adaptive mechanisms enable the system to provide alternate responses to these processes as they alternate between I/O-bound and processor-bound behavior. 8.7.7 Fair Share Scheduling Systems generally support various sets of related processes. For example, UNIX (and other multiuser) systems group processes that belong to an individual user. Fair share scheduling (FSS) supports scheduling across such sets of processes.20, 21, 22, 23 Fair share scheduling enables a system to ensure fairness across groups of processes by restricting each group to a certain subset of the system resources. In the UNIX environment, for example, FSS was developed specifically to "give a pre-specified rate of system resources ... to a related set of users."24 Let us consider an example in which fair share scheduling would be useful. Imagine a research group whose members all share one multiuser system. They are divided into two groups. The principal investigators —of which there are few—use the system to perform important, computationally intensive work such as running simulations. The research assistants—of which there are many—use the system for less intensive work such as aggregating data and printing results. Now imagine that many research assistants and only one principal investigator are using the system. The research assistants may consume a majority of the processor time, to the detriment of the principal investigator, who must perform the more important work. However, if the system allowed the research assistants group to use only 25 percent of the processor time and allowed the principal investigators group to use 75 percent, the principal investigator would not suffer such service degradation. In this way, fair share scheduling ensures that the performance of a process is affected only by the population of its process group, and not by the user population as a whole. Let us investigate how fair share scheduling operates in a UNIX system. Normally, UNIX considers resource-consumption rates across all processes (Fig. 8.5). Under FSS, however, the system apportions the resources to various fair share groups (Fig. 8.6). It distributes resources not used by one fair share group to other fair share groups in proportion to their relative needs. UNIX commands establish fair share groups and associate specific users with them. 25 For the purpose of this discussion, assume that UNIX uses a priority roundrobin process scheduler.26 Each process has a priority, and the scheduler associates processes of a given priority with a priority queue for that value. The process scheduler selects the ready process at the head of the highest-priority queue. Processes 8.7 Scheduling Algorithms 355 Figure 8.5 | Standard UNIX process scheduler. The scheduler grants the processor to users, each of whom may have many processes. (Property of AT&T Archives. Reprinted with permission f AT&T)27 within a given priority are scheduled round-robin. A process requiring further service after being preempted receives a lower priority. Kernel priorities are high and apply to processes executing in the kernel; user priorities are lower. Disk events receive higher priority than terminal events. The scheduler assigns the user priority as the ratio of recent processor usage to elapsed real time; the lower the elapsed time, the higher the priority. The fair share groups are prioritized by how close they are to achieving their specified resource-utilization goals. Groups doing poorly receive higher priority; groups doing well, lower priority. Self Review 1. In FSS, why should groups that are not achieving their resource-utilization goals be given higher priority? 2. How does FSS differ from standard process scheduling disciplines? Ans: 1) Processes that are using fewer resources than specified by their resource-utilization goals are probably suffering from low levels of service. By increasing their priority, the system ensures that these processes execute long enough to use their required resources. 2) FSS apportions resources to groups of processes, whereas standard process schedulers allow all processes to compete for all resources on an equal footing. 356 Processor Scheduling Figure 8.6 | Fair share scheduler. The fair share scheduler divides system resource capacity into portions, which are then allocated by process schedulers assigned to various fair share groups. (Property of AT&T Archives. Reprinted with permission of AT&T)28 8.8 Deadline Scheduling In deadline scheduling, certain processes are scheduled to be completed by a specific time or deadline. These processes may have high value if delivered on time and little or no value otherwise.29, 30, 31 Deadline scheduling is complex. First, the user must supply the precise resource requirements in advance to ensure that the process is completed by its deadline. Such information is rarely available. Second, the system should execute the deadline process without severely degrading service to other users. Also, the system must carefully plan its resource requirements through to the deadline. This may be difficult, because new processes may arrive, making unpredictable demands. Finally, if many deadline processes are active at once, scheduling could become extremely complex. The intensive resource management that deadline scheduling requires may generate substantial overhead (see the Operating Systems Thinking feature, Intensity of Resource Management vs. Relative Resource Value). Net consumption of system resources may be high, degrading service to other processes. 8.9 Real-Time Scheduling 357 As we will see in the next section, deadline scheduling is important to realtime process scheduling. Dertouzos demonstrated that when all processes can meet their deadlines regardless of the order in which they are executed, it is optimal to schedule processes with the earliest deadlines first.32 However, when the system becomes overloaded, the system must allocate significant processor time to the scheduler to determine the proper order in which to execute processes so that they meet their deadlines. Recent research focuses on the speed33-34 or number 35 of processors the system must allocate to the scheduler so that deadlines are met. Self Review 1. Why is it important for the user to specify in advance the resources a process needs? 2. Why is it difficult to meet a process's stated deadline? Ans: 1) This allows the scheduler to ensure that resources will be available for the process so that it can complete its task before its deadline. 2) New processes may arrive and place unpredictable demands on the system that prevent it from meeting a process's deadline, resources could fail, the process could dramatically change its behavior and so on. 8.9 Real-Time Scheduling A primary objective of the scheduling algorithms that we presented in Section 8.7, scheduling Algorithms, is to ensure high resource utilization. Processes that must execute periodically (such as once a minute) require different scheduling algorithms. For example, the unbounded wait times for SPF could be catastrophic for a process that checks the temperature of a nuclear reactor. Similarly, a system using SRT to schedule a process that plays a video clip could produce choppy playback. Real-time scheduling meets the needs of processes that must produce correct output by a certain time (i.e., that have a timing constraint).36 A real-time process may divide its instructions into separate tasks, each of which must complete by a dead- Operating Systems Thinking Intensity of Resource Management vs. Relative Resource Value Operating systems manage hard- the relative value of those devices. The operating systems ware and software resources. A recurring theme you will see throughout this book is that the intensity w i t h which operating resources, and to their scarcity and intensity of use. For example, the processor and high-speed cache memories are managed designer must be aware of trends that could affect the relative value of system resources and people time and must respond systems need to manage particu- much more intensively than sec- quickly to change. lar resources is proportional to ondary storage and other I/O 358 Processor Scheduling line. Other real-time processes might perform a certain task periodically, such as updating the locations of planes in an air traffic control system. Real-time schedulers must ensure that timing constraints are met (see the Mini Case Study, RealTime Operating Systems). Real-time scheduling disciplines are classified by how well they meet a process's deadlines. Soft real-time scheduling ensures that real-time processes are dispatched before other processes in the system, but does not guarantee which, if any, will meet their timing constraints.37-38 Soft real-time scheduling is commonly implemented in personal computers, where smooth playback of multimedia is desirable, but occasional choppiness under heavy system loads is tolerated. Soft real-time systems can benefit from high interrupting clock rates to prevent the system from getting "stuck" executing one process while others miss their deadlines. However, the Mini Case Study Real-Time Operating Systems A real-time system differs from a standard system in that every operation must give results that are both correct and returned within a certain amount of time. 3 9 Real-time systems are used in time-critical applications such as monitoring sensors. They are often small, embedded systems. Real-time operating systems (RTOSs) must be carefully designed to achieve these goals. (Real-time process scheduling is discussed in Section 8.9, Real-Time Scheduling.) The control program for the SAGE (Semi-Automatic Ground Environment) project may have been the first real-time operating system.40, 41 SAGE was an Air Force project to defend against possible bomber attacks in the Cold War.42 Implemented in the end of the 1950s, this system integrated 56 IBM AN/FSQ-7 digital vacuumtube computers that monitored data from radar systems across the country to track all planes in the United States airspace.43, 44 It analyzed this information and directed the interceptor fighter planes, thus requiring real-time response.45 The operating system to handle this task was the largest computer program of its time. 4 6 In contrast, today's RTOSs aim for minimal size. There are many available, but QNX and VxWorks lead the field. QNX implements the POSIX standard APIs w i t h its own microkernel and is tailored to embedded systems. It also uses message passing for interprocess communication. 47 Similarly, VxWorks is a microkernel system that complies w i t h the POSIX standard and is widely used in embedded systems.48 QNX performs better than VxWorks on Intel x86 platforms, 49 but QNX was not available on other processors until version 6.1 (the current version is 6.2).50 VxWorks, however, has concentrated on the PowerPC processor and has a history of cross-platform compatibility.51 VxWorks is currently the most common RTOS for embedded systems.52 Additional RTOSs include Windows CE .NET,53 OS9, 54 OSE,55 and Linux distributions such as uCLinux.56 8.9 Real-Time Scheduling 359 system can incur significant overhead if the interrupt rate is too high, resulting in poor performance or missed deadlines.57 Hard real-time scheduling guarantees that a process's timing constraints are always met. Each task specified by a hard real-time process must complete before its deadline; failing to do so could produce catastrophic results, including invalid work, system failure or even harm to the system's users.58, 59 Hard real-time systems may contain periodic processes that perform their computations at regular time intervals (e.g., gathering air traffic control data every second) and asynchronous processes that execute in response to events (e.g., responding to high temperatures in a power plant's core). 60 Static Real-Time Scheduling Algorithms Static real-time scheduling algorithms do not adjust process priority over time. Because priorities are calculated only once, static real-time scheduling algorithms tend to be simple and incur little overhead. Such algorithms are limited because they cannot adjust to variable process behavior and depend on resources staying up and running to ensure that timing constraints are met. Hard real-time systems tend to use static scheduling algorithms, because they incur low overhead and it is relatively easy to prove that each process's timing constraints will be met. The rate-monotonic (RM) algorithm, for example, is a preemptive, priority-based round-robin algorithm that increases a process's priority linearly (i.e., monotonically) with the frequency (i.e., the rate) with which it must execute. This static scheduling algorithm favors periodic processes that execute frequently.61 The deadline rate-monotonic algorithm can be used when a periodic process specifies a deadline that is not equal to its period.62 Dynamic Real-Time Scheduling Algorithms Dynamic real-time scheduling algorithms schedule processes by adjusting their priorities during execution, which can incur significant overhead. Some algorithms attempt to minimize the scheduling overhead by assigning static priorities to some processes and dynamic priorities to others. 63 Earliest-deadline-first (EDF) is a preemptive scheduling algorithm that dispatches the process with the earliest deadline. If an arriving process has an earlier deadline than the running process, the system preempts the running process and dispatches the arriving process. The objective is to maximize throughput by satisfying the deadlines of the largest number of processes per unit time (analogous to the SRT algorithm) and minimize the average wait time (which prevents short processes from missing their deadlines while long processes execute). Dertouzos proved that, if a system provides hardware preemption (e.g., timer interrupts) and the real-time processes being scheduled are not interdependent, EDF minimizes the amount of time by which the "most tardy" process misses its deadline. 64 However, many real-time systems do not provide hardware preemption, so other algorithms must be employed.65 360 Processor Scheduling The minimum-laxity-first algorithm is similar to EDF but bases priority on a process's laxity. Laxity is a measure of a process's importance based on the amount of time until its deadline and the remaining execution time until its task (which may be periodic) has completed. Laxity is computed using the formula L = D - (T+C), where L is the laxity, D the process's deadline, T the current time and C the process's remaining execution time. For example, if the current time is 5, the deadline for a process is 9 and the process requires 3 units of time to complete, the laxity is 1. If a process has 0 laxity, it must be dispatched immediately or it will miss its deadline. Priorities in minimum-laxity-first are more accurate than those in EDF because they are determined by including the remaining processor time each process requires to complete its task. However, such information is often unavailable.66 Self Review 1. Why are most hard real-time scheduling algorithms static? 2. When does the minimum-laxity-first algorithm degenerate to EDF? Can this ever occur? Ans: 1) Hard real-time systems must guarantee that processes' deadlines are met. Static scheduling algorithms facilitate proving this property for a particular system and reduce the implementation overhead. 2) The minimum-laxity-first algorithm degenerates to the EDF algorithm when C is identical for all processes at any given time. It is possible, though highly unlikely, that this would occur. 8.10 Java Thread Scheduling As discussed in Section 4.6, Threading Models, operating systems provide various levels of support for threads. When scheduling a multithreaded process that implements user-level threads, the operating system is unaware that the process is multithreaded and therefore dispatches the process as one unit, requiring a user-level library to schedule its threads. If the system supports kernel-level threads, it may schedule each thread independently from others within the same process. Still other systems support scheduler activations that assign each process to a kernel-level thread that enables the process's user-level library to perform scheduling operations. System designers must determine how to allocate quanta to threads, and in what order and with what priorities to schedule threads within a process. For example, a "fair-share" approach divides the quantum allocated to a process among its threads. This prevents a multithreaded process from receiving high levels of service simply by creating a large number of threads. Further, the order in which threads are executed can impact their performance if they rely on one another to continue their tasks. In this section, we present Java thread scheduling. Thread scheduling in Windows XP is discussed in Section 21.6.2, Thread Scheduling. The Linux scheduler (which dispatches both processes and threads) is discussed in Section 20.5.2, Process Scheduling. 8.10 Java Thread Scheduling 361 One feature of the Java programming language and its virtual machine is that every Java applet or application is multithreaded. Each Java thread is assigned a priority in the range between Thread.MIN_PRIORITY (a constant of 1) and Thread. MAX_PRIORITY (a constant of 10). By default, each thread is given priority Thread. NORM_PRIORITY (a constant of 5). Each new thread inherits the priority of the thread that creates it. Depending on the platform, Java implements threads in either user or kernel space (see Section 4.6,Threading Models). 67 - 68 When implementing threads in user space, the Java runtime relies on timeslicing to perform preemptive thread scheduling. Without timeslicing, each thread in a set of equal-priority threads runs to completion, unless it leaves the running state and enters the waiting, sleeping or blocked state, or it is preempted by a higher-priority thread. With timeslicing, each thread receives a quantum during which it can execute. The Java thread scheduler ensures that the highest-priority thread in the Java virtual machine is running at all times. If there are multiple threads at the priority level, those threads execute using round-robin. Figure 8.7 illustrates Java's multilevel priority queue for threads. In the figure, assuming a single-processor computer, threads A and B each execute for a quantum at a time in round-robin fashion until both threads complete execution. Next, thread C runs to completion. Threads D, E and F each execute for a quantum in round-robin fashion until they all complete execution. This process continues until all threads run to completion. Note that, depending on the operating system, arriving higher-priority threads could indefinitely postpone the execution of lower-priority ones. A thread can call the y i e l d method of class Thread to give other threads a chance to execute. Because the operating system preempts the current thread whenever a higher-priority one becomes ready, a thread cannot yi el d to a higherpriority thread. Similarly, y i e l d always allows the highest-priority ready thread to run, so if all of the ready threads are of lower priority than the thread calling yi el d, the current thread will have the highest priority and will continue executing. Therefore, a thread yi el ds to give threads of an equal priority a chance to run. On a timesliced system this is unnecessary, because threads of equal priority will each execute for their quantum (or until they lose the processor for some other reason), and other threads of equal priority will execute round-robin. Thus yi el d is appropriate for nontimesliced systems (e.g., early versions of Solaris69), in which a thread would ordinarily run to completion before another thread of equal priority would have an opportunity to run. 1. Why does Java provide the yi el d method? Why would a programmer ever use yield? 2. (T/F) A Java thread of lower priority will never run while a thread of higher priority is ready. Ans: 1) Method yi el d allows the current thread to voluntarily release the processor and let a thread of equal priority execute. Because Java applications are designed to be portable and because the programmer cannot be certain that a particular platform supports timeslicing, 362 Processor Scheduling Figure 8.7 | Java thread priority scheduling. programmers use y i e l d to ensure that their applications execute properly on all platforms. 2) True. Java schedulers will execute the highest-priority ready thread. Web Resources www.linux-tutorial.info/cgi-bin/display.pl?85&0&0&0&3 This portion of the Linux tutorial describes process scheduling in Linux. www.oreilly.com/catalog/linuxkernel/chapter/ ch10.html Contains the online version of O'Reilly's Linux kernel book This page details process scheduling in Linux version 2.4. developer.apple.com/techpubs/mac/Processes/Processes-16.html Describes process scheduling for the Apple Macintosh. www.javaworld.com/javaworld/jw-07-2002/jw-0703javal01.html Describes thread scheduling in Java. Summary csl.cse.ucsc.edu/rt.shtml Mails the results of the real-time operating system scheduler search unit at the University of California, Santa Cruz. 363 www.ittc.ku.edu/kurt/ Describes the Kansas University real-time scheduler that uses a modified Linux kernel. Summary When a system has a choice of processes to execute, it must have a strategy—called a processor scheduling policy (or discipline)—for deciding which process to run at a given time. High-level scheduling—sometimes called job scheduling or long-term scheduling—determines which jobs the system allows to compete actively for system resources. The highlevel scheduling policy dictates the degree of multiprogramming—the total number of processes in a system at a given time. After the high-level scheduling policy has admitted a job (which may contain one or more processes) to the system, the intermediate-level scheduling policy determines which processes shall be allowed to compete for a processor. This policy responds to short-term fluctuations in system load. A system's low-level scheduling policy determines which ready process the system will assign to a processor when one next becomes available. Low-level scheduling policies often assign a priority to each process, reflecting its importance—the more important a process, the more likely the scheduling policy is to select it to execute next. A scheduling discipline can be preemptive or nonpreemptive. To prevent users from monopolizing the system (either maliciously or accidentally), preemptive schedulers set an interrupting clock or interval timer that periodically generates an interrupt. Priorities may be statically assigned or be changed dynamically during the course of execution. When developing a scheduling discipline, a system designer must consider a variety of objectives, such as the type of system and the users' needs. These objectives may include maximizing throughput, maximizing the number of interactive users receiving "acceptable" response times, maximizing resource utilization, avoiding indefinite postponement, enforcing priorities, minimizing overhead and ensuring predictability of response times. To accomplish these goals, a system can use techniques such as aging processes and favoring processes whose requests can be satisfied quickly. Many scheduling disciplines exhibit fairness, predictability and scalability. Operating system designers can use system objectives to determine the criteria on which scheduling decisions are made. Perhaps the most important concern is how a process uses the processor (i.e., whether it is processor bound or I/O bound). A scheduling discipline might also consider whether a process is batch or interactive. In a system that employs priorities, the scheduler should favor processes with higher priorities. Schedulers employ algorithms that make choices about preemptibility, priorities, running time and other process characteristics. FIFO (also called FCFS) is a nonpreemptive algorithm that dispatches processes according to their arrival time at the ready queue. In round-robin (RR) scheduling, processes are dispatched FIFO but are given a limited amount of processor time called a time slice or a quantum. A variant of round-robin called selfish round-robin (SRR) initially places a process in a holding queue until its priority reaches the level of processes in the active queue, at which point, the process is placed in the active queue and scheduled round-robin with others in the queue. Determination of quantum size is critical to the effective operation of a computer system. The "optimal" quantum is large enough so that the vast majority of I/Obound and interactive requests require less time than the duration of the quantum. The optimal quantum size varies from system to system and under different loads. Shortest-process-first (SPF) is a nonpreemptive scheduling discipline in which the scheduler selects the waiting process with the smallest estimated run-time-tocompletion. SPF reduces average waiting time over FIFO but increases the variance in response times. Shortestremaining-time (SRT) scheduling is the preemptive counterpart of SPF that selects the process with the smallest estimated run-time-to-completion. The SRT algorithm offers minimum wait times in theory, but in certain situations, due to preemption overhead, SPF might actually perform better. Highest-response-ratio-next (HRRN) is a nonpreemptive scheduling discipline in which the priority of each process is a function not only of its service time but also of the amount of time it has been waiting for service. Multilevel feedback queues allow a scheduler to dynamically adjust to process behavior. The process that gets a processor next is the one that reaches the head of the highest nonempty queue in the multilevel feedback queuing network. Processor-bound processes are placed at the lowestlevel queue, and I/O-bound processes tend to be located in the higher-level queues. A running process is preempted by a process arriving in a higher queue. Multilevel feedback queuing is a good example of an adaptive mechanism. 364 Processor Scheduling Fair share scheduling (FSS) supports scheduling across related processes and threads. It enables a system to ensure fairness across groups of processes by restricting each group to a certain subset of system resources. In the UNIX environment, for example, FSS was developed specifically to "give a prespecified rate of system resources ... to a related set of users." In deadline scheduling, certain processes are scheduled to be completed by a specific time or deadline. These processes may have high value if delivered on time and be worthless otherwise. Deadline scheduling can be difficult to implement. Real-time scheduling must repeatedly meet processes' deadlines as they execute. Soft real-time scheduling ensures that real-time processes are dispatched before other processes in the system. Hard real-time scheduling guarantees that a process's deadlines are met. Hard realtime processes must meet their deadlines; failing to do so could be catastrophic, resulting in invalid work, system failure or even harm to the system's users. The rate-monotonic (RM) algorithm is a static scheduling priority-based roundrobin algorithm in which priorities are increase (monotonically) with the rate at which a process must be scheduled. The earliest-deadline-first (EDF) is a preemptive scheduling algorithm that favors the process with the earliest deadline. The minimum-laxity-first algorithm is similar to EDF. but bases priority on a process's laxity, which measures the difference between the time a process requires to complete and the time remaining until that its deadline. The operating system may dispatch a process's threads individually or it may schedule a multithreaded process as a unit, requiring user-level libraries to schedule their threads. System designers must determine how to allocate quanta to threads, in what order, and what priorities to assign in scheduling threads within a process. In Java, each thread is assigned a priority in the range 1-10. Depending on the platform, Java implements threads in either user or kernel space (see Section 4.6, Threading Models). When implementing threads in user space, the Java runtime relies on timeslicing to perform preemptive thread scheduling. The Java thread scheduler ensures that the highest-priority thread in the Java virtual machine is running at all times. If there are multiple threads at the same priority level, those threads execute using roundrobin. A thread can call the yield method to give other threads of an equal priority a chance to run. Key Terms adaptive mechanism—Control entity that adjusts a system in response to its changing behavior. earliest-deadline-first (EDF)—Scheduling policy that gives a processor to the process with the closest deadline. admission scheduling—See high-level scheduling. fairness—Property of a scheduling algorithm that treats all processes equally. aging of priorities—Increasing a process's priority gradually, based on how long it has been in the system. asynchronous real-time process—Real-time process that executes in response to events. batch process—Process that executes without user interaction. deadline rate-monotonic scheduling—Scheduling policy in real-time systems that meets a periodic process's deadline that does not equal its period. fair share group—Group of processes that receives a percentage of the processor time under a fair share scheduling (FSS) policy. fair share scheduling (FSS)—Scheduling policy developed for AT&T's UNIX system that places processes in groups and assigns these groups a percentage of processor time. first-in-first-out (FIFO)—Nonpreemptive scheduling policy that dispatches processes according to their arrival time in the ready queue. deadline scheduling—Scheduling policy that assigns priority based on processes' completion deadlines to ensure that processes complete on time. job scheduling—See high-level scheduling. degree of multiprogramming—Total number of processes in main memory at a given time. hard real-time scheduling—Scheduling policy that ensures processes meet their deadlines. dispatcher—Entity that assigns a processor to a process. highest-response-ratio-next (HRRN)—Scheduling policy that assigns priority based on a process's service time and the amount of time the process has been waiting. dispatching—Act of assigning a processor to a process. dynamic real-time scheduling algorithm—Scheduling algorithm that uses deadlines to assign priorities to processes throughout execution. high-level scheduling—Determining which jobs a system allows to compete actively for system resources. Key interactive process—Process that requires user input as it executes. intermediate-level scheduling—Determining which processes may enter the low-level scheduler to compete for a processor. I/O-bound—Process that tends to use a processor for a short time before generating an I/O request and relinquishing a processor. latency (process scheduling)—Time a task spends in a system before it is serviced. laxity—Value determined by subtracting the sum of the current time and a process's remaining execution time from the process's deadline. This value decreases as a process nears its deadline. long-term scheduling—See high-level scheduling. low-level scheduling—Determining which process will gain control of a processor. minimum-laxity-first—Scheduling policy that assigns higher priority to processes that will finish with minimal processor usage. multilevel feedback queue—Process scheduling structure that groups processes of the same priority in the same roundrobin queue. Processor-bound processes are placed in lower-priority queues because they are typically batch processes that do not require fast response times. I/Obound processes, which exit the system quickly due to I/O, remain in high-priority queues. These processes often correspond to interactive processes that should experience fast response times. nonpreemptive scheduling—Scheduling policy that does not allow the system to remove a processor from a process until that process voluntarily relinquishes its processor or runs to completion. periodic real-time process—Real-time process that performs computation at a regular time interval. predictability—Goal of a scheduling algorithm that ensures that a process takes the same amount of time to execute regardless of the system load. preemptive scheduling—Scheduling policy that allows the system to remove a processor from a process. priority—Measure of a process's or thread's importance used to determine the order and duration of execution. Terms 365 processor scheduling policy—Strategy used by a system to determine when and for how long to assign processors to processes. purchase priority—to pay to receive higher priority in a system. quantum—Amount of time that a process is allowed to run on a processor before the process is preempted. rate-monotonic (RM) scheduling—Real-time scheduling policy that sets priority to a value that is proportional to the rate at which the process must be dispatched. real-time scheduling—Scheduling policy that bases priority on timing constraints. round-robin (RR) scheduling—Scheduling policy that permits each ready process to execute for at most one quantum per round. After the last process in the queue has executed once, the scheduler begins a new round by scheduling the first process in the queue, scalability (scheduler) —Characteristic of a scheduler that ensures system performance degrades gracefully under heavy loads. selfish round-robin (SRR) scheduling—Variant of round-robin scheduling in which processes age at different rates. Processes that enter the system are placed in a holding queue, where they wait until their priority is high enough for them to be placed in the active queue, in which processes compete for processor time. shortest-process-first (SPF) scheduling—Nonpreemptive scheduling algorithm in which the scheduler selects a process with the smallest estimated run-time-to-completion and runs the process to completion. shortest-remaining-time (SRT) scheduling—Preemptive version of SPF in which the scheduler selects a process with the smallest estimated remaining run-time-to-completion. soft real-time scheduling—Scheduling policy that guarantees that real-time processes are scheduled with higher priority than non-real-time processes. static real-time scheduling algorithm—Scheduling algorithm that uses timing constraints to assign fixed priorities to processes before execution. throughput—Number of processes that complete per unit time. time slice— See quantum. processor-bound—Process that consumes its quantum when executing. These processes tend to be calculation intensive and issue few, if any, I/O requests. timeslicing—Scheduling each process to execute for at most one quantum before preemption. processor scheduling discipline—See processor scheduling policy. timing constraint—Time period during which a process (or subset of a process's instructions) must complete. 366 Processor Scheduling Exercises 8.1 ers. Distinguish among the following three levels of schedul- iii. The system should admit processes to create a mix that will keep most devices busy. a. high-level scheduler iv. The system should favor important processes. b. intermediate-level scheduler v. c. dispatcher 8.2 Which level of scheduler should make a decision on each of the following questions? vi. During peak periods, the system should not collapse from the overhead it takes to manage a large number of processes. a. Which ready process should be assigned a processor when one becomes available? vii. The system should favor I/O-bound processes. b. Which of a series of waiting batch processes that have been spooled to disk should next be initiated? c. Which processes should be temporarily suspended to relieve a short-term burden on the processor? d. Which temporarily suspended process that is known to be I/O bound should be activated to balance the multiprogramming mix? 8.3 Distinguish between a scheduling policy and a scheduling mechanism. 8.4 The following are common scheduling objectives. a. to be fair b. to maximize throughput c. to maximize the number of interactive users receiving acceptable response times An important process arrives but cannot proceed because an unimportant process is holding the resources the important process needs. viii. Context switches should execute as quickly as possible. 8.5 The following are common scheduling criteria. a. I/O boundedness of a process b. processor boundedness of a process c. whether the process is batch or interactive d. urgency of a fast response e. process priority f. frequency of a process is being preempted by higherpriority processes g. priorities of processes waiting for resources held by other processes h. accumulated waiting time d. to be predictable i. accumulated execution time e. to minimize overhead j . estimated run-time-to-completion. f. to balance resource utilization g. to achieve a balance between response and utilization h. to avoid indefinite postponement For each of the following, indicate which of the preceding scheduling criteria is most appropriate. i. In a real-time spacecraft monitoring system, the computer must respond immediately to signals received from the spacecraft. ii. Although a process has been receiving occasional service, it is making only nominal progress. i. to obey priorities j. to give preference to processes that hold key resources k. to give a lower grade of service to high overhead processes l. to degrade gracefully under heavy loads Which of the preceding objectives most directly applies to each of the following? i. If a user has been waiting for an excessive amount of time, favor that user. ii. The user who runs a payroll job for a 1000employee company expects the job to take about the same amount of time each week. iii. How often does the process voluntarily give up the processor for I/O before its quantum expires? iv. Is the user present and expecting fast interactive response times, or is the user absent? v. One goal of processor scheduling is to minimize average waiting times. vi. Processes holding resources in demand by other processes should have higher priorities. vii. Nearly complete processes should have higher priorities. Exercises 8.6 State which of the following are true and which false. Justify your answers. a. A process scheduling discipline is preemptive if the processor cannot be forcibly removed from a process. b. Real-time systems generally use preemptive processor scheduling. c. Timesharing systems generally use nonpreemptive processor scheduling. d. Turnaround times are more predictable in preemptive than in nonpreemptive systems. e. One weakness of priority schemes is that the system will faithfully honor the priorities, but the priorities themselves may not be meaningful. 8.7 Why should processes be prohibited from setting the interrupting clock? 8.8 State which of the following refer to "static priorities" and which to "dynamic priorities." a. are easier to implement 367 i. Arrange the schemes above in order from lowest to highest runtime overhead. ii. Arrange the schemes in order from least to most responsive to variations in individual processes and system load. iii. Relate your answers in (i) and (ii) to one another. 8.14 State why each of the following is incorrect. a. SPF never has a higher throughput than SRT. b. SPF is fair. c. The shorter the process, the better the service it should receive. d. Because SPF gives preference to short processes, it is useful in timesharing. 8.15 State some weaknesses in SRT. How would you modify the scheme to get better performance? 8.16 Answer each of the following questions about Brinch Hansen's HRRN strategy. b. require less runtime overhead a. How does HRRN prevent indefinite postponement? c. are more responsive to changes in a process's environment b. How does HRRN decrease the favoritism shown by other strategies to short new processes? d. require more careful deliberation over the initial priority value chosen Give several reasons why deadline scheduling is complex. c. Suppose two processes have been waiting for about the same time. Are their priorities about the same? Explain your answer. 8.10 Give an example showing why FIFO is not an appropriate processor scheduling scheme for interactive users. 8.17 Show how multilevel feedback queues accomplish each of the following scheduling goals. 8.9 8.11 Using the example from the previous problem, show why round-robin is a better scheme for interactive users. 8.12 Determining the quantum is a complex and critical task. Assume that the average context-switching time between processes is s, and the average amount of time an I/O-bound process uses before generating an I/O request is t (t >> s). Discuss the effect of each of the following quantum settings, q. a. q is slightly greater than zero b. q = s c. s . Servers," . 87. Linux kernel source code version 2.5.75, < w w w . k e r n e l .org>. 88. Denning, P. J, "The Working Set Model for Program Behavior." Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 323333. 73. Potier, D., "Analysis of Demand Paging Policies with Swapped Working Sets," Proceedings of the Sixth ACM Symposium on Operating Systems Principles, November 1977, pp. 125-131. 89. Denning, P. J, "Virtual Memory," ACM Computing Surveys, Vol. 2, No. 3, September 1970, pp. 153-189. 90. Aho, A. V; P. J. Denning; and J. D. Ullman, "Principles of Optimal Page Replacement," Journal of the ACM, Vol. 18, No. 1, January 1971, pp. 80-93. 74. Franklin, M. A.; G. S. Graham; and R. K. Gupta, "Anomalies with Variable Partition Paging Algorithms," Communications of theACM,Vol. 21, No. 3, March 1978, pp. 232-236. 91. Kaplan, S., et al., "Adaptive Caching for Demand Prepaging. Proceedings of the Third International Symposium on Memor-. Management, 2002, pp. 114-116. 75. Gupta, R. K., and M. A. Franklin, "Working Set and Page Fault Frequency Replacement Algorithms: A Performance Comparison," IEEE Transactions on Computers, Vol. C-27, August 1978, pp. 706-712. 92. Belady, L. A., "A Study of Replacement Algorithms for Virtual Storage Computers," IBM Systems lournal, Vol. 5, No. 2, 196\ pp. 78-101. 76. Denning, P. I, "Working Sets Past and Present," IEEE Transactions on Software Engineering,Vo\. SE-6, No. 1, January 1980, pp. 64-84. 77. Irani, S.; A. Karlin; and S. Phillips, "Strongly Competitive Algorithms for Paging with Locality of Reference," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 228-236. 78. Glass, G, and Pei Cao, "Adaptive Page Replacement Based on Memory Reference Behavior," Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 115-126, June 15-18, 1997, Seattle, Washington. 93. Belady, L. A., and C. J. Kuehner, "Dynamic Space Sharing in Computer Systems," Communications of the ACM,No\. 12, No. 5. May 1969, pp. 282-288. 94. Irani, S.; A. Karlin; and S. Phillips, "Strongly Competitive Algorithms for Paging with Locality of Reference," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms. 1992, pp. 228-236. 95. Albers, S.; L. Favrholdt; and O. Giel, "On Paging with Locality 0: Reference," Proceedings of the 34th Annual ACM Symposium or. Theory of Computation, 2002, pp. 258-267. 96. Morris, J. B., "Demand Paging through the Use of Working Seb on the Maniac II," Communications of the ACM,\'ol. 15, No. 10. October 1972, pp. 867-872. 79. Ganapathy, N, and C. Schimmel, "General Purpose Operating System Support for Multiple Page Sizes," Proceedings of the USENIXConference,1998. 97. Batson, A. P.; S. Ju; and D. Wood, "Measurements of Segmen: Size," Communications of the ACM, Vol. 13, No. 3, March 1970. pp. 155-159. 80. Albers, S.; L. Favrholdt; and O. Giel, "On Paging with Locality of Reference," Proceedings of the 34th Annual ACM Symposium on Theory of Computation, 2002, pp. 258-267. 98. Chu, W. W., and H. Opderbeck, "Performance of Replacement Algorithms with Different Page Sizes," Computer, Vol. 7, No. 11. November 1974, pp. 14-21. Works Cited 521 99- Denning, P. J., "Working Sets Past and Present," IEEE Transactions on Software Engineering,Vol. SE-6, No. 1, January 1980, pp. 64-84. 100. Ganapathy, N., and C. Schimmel, "General Purpose Operating System Support for Multiple Page Sizes," Proceedings of the USENIX Conference, 1998. 101. Freibergs, I. E, "The Dynamic Behavior of Programs," Proceedings AFIPS Fall Joint Computer Conference, Vol. 33, Part 2,1968, pp. 1163-1167. 102. Hatfield, D, "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance," IBM Journal of Research and Development, Vol. 15, No. 1, January 1972, pp. 5862. 103. Morrison, J. E., "User Program Performance in Virtual Storage Systems," IBM Systems Journal,Vol. 12, No. 3,1973, pp. 216-237. 104. Sadeh, E., "An Analysis of the Performance of the Page Fault Frequency (PFF) Replacement Algorithm," Proceedings of the Fifth ACM Symposium on Operating Systems Principles, November 1975, pp. 6-13. 105. Gupta, R. K., and M. A. Franklin, "Working Set and Page Fault Frequency Replacement Algorithms: A Performance Comparison," IEEE Transactions on Computers, Vol. C-27, August 1978, pp. 706-712. 106. Denning, P. J., "Working Sets Past and Present," IEEE Transactions on Software Engineering, Vol. SE-6, No. 1, January 1980, pp. 64-84. 107. Irani, S.; A. Karlin; and S. Phillips, "Strongly Competitive Algorithms for Paging with Locality of Reference," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 228-236. 108. Albers, S.; L. Favrholdts; and O. Giel, "On Paging with Locality of Reference," Proceedings of the 34th Annual ACM Symposium on Theory of Computation, 2002, pp. 258-267. Secondary Storage, Files and Databases A fair request should be followed by the deed in silence. —Dante — Part 4 Computers store programs and vast amounts of data as files and databases on secondary storage devices. The following two chapters explain how operating systems organize and manage data on these devices. We explain the operation of the enormously popular moving-head disk storage device and show how to achieve maximum performance with seek and rotational optimization strategies. RAID (Redundant Arrays of Independent Disks) systems, which achieve high levels of performance and fault tolerance, are presented. We discuss file systems and examine how files are allocated on disk, how free space is managed and how file data is accessed and protected. File servers, and how they are used in distributed systems, are explained. We introduce database systems, and discuss relational databases and the kinds of operating systems services that support database systems. 'Tis in my memory lock'd, And you yourself shall keep the key of it. —William Shakespeare — The path of duty lies in what is near, and man seeks for it in what is remote. —Mencius— A fair request should be followed by the deed in silence. —Dante— ... the latter, in search of the hard latent value with which it alone is concerned, snifs round the mass as instinctively and unerringly as a dog suspicious of some buried bone. —William James— Go where we will on the surface of things, men have been there before us. —Henry David Thoreau— The wheel that squeaks the loudest is the one that gets the grease. —Josh Billings (Henry Wheeler Shaw) — Chapter 12 Disk Performance Optimization Objectives After reading this chapter, you should understand: • how disk input/output is accomplished. • the importance of optimizing disk performance. • seek optimization and rotational optimization. • various disk scheduling strategies. • caching and buffering. • other disk performance improvement techniques. • key schemes for implementing redundant arrays of independent disks (RAID). Chapter Outline 12.1 Introduction 12.2 Evolution of Secondary Storage 12.3 Characteristics of Moving-Head Disk Storage 12.4 Why Disk Scheduling Is Necessary 12.5 Disk Scheduling Strategies 12.5.1 First-Come-First-Served (FCFS) Disk Scheduling 12.5.2 Shortest-Seek-Time-First (SSTF) Disk Scheduling Anecdote: Every Problem Has a Solution and Everu Solution Has a Problem 12.6.2 SPTF and SATF Scheduling 12.7 System Considerations Operating Systems Thinking: Saturation and Bottlenecks 12.8 Caching and Buffering Other 12.9 Disk Performance Techniques Operating Systems Thinking: Compression and Decompression Operating Systems Thinking: Redundancy 12.10 Redundant Arrays of 12.5.3 SCAN Disk Scheduling Independent Disk (RAID) 12.5.4 C-SCAN Disk Scheduling 12.10.1 RAID Overview 12.5.5 FSCAN and N-Step SCAN Disk Scheduling 12.5.6 LOCK and C-LOCK Disk Scheduling 526 12.6 Rotational Optimization 12.6.1 SLTF Scheduling Operating Systems Thinking: MissionCritical Systems Operating Systems Thinking: Fault Tolerance 12.10.2 Level 0 (Striping) 12.10.3 Level 1 (Mirroring) 12.10.4 Level 2 (Bit-Level Hamming ECC Parity) 12.10.5 Level 3 (Bit-level XOR ECC Parity) 12.10.6 Level 4 (Block-level XOR ECC Parity) 12.10.7 Level 5 (Block-Level Distributed XOR ECC Parity) Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited 527 528 Disk Performance Optimization 12.1 Introduction In recent years, processor and main memory speeds have increased more rapidly than those of secondary storage devices, such as hard disks. As a result, processes requesting data on secondary storage tend to experience relatively long service delays. In this chapter, we discuss the characteristics of moving-head disk storage and consider how operating system designers can manage such devices to provide better service to processes. We explain how to optimize disk performance by reordering disk requests to increase throughput, decrease response times and reduce the Variance of response times. We also discuss how operating systems reorganize data on disk and exploit buffers and caches to boost performance/Finally, we discuss Redundant Arrays of Independent Disks (RAIDs), which improve disk access times and fault tolerance by servicing requests using multiple disks at once. 12.2 Evolution of Secondary Storage In early computers, persistent data was stored on punched cards and punched paper tape, which used the absence or presence of holes to represent bits of data. 1 Writing software and loading it into the computer using such media was both labor intensive and slow. The need for an inexpensive, rewritable persistent storage device led researchers to develop magnetic storage, which records bits of data by changing the direction of magnetization of regions on the medium's surface. To access data, a current-carrying device called a read-write head hovers above the medium as it moves. The head reads data by measuring how the magnetized medium changes the current; it "writes" data by using the current to change the magnetization on the medium. A challenge in building these devices is that the head must float extremely close to the medium's surface without touching it. In 1951, the designers of UNIVAC 1 (UNIVersal Automatic Computer) introduced magnetic tape storage, which was both persistent and rewritable. 2 Magnetic tape is a form of sequential access storage, like audio or video cassettes. Such a medium is inappropriate for transaction-processing applications, where, for example, the system must be able to locate and update any record in a fraction of a second. To address this problem, IBM introduced the first commercial hard disk drive. the RAMAC (Random Access Method of Accounting and Control), in 1957. Hard disk drives are random access (also called direct access) devices because they are not limited to accessing data sequentially. RAMAC's capacity totalled five megabytes, and its cost was $50,000; it was generally rented to installations for hundreds of dollars per month. 3 4 Although hard disks provided better performance than magnetic tape storage, their high cost limited their use to large installations. As the decades passed, hard drive capacity and performance increased, while costs declined. Typical personal computer hard disk capacity increased from hundreds of megabytes to several gigabytes during the 1990s while prices fell to a few pennies per megabyte. By 2003, hard drive capacity had exceeded 200GB and cost less than a dollar per gigabyte.5 Due to mechanical constraints that we discuss in the 12.3 Characteristics of moving-Head Disk Storage 529 next section, hard disk speeds have improved more slowly than their capacity. As processor speeds increased and applications consumed larger amounts of data, systems became increasingly I/O bound.6 Research in persistent storage technology continues to focus on increasing capacity and performance. Some solutions attempt to improve the performance of existing magnetic disk devices; others employ novel techniques and media. Self Review 1. Why have many of today's systems become I/O bound? 2. Why are disks more appropriate than tapes for secondary storage? Ans: 1) Processor speeds have increased faster than hard disk speeds. 2) Magnetic tape performs well only in an application in which data is accessed sequentially. In transaction-processing applications and in multiprogrammed systems, requests to access secondary storage from multiple processes can lead to essentially random access patterns. In this case, direct access storage devices are essential. 12.3 Characteristics of Moving-Head Disk Storage Unlike main memory, which provides (nearly) uniform access speed to all of its contents, moving-head disk storage exhibits variable access speed that depends on the relative positions of the read-write head and the requested data. Figure 12.1 shows a simplified view of a moving-head disk.7-8-9 Data is recorded on a series of Figure 12.1 | Schematic side view of a moving-head disk. 530 Disk Performance Optimization magnetic disks, or platters, connected to a spindle that rotates at high speed (typically thousands of revolutions per minute 10). The data on each disk surface is accessed by a read-write head that is separated by a small amount of space from the surface (much smaller than a particle of smoke). For example, the disk in Fig. 12.1 contains two platters, each having two surfaces (top and bottom) and four read-write heads, one for each surface. A readwrite head can access data immediately below (or above) it. Therefore, before data can be accessed, the portion of the disk surface at which the data is to be read (or written) must rotate until it is immediately below (or above) the read-write head. The time it takes for data to rotate from its current position to the beginning of the read-write head is called rotational latency time. A disk's average rotational latency is simply half the time it takes to complete one revolution. Most hard disks exhibit average rotational latency on the order of several milliseconds (Fig. 12.2). As the platters spin, each read-write head sketches out a circular track of data on a disk surface. Each read-write head is positioned at the end of a disk arm, which is attached to an actuator (also called a boom or moving-arm assembly). The disk arm moves parallel to the surface of the disk. When the disk arm moves the readwrite heads to a new position, a different vertical set of circular tracks, or cylinder. becomes accessible. The process of moving the disk arm to a new cylinder is called a seek operation. 1112 To locate small units of data, disks divide tracks into several sectors, often 512 bytes (Fig. 12.3).13 Therefore, an operating system can locate a particular data item by specifying the head (which indicates which disk surface to read from), the cylinder (which indicates which track to read from), and the sector in which the data is located. To access a particular record of data on a moving-head disk, several operations are usually necessary (Fig. 12.4). First, the disk arm must move to the appropriate cylinder (i.e., perform a seek operation). The time it takes for the head to move from its current cylinder to the one containing the data record is called the seek time. Then the portion of the disk on which the data record is stored must rotate until it is immediately under (or over) the read-write head. Then the record, which is of arbitrary size, must be made to spin by the read-write head. This is called transmission time. Because each of these operations involves mechanical movement, the total time a disk access takes is often an appreciable fraction of a second Average Seek Time (ms) Average Rotational latency (ms) Maxtor DiamondMax Plus 9 (High-end desktop) 9.3 4.2 Model (Environment) WD Caviar (High-end desktop) 8.9 4.2 Toshiba MK8025GAS (Laptop) 12.0 7.14 WD Raptor (Enterprise) 5.2 2.99 Cheetah 15K.3 (Enterprise) 3.6 2.0 Figure 12.2 | Hard disk track-to-track seek times and latency times. 14, 15, 16, 17, 18 12.4 Why Disk Scheduling Is Necessary 531 Figure 12.3 | Schematic top view of a disk surface. (several milliseconds at least). During this same period of time, a processor can execute tens or even hundreds of millions of instructions. Self Review 1. One focus of research in the field of storage technology is on increasing a hard disk's areal density (i.e., amount of data per unit area). How does increasing the areal density alone affect disk access times? 2. (T/F) Rotational latency is identical for every disk access. Ans: 1) As the amount of data per unit area increases, then the amount of data contained in each track increases. If a particular data record is located on a single track, the transmission time decreases compared to that for a lower areal density because more data can be read per unit time. 2) False. The rotational latency depends on the location of the beginning of the requested sector relative to the position of the arm. 12.4 Why Disk Scheduling Is Necessary Many processes can generate requests for reading and writing data on a disk simultaneously. Because these processes sometimes make requests faster than they can be serviced by the disk, waiting lines or queues build up to hold disk requests. Some early computing systems simply serviced these requests on a first-come-first-served 532 Disk Performance Optimization Figure 12.4 | Components of a disk access. (FCFS) basis, in which the earliest arriving request is serviced first.19 FCFS is a fair method of allocating service, but when the request rate (i.e., the load) becomes heavy, can be long waiting times. FCFS exhibits a random seek pattern in which successive requests can cause time-consuming seeks from the innermost to the outermost cylinders. To reduce the time spent seeking records, it seems reasonable to order the request queue in some manner other than FCFS. 20 This process, called disk scheduling, can significantly improve throughput. 21 Disk scheduling involves a careful examination of pending requests to determine the most efficient way to service them. A disk scheduler examines the positional relationships among waiting requests, then reorders the queue so that the requests will be serviced with minimum mechanical motion. Because FCFS does not reorder requests, it is considered by many to be the simplest disk scheduling algorithm. The two most common types of scheduling are seek optimization and rotational optimization. Because seek times tend to be greater than latency times, most scheduling algorithms concentrate on minimizing total seek time for a set of requests. As the gap between rotational latency and seek times narrows, minimizing the rotational latency of a set of requests can also improve overall system performance, especially under heavy loads. 12.5 Disk Scheduling Strategies 533 Self Review 1. Indicate which type of disk scheduling algorithm is most appropriate in the following scenarios: seek time is significantly greater than latency time, seek time and latency time are nearly equal, seek time is significantly shorter than latency time. 2. Which characteristics of disk geometry are most important to seek optimization and rotational optimization? Ans: 1) When seek time is significantly greater than latency time, access times are most affected by seek operations, so the system should implement seek optimization. When seek times and latency times are nearly equal, the system can benefit from combining both seek and rotational optimization techniques. If seek time is much shorter than latency time, then access times are most affected by latency time, so the system should concentrate on rotational optimization. Because today's processors are so fast, both forms of optimization should be employed to improve overall performance. 2) Cylinder locations are most important when optimizing seek times, and sector locations are most important when optimizing latency times. 12.5 Disk Scheduling Strategies A system's disk scheduling strategy depends on the system objectives, but most strategies are evaluated by the following criteria • throughput—the number of requests serviced per unit time • mean response time—the average time spent waiting for a request to be serviced • variance of response times—& measure of the predictability of response times. Each disk request should be serviced within an acceptable time period (i.e., the strategy should prevent indefinite postponement). Clearly, a scheduling policy should attempt to maximize throughput and minimize the mean response time. Many scheduling policies attempt to accomplish these goals by minimizing the time spent performing lengthy seeks. When throughput and mean response time are optimized, average system performance improves, nut individual requests may be delayed. Variance measures how individual requests are serviced relative to average system performance. The smaller the variance, the more likely it is that most disk requests are serviced after waiting for a similar amount of time. Therefore, variance can be seen as a measure of fairness and of predictability. We desire a scheduling policy that minimizes variance (or at least keeps it at reasonable levels) to avoid erratic service times. In a business-critical system, such as a Web server, a high variance of response times could result in loss of sales if, for example, users' requests to purchase products were indefinitely postponed or suffer lengthy waits. In missioncritical systems, the result of such a delay could be catastrophic. The following sections describe several common scheduling policies. We use the set of disk requests in Fig. 12.5 to demonstrate the result of each policy on an arbitrary series of requests. The arbitrary series of requests is intended to demonstrate how each policy orders disk requests, it does not necessarily indicate the relative performance of each policy in a real system. In the examples that follow, we 534 Disk Performance Optimization Figure 12.5 | Disk request pattern. assume that the disk contains 100 cylinders, numbered 0-99, and that the read/write head is initially located at cylinder 63, unless stated otherwise. For simplicity, we also assume that the disk scheduler has already determined the cylinder number corresponding to each request. Self Review 1. Why is minimizing the variance of response times an important goal in disk scheduling? 2. We mentioned that FCFS is a fair scheduling algorithm. Which of the criteria mentioned in this section most directly relates to fairness? Ans: 1) Otherwise the system could experience erratic response times. In a business-critical system, this could cause a company to lose customers; in a mission-critical system, it could put people's lives at risk. 2) FCFS is fair in the sense that arriving requests cannot get in line ahead of waiting requests. This helps reduce the variance of response times. 12.5 Disk Scheduling Strategies 535 12.5.1 First-Come-First-Served (FCFS) Disk Scheduling FCFS scheduling uses a FIFO queue so that requests are serviced in the order in which they arrive.22'23-24 FCFS is fair in the sense that a request's position in the queue is unaffected by arriving requests. This ensures that no request can be indefinitely postponed, but it also means that FCFS might perform a lengthy seek operation to service the next waiting request, even if another request in the queue is closer and can be serviced faster. Although this technique incurs low executiontime overhead, it can result in low throughput due to lengthy seeks. When requests are uniformly distributed over the disk surfaces, FCFS scheduling leads to a random seek pattern because it ignores positional relationships among the pending requests (Fig. 12.6).This is acceptable when the load on a disk is light. However, as the load grows, FCFS tends to saturate (i.e., overwhelm) the device, and response times become large. The random seek pattern of FCFS results in low variance (because arriving requests cannot get ahead of waiting requests), but this is of little solace to the request sitting at the back of the disk queue while the disk arm rambles around in a torrid "disk dance." Self Review 1. One measure of fairness is a low variance in response times. In this sense, is FCFS disk scheduling fair? 2. Can indefinite postponement occur with FCFS disk scheduling? Explain. Ans: 1) Yes, in this sense it is fair, but because no attempt is made to minimize response times, average response times tend to be longer than necessary. 2) No. Indefinite postponement cannot occur because arriving requests can never be placed ahead of requests in the queue. Figure 12.6 | Seek pattern under the FCFS strategy. 536 Disk Performance Optimization 12.5.2 Shortest-Seek-Time-First (SSTF) Disk Scheduling Shortest-seek-time-first (SSTF) scheduling next services the request that is closest to the read-write head's current cylinder (and thus incurs the shortest seek time). even if that is not the first one in the queue.25, 26, 27, 28 SSTF does not ensure fairness and can lead to indefinite postponement because its seek pattern tends to be highly localized, which can lead to poor response times for requests to the innermost and outermost tracks (Fig. 12.7). By reducing average seek times, SSTF achieves higher throughput rates than FCFS, and mean response times tend to be lower for moderate loads. One significant drawback is that it leads to higher variances of response times because of the discrimination against the outermost and innermost tracks; in the extreme, starvation (see Chapter 7) of requests far from the read/write heads could occur if new arrivals tend to be clustered near the middle cylinders. The high variance is acceptable in batchprocessing systems, where throughput and mean response times are more important goals. However, SSTF is inappropriate for interactive systems, where the system must ensure that each user experiences prompt, predictable response times (see the Anecdote, Every Problem Has a Solution and Every Solution Has a Problem). Anecdote Every Problem Has a Solution and Every Solution Has a Problem This true story happened to a neighbor w h o is in no way involved w i t h the computer industry. Her husband left for work shortly before she did and he forgot to close the garage door that day. When she was leaving her home to go to work, she walked into her garage and found a skunk had pushed over a garbage pail and was sitting next to the door to her car merrily chomping on the garbage. She called the town's animal control department, but they had a backlog and said they would not be able to come out for a few hours. They told her, though, that skunks love bacon, so she should fry up a batch of bacon and t h r o w it out on her front lawn. The skunk would smell the bacon and go out on the front lawn to eat it. She could then hop in her car, pull out of the garage, close the garage door with her remote control and head off to work. She followed that advice. But much to her chagrin, when she threw the bacon onto the front lawn, another skunk came out of the woods to eat the bacon, while the first skunk continued eating the garbage in her garage. She got to work a few hours late that day. Lesson to operating systems designers: Always try to anticipate the consequences of your design choices. 12.5 Disk Scheduling Strategies Figure 1 2 . 7 | Seek pattern under the SSTF strategy. Self Review 1. To which scheduling algorithm from Chapter 8 is SSTF most similar? How do they differ? 2. Compare the overhead incurred by SSTF and FCFS. Ans: 1) SSTF is most similar to the shortest-process-first (SPF) strategy. Unlike SPF, SSTF does not require future knowledge (such as runtime-to-completion)—the request's location is sufficient information to determine which request to service next. 2) Both FCFS and SSTF must maintain a queue of waiting requests in memory. Because FCFS maintains a simple FIFO queue, the execution time required to determine which request to service next is small and is unaffected by the number of requests in the queue. SSTF incurs overhead by either maintaining a sorted list (or lists) of requests or by searching the entire request queue for the appropriate request to service. In this case, the overhead incurred by SSTF is proportional to the number of requests in the queue (whereas the overhead of FCFS is constant—simply the amount of time it takes to insert entries at the back of the queue and remove them from the front). 12.5.3 SCAN Disk Scheduling Denning developed the SCAN disk scheduling strategy to reduce the unfairness and variance of response times exhibited by SSTF.29 SCAN chooses the request that requires the shortest seek distance in a preferred direction (Fig. 12.8). Thus, if the preferred direction is currently outward, the SCAN strategy chooses the shortest seek distance in the outward direction. SCAN does not change its preferred direction until it reaches the outermost cylinder or the innermost cylinder. In this sense, it is called the elevator algorithm, because an elevator continues in one direction servicing requests before reversing direction. SCAN behaves much like SSTF in terms of high throughput and good mean response times. However, because SCAN ensures that all requests in a given direction will be serviced before the requests in the opposite direction, it offers a lower variance of response times than SSTF. SCAN, like SSTF, is a cylinder-oriented strategy. 537 538 Disk Performance Optimization Figure 12.8 | Seek pattern under the SCAN strategy. Because the read-write heads oscillate between opposite ends of each platter in SCAN, the outer tracks are visited less often than the midrange tracks, but generally more often than they would be using SSTF. Because arriving requests can be serviced before waiting requests, both SSTF and SCAN can suffer indefinite postponement. Self Review 1. One limitation of the SCAN algorithm is that it might perform unnecessary seek operations. Indicate where this occurs in Fig. 12.8. 2. Can requests be indefinitely postponed under SCAN? Ans: 1) The unnecessary seek operation occurs after servicing the request to cylinder 8. Because there are no further requests in the preferred direction, it would be more efficient to change direction after servicing that request. We examine a modification to the SCAN strategy that addresses this limitation in Section 12.5.6, LOOK and C-LOOK Disk Scheduling. 2) One could imagine a scenario in which processes issue continuous requests to the same cylinder, so that the read/write head becomes "stuck" on that cylinder. 12.5.4 C-SCAN Disk Scheduling In the Circular SCAN (C-SCAN) modification to the SCAN disk scheduling strategy, the arm moves from the outer cylinder to the inner cylinder, servicing requests on a shortest-seek basis (Fig. 12.9). When the arm has completed its inward sweep, it jumps (without servicing

requests) to the outermost cylinder, then resumes its inward sweep, processing requests. C-SCAN maintains high levels of throughput while further limiting variance of response times by avoiding discrimination against the innermost and outermost cylinders.30-31 As in SCAN, requests in C-SCAN can be indefinitely postponed if requests to the same cylinder continuously arrive (although this is less likely than with SCAN or SSTF). In the sections that follow, we discuss modifications to SCAN that address this problem. 12.5 Disk Scheduling Strategies 539 Figure 12.9 | Seek pattern under the C-SCAN strategy. Simulation results in the literature indicate that the best disk scheduling policy might operate in two stages.32 Under a light load, the SCAN policy is best. Under medium to heavy loads, C-SCAN and other adaptations to the SCAN policy yield better results. C-SCAN with rotational optimization handles heavy loading conditions effectively.33 Sel Review 1. Which criteria of disk scheduling strategies does C-SCAN improve upon? 2. (T/F) C-SCAN incurs essentially the same execution-time overhead as SCAN. Ans: 1) C-SCAN reduces variance of response times compared to SCAN because it is less biased against the outermost and innermost cylinders. 2) False. C-SCAN incurs overhead over SCAN when it finishes its inward sweep and skips requests while moving the head to the outermost cylinder. 12.5.5 FSCAN and N-Step SCAN Disk Scheduling The FSCAN and TV-Step SCAN modifications to the SCAN strategy eliminate the possibility of indefinitely postponing requests. 34, 35 FSCAN uses the SCAN strategy to service only those requests waiting when a particular sweep begins (the "F" stands for "freezing" the request queue at a certain time). Requests arriving during a sweep are grouped together and ordered for optimum service during the return sweep (Fig. 12.10). TV-Step SCAN services the first n requests in the queue using the SCAN strategy. When the sweep is complete, the next n requests are serviced. Arriving requests are placed at the end of the request queue (Fig. 12.11). TV-Step SCAN can be tuned by varying the value for n. When n-1, N-Step SCAN degenerates to FCFS. As n approaches infinity, TV-Step SCAN degenerates to SCAN. 540 Disk Performance Optimization Figure 12.10 | Seek pattern under the FSCAN strategy. Figure 12.11 | Seek pattern under the N-Step SCAN strategy (n = 3). FSCAN and TV-Step SCAN offer good performance due to high throughput and low mean response times. Because they prevent indefinite postponement, their distinguishing characteristic is a lower variance of response times than with SSTF and SCAN, especially when requests would be indefinitely postponed under the latter two strategies. 12.5 Disk Scheduling Strategies 541 Sel Review 1. Explain how FSCAN can lead to lower throughput than SCAN. 2. Compare and contrast FSCAN and JV-Step SCAN. Ans: 1) If a request to a cylinder in the preferred direction arrives after FSCAN freezes the queue, the disk head will pass by the cylinder without servicing the request until the next pass. 2) Both strategies use SCAN to service a portion of the request queue to prevent indefinite postponement. FSCAN services all the requests in the queue before beginning a sweep in a new preferred direction; TV-Step SCAN services only the next n requests in the queue. 12.5.6 LOOK and C-LOOK Disk Scheduling The LOOK variation of the SCAN strategy "looks" ahead to the end of the current sweep to determine the next request to service. If there are no more requests in the current direction, LOOK changes the preferred direction and begins the next sweep (Fig. 12.12). In this sense, it is appropriate to call this the elevator algorithm, because an elevator continues in one direction until it reaches the last request in that direction, then reverses direction. This strategy eliminates unnecessary seek operations exhibited by other variations of the SCAN strategy (compare the left side of Fig. 12.12 to the corresponding location in Fig. 12.8). Circular LOOK (C-LOOK) variation of the LOOK strategy uses the same technique as C-SCAN to reduce the bias against requests located at the extreme ends of the platters. When there are no more requests on a current inward sweep, the read/write head moves to the request closest to the outer cylinder (without servicing requests in between) and begins the next sweep. The C-LOOK policy is characterized by a potentially lower variance of response times compared to LOOK and nigh throughput (although generally lower than LOOK). 36 Figure 12.13 summarizes each of the disk scheduling strategies we have discussed. Figure 12.12 | Seek pattern under the LOOK strategy. 542 Disk Performance Optimization Strategy Description FCFS Services requests in the order in which they arrive. SSTF Services the request that results in the shortest seek distance first. SCAN Head sweeps back and forth across the disk, servicing requests according to SSTF in a preferred direction. C-SCAN Head sweeps inward across the disk, servicing requests according to SSTF in the preferred (inward) direction. Upon reaching the innermost track, the head jumps to the outermost track and resumes servicing requests on the next inward pass. FSCAN Requests are serviced the same as SCAN, except newly arriving requests are postponed until the next sweep. Avoids indefinite postponement. N-Step SCAN Services requests as in FSCAN, but services only n requests per sweep. Avoids indefinite postponement. LOOK Same as SCAN except the head changes direction upon reaching the last request in the preferred direction. C-LOOK Same as C-SCAN except the head stops after servicing the last request in the preferred direction, then services the request to the cylinder nearest the opposite side of the disk. Figure 12.13 | Seek optimization strategies summary. Self Review 1. Why does C-LOOK typically exhibit throughput lower than that of LOOK? 2. Under C-LOOK, in what order would the requests presented in Fig. 12.5 be serviced? Ans: 1) The time spent seeking from one end of the platter to the furthest request from the position of the read/write head at the end of the sweep increases the mean response time, which decreases throughput. 2) 52, 47, 33, 8, 99, 75, 74, 72 12.6 Rotational Optimization Because the dominant component of access time in early hard drives was seek time, research focused on seek optimization. However, today's hard disks exhibit seek times and average latencies that are of the same order of magnitude, meaning that rotational optimization can often improve performance.37 Processes that access data sequentially tend to access entire tracks of data and thus do not benefit much from rotational optimization. However, when there are numerous requests to small pieces of data randomly distributed throughout the disk's cylinders, rotational optimization can improve performance significantly. In this section, we discuss how to combine seek and rotational optimization strategies to achieve maximum performance. 12.6.1 SLTF Scheduling Once the disk arm arrives at a particular cylinder, there might be many requests pending on the various tracks of that cylinder. The shortest-latency-time-first 12.6 (SLTF) strategy examines all of these requests and services the one with the shortest rotational delay first (Fig. 12.14). This strategy has been shown to be close to the theoretical optimum and is relatively easy to implement. 38 Rotational optimization is sometimes referred to as sector queuing; requests are queued by sector position around the disk, and the nearest sectors are serviced first. Self Review 1. Why is sector queuing easy to implement? 2. Is rotational optimization appropriate for today's hard disks? Why? Ans: 1) Sector queuing determines the best access pattern for sectors on a track. Assuming that sectors are at fixed locations and that the disk can spin only in one direction, sector queuing simply is a sorting problem. 2) Yes, rotational optimization is appropriate. Today's hard disks exhibit seek times and average latencies that are of the same order of magnitude. 12.6.2 SPTF and SATF Scheduling The shortest-positioning-time-first (SPTF) strategy next services the request that requires the shortest positioning time, which is the sum of the seek time and rotational latency time. Like SSTF, SPTF results in high throughput and a low mean Figure 12.14 | SLTF scheduling. The requests will be serviced in the indicated order regardless of the order in which they arrived. Rotational Optimization 543 544 Disk Performance Optimization response time, and it can also indefinitely postpone requests to the innermost and outermost cylinders.39 A variation of SPTF is the shortest-access-time-first (SATF) strategy, which next services the request that requires the shortest access time (i.e., positioning time plus transmission time). SATF exhibits higher throughput than SPTF, but large requests can be indefinitely postponed by a series of smaller requests, and requests to the innermost or outermost cylinders can be indefinitely postponed by requests to midrange cylinders. Both SPTF and SATF can improve performance by implementing the "look-ahead" mechanisms described in Section 12.5.6.40 Figure 12.15 demonstrates the difference between SPTF and SATF. In Fig. 12.15(a), the disk receives two requests for data records of the same size, A and B, that are located on adjacent cylinders. Data record A is located in the same cylinder as the read/write head, but approximately one-half rotation away from the read/write head. Data record B is near the read/write head but is located on an adjacent cylinder. In this case, the transmission time is identical for A and B because A and B are the same size, so SATF reduces to SPTF. Thus, for this particular disk, both SATF and SPTF would service the request for data record B first, because the disk takes less time to perform a single-cylinder seek operation than to rotate the platter 180 degrees. If the seek time is adequately small, the disk can service the request for B, then reposition the head to service the request for A within one disk rotation. On the contrary, SSTF would service A first, requiring more than one revolution of the platter to service both requests. Thus, SATF and SPTF can increase throughput compared to SSTF Now consider that data record B consumes an entire track whereas A is stored in one sector, as shown in Fig. 12.15(b). Assume that the location of the first byte of Figure 12.15 | SPTF (a) and SATF (b) disk scheduling examples. 12.6 each record is the same as in the previous example. In this case, SPTF would service requests in exactly the same order as in the previous example, because the positioning times are the same. However, SATF would service request A first, because request A requires one half of a rotation plus nominal transmission time, whereas request B requires a nominal rotational latency time plus a transmission time equal to the time required for the platter to complete one revolution. SPTF and SATF require knowledge of disk characteristics including latency time, track-to-track seek times and relative locations of sectors. Unfortunately, many of today's hard drives expose misleading geometry. For example, many disks hide error-checking and correcting data from the operating system (so that it cannot be unintentionally or maliciously modified), so consecutively numbered sectors may not precisely correspond to contiguous physical locations on disk. On some systems, when bad sectors are detected, alternate ones are assigned. For example, if sector 15 becomes unusable, the disk might send its requests to a reserved sector other than 14 or 16. These alternate sectors can be widely dispersed over the disk surfaces, causing seeks where none might actually be expected. These features, such as using alternate sectors, which improve data integrity, tend to counteract efforts to improve disk performance using scheduling strategies, because the disk provides the operating system with incomplete or misleading positional information.41 Although some architectures and hard disks provide commands to retrieve the real geometry of the disk, this feature is not supported by all disks. Other hard disk architectural features complicate disk scheduling strategies; for more information, see the links provided in the Web Resources section at the end of the chapter. A straightforward performance enhancement technique is to decrease rotational latency by increasing the rotational speed of the disk. However, designers have encountered significant problems when increasing the rotations per minute RPM) of a hard disk. Faster-rotating disks consume greater amounts of energy (a key concern on laptop computers with limited battery power), radiate more heat, make more noise and require more costly mechanical and electrical mechanisms to control them. 42 Given these and other problems, RPM speed has increased only modestly (only a few percent a year over the past decade) to approximately 5,400 to 7.200 RPM for PCs and 10,000-15,000 RPM for servers and high-end machines.43 Self Review 1. What is the difference between the SPTF and SATF algorithms? Why would a designer choose one over the other? 2. What factors prevent operating systems from obtaining accurate knowledge of hard disk geometries? Ans: 1) SPTF services the request that it can begin accessing within the shortest period of time. SATF services the request that it can complete accessing within the shortest time. Although SATF has higher throughput, it favors requests to small amounts of data over requests to large amounts of data. Thus, SATF is more appropriate for systems that issue Relational Optimization 545 546 Disk Performance Optimization many requests for small data records. SPTF is more appropriate for systems that generate requests for both large and small data records, because it does not consider the size of a request. 2) Some types of hard disk hide information, such as the location of error-protection data and bad sectors. 12.7 System Considerations When is disk scheduling useful? When might it degrade performance? These questions must be answered in the context of the overall system. The sections that follow discuss several considerations that might influence a designer's decisions. Storage as Limiting Resource When disk storage proves to be a bottleneck, some designers recommend adding more disks to the system (see the Operating Systems Thinking feature, Saturation and Bottlenecks). This does not always solve the problem, because the bottleneck could be caused by a large request load on a relatively small number of disks. When this situation is detected, disk scheduling may be used as a means of improving performance and eliminating the bottleneck. System Load Disk scheduling might not be useful in a batch-processing system with a relatively low degree of multiprogramming. Scheduling becomes more effective as the randomness of multiprogramming increases, which increases the system load and leads to erratic disk request patterns. For example, file servers in local area networks can receive requests from hundreds of users, which normally results in the kind of ran- Operating Systems Thinking Saturation and Bottlenecks Operating systems often manage hardware and software configurations. When a system is performing poorly, you might be inclined to speak of the problem "in the whole"—the "whole syst e m " is performing poorly. In fact, its often the case that only one or a few resources have become saturated, i.e., they have reached their capacity and cannot service requests any faster. Such resources are called bottlenecks. Locating bottlenecks and dealing w i t h them by adding resources in only one or a few areas of the system, could result in overall system improvement at a relatively modest cost. Systems should be built in ways that make it easy to locate bottlenecks. Chapter 14 presents a thorough treatment of performance issues, including saturation and bottlenecks. 12.7 System Considerations 547 dom request patterns that are best serviced using disk scheduling. Similarly, online transaction processing (OLTP) systems, such as Web and database servers, typically receive many disk requests to randomly distributed locations containing small amounts of data (e.g., HTML files and database records). Research has shown that disk scheduling algorithms such as C-LOOK and SATF (with look-ahead) can improve performance in this type of environment.44-45 Much of the analytical work in the literature assumes that request distributions are uniform. Conclusions based on the assumption of uniformity may be invalid on systems whose request distributions are not uniformly distributed over the disk surfaces. Nonuniform request distributions are common in certain situations, and their consequences have been investigated.46 In one study, Lynch determined that the vast majority of disk references are to the same cylinder as the immediately preceding reference.47 One common cause of highly localized nonuniform request distributions is the use of large sequential files on dedicated disks. When an operating system allocates space for the adjacent records of a user's sequential file, it usually places adjacent records on the same track. When a track is full, additional records are placed on adjacent tracks within the same cylinder; when a cylinder is full, additional records are placed on adjacent cylinders. Thus requests for successive records in a sequential file often cause no seeking at all. When needed, seeks are short, because they are usually to immediately adjacent cylinders. Obviously, an FCFS disk scheduling policy would be adequate in this situation. In fact, overhead incurred in more complex scheduling strategies might actually result in degraded performance. File Organization Techniques As we discuss in Chapter 13, File and Database Systems, sophisticated file organization techniques can cause a proliferation of requests with large seek times. In some cases, file data retrieval may involve reference to a master index, reference to a cylinder index, and then location of the actual record, a process that could incur several seek delays. Because the master index and cylinder index are normally stored on disk (but separate from the main data area), these seeks can be costly. Such file organization techniques are convenient for the applications designer but can degrade performance. Self Review 1. Why do many systems exhibit nonuniform request distributions? 2. Why does disk scheduling become more effective as system load increases? Ans: 1) File data is often stored and accessed sequentially, so a request to a particular cylinder is likely to be followed by a request to the same cylinder, or to an adjacent cylinder.2) As system load increases, request patterns tend to become more random, which could lead to 548 Disk Performance Optimization substantial seek activity. Disk scheduling improves performance by reducing the number of wasteful seek operations. 12.8 Caching and Buffering Many systems maintain a disk cache buffer, which is a region of main memory that the operating system reserves for disk data. In one context, the reserved memory acts as a cache, allowing processes quick access to data that would otherwise need to be fetched from disk. The reserved memory also acts as a buffer, allowing the operating system to delay writing modified data until the disk experiences a light load or until the disk head is in a favorable position to improve I/O performance. For example, an operating system may delay writing modified data to disk to allow time for multiple requests to contiguous locations to enqueue, so that they can be serviced with one I/O request. The disk cache buffer presents several challenges to operating system designers. Because the size of the disk cache must be limited to allow enough memory for active processes, the designer must implement some replacement strategy. The cache-replacement question is similar to the page-replacement question, and designers use many of the same heuristics. Most commonly, designers choose a strategy that replaces the least-recently used item in the disk cache buffer. A second concern arises because disk caching can lead to inconsistencies. Disk cache buffers are maintained in volatile memory, so if the system fails or loses power while modified data is in the cache buffer, those changes are lost. To safeguard data against such problems, the contents of the disk cache buffer are periodically flushed to the hard disk; this reduces the probability of data loss if the system crashes. A system that employs write-back caching does not write modified data to disk immediately. Instead, the cache is written to disk periodically, enabling the operating system to batch multiple I/Os that are serviced using a single request, which can improve system performance. A system that employs write-through caching writes data both to the disk cache buffer and to disk each time cached data is modified. This technique prevents the system from batching requests, but reduces the possibility of inconsistent data in the event of a system crash.48 Many of today's hard disk drives maintain an independent high-speed buffer cache (often called an on-board cache) of several megabytes. 49 If requested data is stored in the on-board cache, the hard drive can deliver the data at or near the speed of main memory. Additionally, some hard disk interface controllers (e.g., SCSI and RAID controllers) maintain their own buffer caches separate from main memory. When requested data is not located in the main memory buffer cache, onboard buffer caches improve I/O performance by servicing requests without performing relatively slow mechanical operations. 50 However, one study showed that disks that include on-board caches typically use replacement strategies that adapt poorly to random request patterns, leading to suboptimal performance. 51 It is also likely that on-board buffer caches and main memory buffer caches will contain the same data, which leads to inefficient resource utilization. 12.9 Other Disk Performance Techniques 549 Self Review 1. What are the trade-offs when selecting the size of the system's disk cache buffer? 2. State when write-back caching and write-through caching are appropriate, and why. Ans: 1) A small disk cache buffer allows enough memory for active processes, but is likely to contain only a small portion of requested data, so many requests must be serviced using the disk. A small cache buffer also limits the number of write requests that can be buffered simultaneously, which means that data will need to be flushed frequently. A large cache buffer reduces the number of I/O operations the system performs, but can reduce the memory available to store process instructions and data, which could lead to thrashing. 2) Write-back caching enables the system to batch modified data to reduce the number of disk requests, but does not prevent loss of buffered data in the event of a power or system failure. Thus, write-back caching is appropriate for systems where performance is more important than reliability (e.g., some supercomputers). Because write-through caching immediately sends modified data to disk, it is appropriate for systems that cannot tolerate loss of data (e.g., database systems). 12.9 Other Disk Performance Techniques We have considered optimizing the performance of rotational storage devices using scheduling policies and optimizing system architecture. In this section we analyze several other disk performance optimization techniques. As files and records are added and deleted, a disk's data tends to become dispersed throughout the disk, or fragmented. Even sequential files, which one would expect to exhibit low access latency, become severely fragmented, which increases access time. Many operating systems provide defragmentation (or disk reorganization) programs that can be used periodically to reorganize files. This allows the consecutive records of sequential files to be placed contiguously on the disk. Systems that use noncircular variations of the SCAN strategy tend to visit the midrange cylinders more frequently than the outermost ones. For these systems, frequently referenced items can be placed on the midrange tracks to reduce average access times. Also, operating systems can place files that are likely to be modified near free space to reduce future fragmentation—as the files grow, new data can be placed in the adjacent free space instead of elsewhere on disk. Some operating systems allow users to partition a disk into separate areas. Files are then restricted to these partitions, so fragmentation is reduced. 52 However, partitions can lead to wasted memory similar to internal fragmentation in paged virtual memory systems. Some systems use data compression techniques to reduce the amount of space required by information on disk (see the Operating Systems Thinking feature, Compression and Decompression). Data compression decreases the size of a record by replacing common bit patterns with shorter ones. Therefore, compressed data consumes less space without losing information. This can ultimately reduce the number of seeks, latency times and transmission times. However, it can require substantial processor time to compress the data and later to decompress it to make it available to applications. Systems that need to access certain information quickly benefit from placing multiple copies of that data at different positions on the disk (see the Operating 550 Disk Performance Optimization Systems Thinking feature, Redundancy). This can substantially reduce seek and rotation times, but the redundant copies can consume a significant portion of the disk. This technique is more useful for read-only data, or for data that changes nominally. Frequent changes degrade the performance of the scheme, because each of the copies must be updated regularly; another danger is that a system crash could leave the multiple copies in an inconsistent state. In a multiple-disk system in which only one disk request may be serviced at a time, performance can be improved by replicating frequently referenced data on separate drives, which enables higher throughput. Some RAID techniques, discussed in Section 12.10, implement this concept. Additionally, record blocking can yield significant performance improvements. When contiguous records are read or written as a single block, only a single seek is required; if these records are read or written individually, then one seek per record could be required. Systems that monitor disk access attempts can try to keep frequently accessed data in a favorable position in the memory hierarchy (i.e., in main or cache memory) while transferring infrequently referenced data to slower storage (such as a hard disk or compact disk). This yields an overall performance improvement, but it might provide poor response to users of infrequently accessed data. This is unacceptable if the infrequently accessed data belongs to high-priority applications. It might also be helpful to dedicate a disk to a single high-performance application. In many environments, there are often short periods when there are no waiting requests for the disk to service and the disk arm sits idle, waiting for the next Operating Systems Thinking Compression and Decompression An excellent use of abundant, cheap processor power is compression and decompression of data. On hard disks, this helps reduce the space required to store files. It is also important in network transmissions. The world's networking infrastructure bandwidths tend to increase much slower than processor power does. The capital investment in networking infrastructure, especially transmission lines, is enormous. It simply is not feasible to keep upgrading the entire world's networking infrastructure every time a faster transmission technology becomes available. This means that the ratio of processor speeds to transmission speeds will tend to increase over time, with processor power remaining rela- tively cheap compared to transmission bandwidth. Compression enables messages to be shrunk for transmission over relatively slow networks, essentially increasing network throughput. Compressed messages must be decompressed at the receiving end to be usable. Operating systems will increasingly include compression and decompression mechanisms. 12.9 Other Disk Performance Techniques 551 request. 53 If the disk arm is currently at one edge of the disk, it is likely that a long seek will be required to process the next request. By contrast, if the disk arm is in the center of the disk, or at some hot spot of disk activity, the average seek time will be less. Moving the disk arm to a location that will minimize the next seek is known as disk arm anticipation.54 Disk arm anticipation can be useful in environments where the disk request patterns of processes exhibit locality. For best results, the arm should move to the hot spot of disk activity (rather than simply the center of the disk). When processes issue requests to sequential locations on disk, however, moving the head to the center of the disk after servicing each request can lead to excessive seek activity. In this case, disk arm anticipation might degrade performance. 55 Self Review 1. How do defragmentation and disk reorganization improve performance? 2. Why might data compression techniques become more practical in the future? 3. Under what conditions is it inappropriate to move the read/write head to the center of the disk during idle periods? 4. Why does multiprogramming complicate efforts to anticipate the next request? Ans: 1) Defragmentation places file data in contiguous blocks on disk, which improves access times by reducing seek activity when accessing sequential data. Disk reorganization places frequently or heavily used data in favorable locations on disk (e.g., midrange tracks for noncircular scheduling strategies) to reduce average seek times. 2) There is a growing gap between processor speed and disk speed. The reduced access time due to data compression Operating Systems Thinking Redundancy We will see many examples of redundancy being employed in operating systems for a variety of reasons. A common use of redundancy is creating backups to ensure that if one copy of information is lost, it can be restored. A multiprocessing system can have a pool of identical processors available to assign to processes and threads as needed. Such redundancy has several advantages. Although the system could still function w i t h only a single processor, having the extra processors yields better performance because the processors can all work in parallel. It is also effective for fault tolerance—if one processor fails, the system can continue operating. RAID reduces access times to data on disks by placing redundant copies of that data on separate disks that may function in parallel. We can also place redundant copies of the data on differ- ent regions of the same disk, so that we can minimize the movement of the read/write head and the amount of rotational movement of the disk before the data becomes accessible, thus increasing performance. Redundancy, of course, has its price. The resources costs money and the hardware and the software to support them can become more complex. This is yet another example of trade-offs in operating systems. 552 Disk Performance Optimization might outweigh the overhead incurred by compressing and decompressing data. 3) If programs generally exhibit spatial locality, and the area of locality is not at the center of the disk, then moving the head to the center of the disk during each idle period requires a wasteful seek to return to the disk's hot spot when requests resume. 4) A multiprogramrned system can service requests from multiple concurrent processes, which may lead to several hot spots on the disk. In this case, it is difficult to determine which, if any, hot spot the read/write head should move to when it is idle. 12.10 Redundant Arrays of Independent Disk (RAID) RAID (Redundant Arrays of Independent Disks) is a family of techniques that use multiple disks (called an array of disks) that are organized to provide high performance and/or reliability. RAID was originally proposed by Patterson, Gibson and Katz; in their paper, the "I" in RAID denotes "Inexpensive," but this has been changed to "Independent" because many RAID systems employ expensive, highperformance disks.56 Patterson et al. recognized that processor speed, main memory size and secondary storage size were increasing rapidly while I/O transfer rates (particularly in hard disks) were growing at a much slower pace. Computer systems were becoming increasingly I/O bound—they could not service I/O requests as quickly as they were generated, and could not transfer data as quickly as it was capable of being consumed. To improve throughput and transfer rates, the authors recommended creating arrays of disks that could be accessed simultaneously.57 12.10.1 RAID Overview In their original paper, Patterson et al. proposed five different organizations, or levels, of disk arrays.58 Each RAID level is characterized by data striping and redundancy. Data striping entails dividing storage into fixed-size blocks called strips. Contiguous strips of a file are typically placed on separate disks so that requests for file data can be serviced using multiple disks at once, which improves access times. A stripe consists of the set of strips at the same location on each disk in the array. In Fig. 12.16, a portion of a file is divided into four strips of identical length, each on a different disk in the array. Because each strip is placed at the same location on each disk, the four strips form one stripe. Striping distributes a system's data across multiple disks, which enables higher throughput than a single-disk system because data can be accessed from multiple disks simultaneously. When selecting a strip size, the system designer should consider the average size of disk requests. Smaller strip sizes, also called fine-grained strips, tend to spread file data across several disks. Fine-grained strips can reduce each request's access time and increase transfer rates, because multiple disks simultaneously retrieve portions of the requested data. While these disks service a request, they cannot be used to service other requests in the system's request queue. Large strip sizes, also called coarse-grained strips, enable some files to fit entirely within one strip. In this case, some requests can be serviced by only a por- 12.10 Redundant Arrays of Independent Disks (RAID) 553 Figure 12.16 | Strips and stripe created from a single file in RAID systems. tion of the disks in the array, so it is more likely that multiple requests can be serviced simultaneously. However, smaller requests are serviced by one disk at a time, which reduces transfer rates for individual requests compared to fine-grained strips. Systems such as Web servers and databases, which typically access multiple small records simultaneously, benefit from coarse-grained strips, because several 1/ O operations can occur concurrently. Systems such as supercomputers that require fast access to a small number of records benefit from fine-grained strips, which provide high transfer rates for individual requests. 59 The increased transfer rate provided by RAID systems have a price. As the number of disks in the array increases, so does the likelihood of disk failure. For example, if the mean-time-to-failure (MTTF) of a single disk is 200,000 hours (roughly 23 years), the MTTF for a 100-disk array would be 2,000 hours (about three months). 60 If one disk in an array fails, any files containing strips on that disk are lost. In business-critical and mission-critical systems, this loss of data could be catastrophic (see the Operating Systems Thinking feature, Mission-Critical Systems). As a result, most RAID systems store information that enables the system to recover from errors, a technique called redundancy. RAID systems use redundancy to provide fault tolerance (i.e., to sustain failures without loss of data; see the Operating Systems Thinking feature, Fault Tolerance). A straightforward way to provide redundancy is disk mirroring, a technique that places each unique data item on two disks. A drawback to mirroring is that only half of the storage capacity of the array can be used to store unique data. As we discuss in the sections that follow, some RAID levels employ a more sophisticated approach that reduces this overhead. 61 554 Disk Performance Optimization To realize the improved performance of striped disks with redundancy, the system must efficiently divide files into strips, form files from strips, determine the locations of strips in the array and implement the redundancy scheme. Using a general-purpose processor for this purpose can significantly degrade the performance of processes competing for processor time. Thus, many RAID systems contain spe- Operating Systems Thinking Mission-Critical Systems There are systems where the cost of failure is so high that enormous investment must be made in resources to ensure that the systems do not fail and perform well enough to meet their requirements. Such systems are often called mission-critical or businesscritical systems. If an air traffic control system were to fail. human lives could be lost. If the New York Stock Exchange systems were to fail, the world's stock markets could crash. Thus, operating systems must be designed to meet the unique reliability and performance requirements of particular applications systems. In some cases, the operating system must make extraordinary resource commitments to particular applications, far greater than what general-purpose applications would normally receive. In the Linux and Windows XP case studies, we will see, for example, that real-time applications can be given special treatment. Operating Systems Thinking Fault Tolerance Computer systems are employed in many key aspects of our personal and professional lives. We depend on computers not to fail. Fault tolerant systems are designed to keep operating, even in the presence of problems that might typically cause system failure. Redundancy is a popular means of achieving fault toler- ance—if a component fails, another " e q u a l " component takes over. It is costly to design fault tolerance into a system, but the costs can be small compared to the costs of system failure, especially in business-critical or mission-critical systems. We discuss issues of fault tolerance in many parts of the book, w i t h par- ticular emphasis on these techniques in our treatments of RAID, multiprocessing in Chapter 15, computer networking in Chapter 16, distributed systems in Chapter 17, distributed file systems in Chapter 18 and computer security in Chapter 19. 12.10 Redundant Arrays of Independent Disks (RAID) 555 cial-purpose hardware called a RAID controller to perform such operations quickly. RAID controllers also simplify RAID implementation by enabling the operating system simply to pass read and write requests to the RAID controller, which then performs striping and maintains redundant information as necessary. However, RAID controllers can significantly increase the cost of a RAID system. A systems designer choosing to adopt a RAID system must balance cost, performance and reliability. Typically, improving one characteristic worsens the other two. For example, to reduce the cost of the RAID system, one might reduce the number of disks in the array. A smaller number of disks often reduces performance by limiting transfer rates and can reduce reliability by limiting the array's capacity to store redundant information. Self Review 1. Explain when fine-grained and coarse-grained strips are appropriate in RAID systems. 2. Why should RAID systems provide some degree of fault tolerance? Ans: 1) Fine-grained strips are appropriate for systems requiring individual I/O operations to be performed quickly. Coarse-grained strips are appropriate for systems that must fulfill many I/O requests simultaneously. 2) The greater the number of disks in an array, the more likely it is that one of them will fail. Thus, RAID systems should provide additional fault tolerance. 12.10.2 Level 0 (Striping) RAID level 0 uses a striped disk array with no redundancy. Figure 12.17 illustrates a RAID 0 configuration containing four striped disks. If an application requests to read data stored in strips A and B of the array data from both strips can be read simultaneously, because they are on separate disks (Di and D2). Further, the RAID controller can simultaneously service a write request to strip K, located on disk D3. RAID level 0 was not one of the five original levels of RAID and is not considered a "true" RAID level because it does not provide fault tolerance. If one of the disks fails, all the data in the array that depend on the failed disk are lost. Depending on the array's strip size, all data stored in the array could become unusable with the loss of a single disk.62 RAID level 0 is simple to implement and does not incur storage overhead to provide fault tolerance. Further, a RAID 0 system with n disks performs reads and writes at a rate up to n times greater than that of a single disk. RAID 0 systems are appropriate for systems where high performance and low cost are more important than reliability.63, 64 Self Review 1. Why is RAID level 0 not considered a "true" RAID level? 2. What are the benefits of RAID level 0? Ans: 1) RAID level 0 does not provide fault tolerance through redundancy. 2) RAID level 0 provides high transfer rates, is simple to implement and does not incur storage overhead to provide fault tolerance. 556 Disk Performance Optimization Figure 12.17 | RAID level 0 (striping). 12.10.3 Level 1 (Mirroring) RAID level 1 employs disk mirroring (also called shadowing) to provide redundancy, so each disk in the array is duplicated. Stripes are not implemented in level 1, reducing both hardware complexity and system performance. Figure 12.18 displays a RAID 1 array. Notice that disks D1 and D2 contain the same data, and disks D3 and D 4 contain the same data. Level 1 arrays permit multiple I/O operations to be serviced simultaneously. For example, in Fig. 12.18, requests to read data stored on blocks A and B can be serviced concurrently by each disk of the mirrored pair, D1 and D 2 . To ensure consistency, modified data must be written to a pair of mirrored disks, so multiple write requests to the same mirrored pair must be serviced one at a time. Note that in Fig. 12.18, read requests for blocks A and B can be serviced by disks D1 and D2 while a write request to block I is serviced by disks D3 and D 4 . Although RAID level 1 provides the highest degree of fault tolerance of any RAID level, only half of the array's disk capacity can be used to store unique data. Thus, the cost per unit storage in a RAID 1 array is twice that in a RAID 0 array. Because each block of data is stored on a pair of disks, the system can sustain multiple disk failures without loss of data. For example, if disk D3 fails, the system can continue to operate using the mirrored data on D 4 . A new disk can subsequently be installed to replace D 3 . Recovering and rebuilding data from a failed disk, called data regeneration, entails copying the redundant data from the mirrored disk. However, if both disks in a mirrored pair fail, their data is unrecoverable. 65 12.10 Redundant Arrays of Independent Disks (RAID) 557 Figure 12.18 | RAID level 1 (mirroring). Some RAID systems contain "spare" disks (also called hot spare disks or online spares) that can replace failed disks, much like a spare tire for an automobile. Some systems also feature hot swappable disk drives that can be replaced while the system is online, which allows data regeneration to occur as the system runs—necessary in high-availability environments such as online transaction-processing systems.66 Level 1 arrays are characterized by the following advantages and disadvantages: • High storage overhead—only half of the array's storage capacity can be used to store unique data. • Fast average read transfer rates—two different read requests to data stored in the same mirrored pair can be serviced simultaneously. • Slower average write transfer rates—multiple write requests to a mirrored pair must be performed one at a time. However, write requests to different mirrored pairs can be performed simultaneously. • High fault tolerance—RAID level 1 can sustain multiple drive failures without loss of data or availability. It provides the highest degree of fault tolerance among the most popular RAID levels. • High cost—the storage overhead increases cost per unit capacity. 558 Disk Performance Optimization RAID 1 is best suited for environments in which reliability is a higher priority than cost or performance. 67 Self Review 1. Compare and contrast RAID 0 and RAID 1 in terms of cost, performance and reliability. 2. Why is RAID 1 appropriate for business- and mission-critical systems? Ans: 1) RAID 0 costs less than RAID 1 when capacity is held fixed. Read and write performance is dependent on system load and strip sizes, but both RAID 0 and RAID 1 can service read requests for data on different disks simultaneously. RAID 0 can service write requests for data on different disks simultaneously; RAID 1 can perform write requests simultaneously only if they are to data stored on different mirrored pairs. RAID 0 is much more likely to fail than a single disk; RAID 1 is much less likely to fail than a single disk. 2) In these systems, performance, reliability and availability are more important than cost. RAID 1 offers high performance for read requests and extremely high reliability, and it can continue to operate without loss of data if one or more disks fail (but at reduced performance levels). 12.10.4 Level 2 (Bit-Level Hamming ECC Parity) RAID level 2 arrays are striped at the bit level, so each strip stores one bit. Level 2 arrays are not mirrored, which reduces the storage overhead incurred by level 1 arrays. When a disk in a RAID system fails, it is analogous to a portion of a message that has been corrupted during data transfer, such as during network transmission. The problem of unreliable transmission has been well researched and several solutions have been proposed (for examples of solutions to the problem of unreliable transmission, see www. eccpage. com). RAID level 2 borrows from a technique that is commonly implemented in memory modules called Hamming error-correctins codes (Hamming ECCs), which uses parity bits to check for errors in data transmitted from disks and to correct them, if possible.68 Parity bits can be calculated as follows. Consider an array in which each stripe stores four bits. When the RAID system receives a request to write the data 0110 to a stripe, it determines the parity of the sum of the bits, i.e., whether the sum is odd or even. In this case, the parity sum (2) is even, so the system writes the stripe to the corresponding four data disks of the array and a 0, representing even parity, to a parity disk. [Note: Although data is striped at the bit level, most requests correspond to data in multiple stripes that can be written at once to improve performance.] RAID systems generally place parity bits on a separate disk, so that reads and writes to each stripe can be performed by all disks simultaneously. When the stripe is next accessed, the system reads the stripe and its corresponding parity bit. calculates the stripe's parity and compares it to the value read from the parity disk. If a disk error causes the third bit of the stripe to change from 1 to 0 (i.e., the stripe stores 0100), the stripe will have odd parity, and the system can detect the error by comparing the parity to the value stored on the parity disk. A limitation of this particular form of parity is that it cannot detect an even number of errors. For example, if both the second and third bits change from 1 to 0 12.10 Redundant Arrays of Independent Disks (RAID) 559 (i.e., the stripe stores 0000), the stripe will still have even parity. Further, this technique does not enable the system to determine which, if any, bits have errors. Hamming error-correcting codes (Hamming ECCs) use a more sophisticated approach to enable the system to detect up to two errors, correct up to one error and determine the location of the error in a stripe (the algorithm behind Hamming ECC parity generation is beyond the scope of this book; it is described at www2.rad.com/networks/1994/err_con/hamming.htm). The size of Hamming ECC codes, and thus the number of parity disks, increases according to the logarithm of the number of data disks. For example, an array of 10 data disks requires four parity disks; an array of 25 data disks requires five parity disks.69 Thus, level 2 arrays containing a large number of disks incur significantly less storage overhead than level 1 arrays. Figure 12.19 illustrates a level 2 array containing four data disks and three parity disks. Stripe A is composed of the strips A0-A3, which occupy the first strips on the data disks D1-D4. Stripe As corresponding Hamming ECC is composed of the bits A x -A z , which occupy the first strips on the parity disks P1-P3. Consider a write request to store data in strips A 0 , A 1 , A 2 , A 3 , B 0 and B 1 . In this case, because the Hamming ECC code is computed for each write, the controller must write stripes A and B, and the Hamming ECC code for each stripe. Even Figure 12.19 | RAID level 2 (bit-level ECC parity). 560 Disk Performance Optimization though only two of the four bits in stripe B are modified, the entire B stripe must be read and its Hamming ECC code calculated and written to disk. This process, which degrades performance because the system must access the array twice for each write, is called a read-modify-write cycle. Although the ECC disks reduce the overhead associated with redundancy when compared to mirroring, they can degrade performance because multiple requests cannot be serviced simultaneously. Each read request requires the array to access all disks, compute the ECC, and compare it with the value retrieved from the parity disks. Some systems reduce this bottleneck by computing parity information only for write requests. However, the system must access all disks to maintain parity information when performing a write, meaning that write requests must be serviced one at a time. One way to increase the maximum number of concurrent reads and writes using a RAID level 2 array is to divide the system's disks into several small RAID level 2 arrays. Unfortunately, this technique increases storage overhead, because smaller groups require a a larger ratio of parity disks to data disks. For example, a 32-data-disk level 2 array requires six parity disks. If the system were divided into eight groups of four data disks, the resulting arrays would require 24 parity disks.70 The primary reason that RAID 2 is not implemented in today's systems is that modern disk drives maintain Hamming ECCs or comparable fault-tolerance mechanisms transparently in hardware. Almost all current SCSI and IDE drives contain built-in error detection and disk monitoring. 71 Self Review 1. How does dividing a large RAID level 2 array into subgroups affect cost, performance and reliability? 2. Why is RAID level 2 rarely implemented? Ans: 1) Dividing a level 2 array into subgroups improves performance because multiple requests can be serviced simultaneously. Subgroups do not affect reliability, because the same error-checking routines are performed for each subgroup. As a whole, the subgroups require more parity disks than a single large array, which increases cost. 2) Because modern hardware devices have built-in error detection and disk monitoring that perform the same checks as level 2 arrays, designers seek a lower-cost and higher-performance alternative to providing fault tolerance. 12.10.5 level 3 (Bit-level XOR ECC Parity) RAID level 3 stripes data at the bit or byte level. Instead of using Hamming ECC for parity generation, RAID 3 uses XOR (exclusive or) error-correcting codes (XOR ECCs). The XOR ECC algorithm is much simpler than the Hamming ECC. It is called XOR ECC because it uses the logical XOR operation: (a XOR b) = 0 when a and b are both zero or both one; = 1 otherwise. 12.10 Redundant Arrays of Independent Disks (RAID) 561 It follows that (a XOR (b XOR c)) = 0 only when an even number of the arguments are 1 or 0. RAID level 3 exploits this fact to perform nested XOR operations on each byte to generate its XOR ECC. For example, consider stripe A in Fig. 12.20. Let A0 = 1, A1 = 0, A2 = 0 and A3 = 1; or, more succinctly, A = 1001. The system uses nested XOR operations to compute A's parity. If the number of l's or 0's is even, the parity bit, Ap, is 0; otherwise, Ap is 1. In this case there are two l's, so Ap = 0. If the system stripes at the byte level, the XOR ECC is calculated for each bit, so each parity strip stores 8 bits. XOR ECC uses only one disk to hold parity information, regardless of the size of the array. Note that, unlike Hamming ECCs, XOR ECCs do not enable the system to detect which bit contains erroneous data. This is acceptable, because most parity errors in RAID systems result from the failure of an entire disk, which is easy to detect. For example, assume that disk D2 has failed (Fig. 12.20). In this case, when the system attempts to read stripe A, it finds A = 1x01 (where x is the unknown bit) and Ap - 0. Because Ap = 0, stripe A must have stored an even number of l's. Thus, the system can determine that A2 = 0 and regenerate the lost data. This technique can be applied to recover from any single data disk failure; if the parity disk fails, recovery entails recomputing the parity from the data disks.72 Figure 12. 20 | RAID level 3 (bit-level, single parity disk). 562 Disk Performance Optimization Due to fine-grained striping, most reads require access to the entire array. Further, due to parity generation, only one write can be performed at a time. Similar to RAID level 2, this yields high transfer rates when reading and writing large files, but, in general, only one request can be serviced at a time.73 The primary advantage of RAID level 3 is that it is easy to implement, offers reliability similar to RAID level 2, and entails significantly lower storage overhead than RAID level 2.74 Self Review 1. Compare and contrast RAID levels 2 and 3. 2. Why is XOR ECC parity, instead of Hamming ECC parity, sufficient for most RAID systems? Ans: 1) Both RAID levels 2 and 3 provide fault tolerance using parity to reduce the overhead of mirroring, both use dedicated parity disks and, in general, both cannot service multiple requests simultaneously. RAID level 2 incurs greater storage overhead than level 3. 2) Although XOR ECCs cannot determine which disk has failed, the system can easily determine this, because failed disks do not respond to requests.Therefore, the system can regenerate one failed disk using a single parity disk. 12.10.6 Level 4 (Block-Level XOR ECC Parity) RAID level 4 systems are striped using fixed-size blocks (typically much larger than a byte) and use XOR ECC to generate parity data, which requires a single parity disk. Figure 12.21 depicts a RAID 4 array. Note that the only organizational difference between level 3 and level 4 arrays is that each data strip in a level 4 array stores more data. In RAID levels 2 and 3, each request's data is typically stored on each data disk in the array. However, because RAID level 4 permits coarse-grained strips, it is possible that requested data might be stored on a small fraction of the disks in the array. Thus, if parity is not determined for each read, the system can potentially service multiple read requests simultaneously. Because the ECC codes are used primarily for data regeneration as opposed to error checking and correcting, many systems eliminate the parity calculation when reads are performed so that multiple read requests can be serviced simultaneously.75 When servicing a write request, however, the system must update parity information to ensure that no data is lost in the event of a disk failure. When coarsegrained strips are employed, write requests rarely modify data on each disk in the array. Again, accessing each disk in the array to compute parity information can lead to substantial overhead. Fortunately, the system can calculate the new parity bit, Ap', simply by using the data block before modification, Ad, the modified data block, Ad', and the corresponding parity block on disk, Ap: Ap' = (Ad XOR Ad') XOR Ap.76 12.10 Redundant Arrays of Independent Disks (RAID) 563 You will be asked to prove this relation in the exercises. This technique eliminates wasteful disk I/O operations, because writes do not require access to the entire array. However, because each write request must update the parity disk, write requests must be performed one at a time, creating a write bottleneck. As we discuss in the next section, RAID level 5 removes the write bottleneck, so RAID level 4 is rarely implemented.77-78-79-80 Figure 12.21 | RAID level 4 (block-level parity). Self Review 1. How does RAID level 4 offer higher performance than RAID level 3? 2. Why can RAID level 4 perform only one write at a time? Ans: 1) RAID level 4 permits coarse-grained striping, which enables multiple read requests to be performed simultaneously and reduces the number of disks accessed for each write request. 2) RAID level 4 uses one parity disk that must be accessed to update parity information each time data in the array is modified. 12.10.7 Level 5 (Block-Level Distributed XOR ECC Parity) RAID level 5 arrays are striped at the block level and use XOR ECC parity, but parity blocks are distributed throughout the array of disks (Fig. 12.22).81 Note that in Fig. 12.22, the parity for the first stripe is placed on disk D5, while the parity for the second stripe is placed on disk D 4 . Because parity blocks are distributed across many disks, multiple parity strips can be accessed simultaneously, removing the write bottleneck for many requests. For example, consider how RAID level 5 services write requests to strips A1 and C 2 . The controller accesses disk D2 to write strip A 1 and disk D 5 to update the corresponding parity block, Ap. The controller can simultaneously access disk D 4 to write strip C 2 and access disk D 3 to update its corresponding parity block, Cp. Although RAID level 5 improves write performance by distributing parity, level 5 arrays must still perform a read-modify-write cycle for each write request, 564 Disk Performance Optimization Figure 12.22 | RAID level 5 (block-level distributed parity). requiring at least four I/O operations to service each write request. If a system consistently writes small amounts of data, the number of costly I/O operations can significantly degrade performance. 82 Several methods have been developed to address this issue. Caching recently accessed data and parity blocks can reduce the number of I/O operations in the read-modify-write cycle. Parity logging can improve RAID level 5 performance by storing the difference between the old parity and new parity (called the update image) in memory instead of performing a read-modify-write cycle. Because a single update image can store parity information corresponding to multiple write requests, the system can reduce I/O overhead by performing a single update to the parity block in the array after several writes have been performed. 83 A similar performance-enhancing technique is called AFRAID (A Frequently Redundant Array of Independent Disks). Instead of performing the read-modify-write cycle with every write, parity data generation is deferred to times when system load is light. This can greatly improve performance in environments characterized by intermittent bursts of requests generating small writes.84 Although RAID level 5 increases performance relative to RAID levels 2-4, it is complex to implement, which increases its cost. Also, because parity is distributed throughout the array, data regeneration is more complicated than in other RAID levels.85 Despite its limitations, RAID level 5 is frequently adopted due to its effective balance of performance, cost and reliability. Level 5 arrays are considered general-purpose arrays and are often found in file and application servers, enterprise resource planning and other business systems.86 Figure 12.23 compares the six levels of RAID presented in this chapter. 12.10 Redundant Arrays of Independent Disks (RAID) 565 Figure 12.23 | Comparison of RAID levels 0-5. Other RAID levels Many other RAID levels exist, each borrowing from one or more of the RAID levels we discussed in the preceding sections.87 Unfortunately, there is no standard naming convention for RAID levels, which often leads to confusing or misleading names. Some of the more notable RAID levels include • RAID level 6—extends RAID 5 by distributing two parity blocks per stripe for higher reliability upon disk failure.88 • RAID level 0+1 — a set of striped disks (level 0) whose image is mirrored to a second set of disks (level 1). • RAID level 10—a set of mirrored data (level 1) that is striped across another set of disks (level 0), requiring a minimum of four disks. Other RAID arrays that have been developed include levels 0+3, 0+5, 50,1+5, 51 and 53.89 RAID level 7, which is a high-performance, high-cost solution, is a proprietary technique whose name is trademarked by the Storage Computer Corporation.90 Self Review 1. How does RAID level 5 address the write bottleneck in RAID level 4? 2. What techniques have been implemented to improve performance in RAID systems that receive many requests for small writes? Ans: 1) RAID level 5 distributes parity information across all disks in the array so that multiple write operations can be performed simultaneously. 2) Caching recently accessed data improves write performance by reducing the number of I/O operations generated by readmodify-write cycles. Delaying the update of parity information improves performance by reducing the number of disks accessed to perform each write operation. 566 Disk Performance Optimization Web Resources www.disk-tape-data-recovery.com/storage-history.htm Provides a short history of persistent data storage media. www.almaden.ibm.com/sst/storage/index.shtml Describes physical limitations of storage technology and promising fields of research IBM has pursued to solve them. colossalstorage.net/colossalll.htm Provides links to sites describing evolving storage technologies. www.cose.brocku.ca/~cspress/Hel1oWorld/1999/03mar/disk_scheduling_algorithms.html Contains an article from Hello World, a magazine for computer science students; the article overviews disk scheduling algorithms. www.peguide.com/ref/hdd/index.htm Discusses hard disk characteristics. www.microsoft.com/windows2000/techinfo/administration/fileandprint/defrag.asp Describes how to use the disk defragmenting utilities in Windows 2000. www.pcguide.com/ref/hdd/perf/raid/ Provides an in-depth introduction to RAID. www.acne.com/04_00.html Provides a summary of most RAID levels. linas.org/linux/raid.html Contains articles about RAID and its implementation in Linux systems. www.pdl.emu.edu/RAID/ Overviews RAID and provides links to current RAID research. Summary As processor and main memory speeds increase more rapidly than those of secondary storage devices, optimizing disk performance has become important to realizing optimal performance. Magnetic storage records data by changing the direction of magnetization of regions, each representing a 1 or a 0. To access data, a current-carrying device called a read/write head hovers above the medium as it moves. Most modern computers use hard disks as secondary storage. As the platters spin, each read-write head sketches out a circular track of data on a disk surface. All read-write heads are attached to a single disk arm (also called an actuator, boom or moving-arm assembly). When the disk arm moves the read/write heads to a new position, a different set of tracks, or cylinder, becomes accessible. The process of moving the disk arm to a new cylinder is called a seek operation. The time it takes for the head to move from its current cylinder to the one containing the data record being accessed is called the seek time. The time it takes for data to rotate from its current position to a position adjacent to the read/write head is called latency time. Then the record, which is of arbitrary size, must be made to spin by the read/write head so that the data can be read from or written to the disk. This is called transmission time. Many processes can generate requests for reading and writing data on a disk simultaneously. Because these processes sometimes make requests faster than they can be serviced by the disk, queues build up to hold disk requests. Some early computing systems simply serviced these requests on a first-come-first-served (FCFS) basis, in which the earliest arriving request is serviced first. FCFS is a fair method of allocating service, but when the request rate (i.e., the load) becomes heavy, FCFS results in long waiting times. FCFS exhibits a random seek pattern in which successive requests can cause time-consuming seeks from the innermost to the outermost cylinders. To reduce the time spent seeking records, it seems reasonable to order the request queue in some other manner than FCFS. This reordering is called disk scheduling. The two most common types of scheduling are seek optimization and rotational optimization. Disk scheduling strategies often are evaluated by comparing their throughput, mean response time and variance of response times. First-come-first-served scheduling uses a FIFO queue so that requests are serviced in the order they arrive. Although this technique incurs low execution-time overhead, it can result in low throughput due to lengthy seeks. Shortest-seek-time-first (SSTF) scheduling next services the request that is closest to the read-write head's current cylinder (thus incurring the shortest seek time), even if that is not the first one in the queue. By reducing average seek times, SSTF achieves higher throughput rates than FCFS, and mean response times tend to be lower for moderate loads. One significant drawback is that higher variances occur on response times because of the discrimination against the outermost and innermost tracks: in the extreme, indefinite postponement of requests far from the read-write heads could occur. The SCAN scheduling strategy reduces unfairness and variance of response times by choosing the request that requires the shortest seek distance in a preferred direction. Summary Thus, if the preferred direction is currently outward, the SCAN strategy chooses the shortest seek distance in the outward direction. SCAN behaves much like SSTF in terms of high throughput and good mean response time. However, because SCAN ensures that all requests in a given direction will be serviced before the requests in the opposite direction, it offers a lower variance of response times than SSTF. The circular SCAN (C-SCAN) modification to the SCAN strategy moves the arm from the outer cylinder to the inner cylinder, servicing requests on a shortest-seek basis. When the arm has completed its inward sweep, it jumps (without servicing requests) to the outermost cylinder, then resumes its inward sweep processing requests. CSCAN maintains high levels of throughput while further limiting variance of response times by avoiding the discrimination against the innermost and outermost cylinders. SCAN is often called the elevator algorithm. The FSCAN and JV-Step SCAN modifications to the SCAN strategy eliminate the possibility of indefinitely postponing requests. FSCAN uses the SCAN strategy to service only those requests waiting when a particular sweep begins (the "F" stands for "freezing" the request queue at a certain time). Requests arriving during a sweep are grouped together and ordered for optimum service during the return sweep. jV-Step SCAN services the first n requests in the queue using the SCAN strategy. When the sweep is complete, the next n requests are serviced. Arriving requests are placed at the end of the request queue, which prevents requests in the current sweep from being indefinitely postponed. FSCAN and N-Step SCAN offer good performance due to high throughput, low mean response times and a lower variance of response times than SSTF and SCAN. The LOOK variation of the SCAN strategy (also called the elevator algorithm) "looks" ahead to the end of the current sweep to determine the next request to service. If there are no more requests in the current direction, LOOK changes the preferred direction and begins the next sweep, stopping when passing a cylinder that corresponds to a request in the queue. This strategy eliminates unnecessary seek operations experienced by other variations of the SCAN strategy by preventing the read/write head from moving to the innermost or outermost cylinders unless it is servicing a request to those locations. The circular LOOK (C-LOOK) variation of the LOOK strategy uses the same technique as C-SCAN to reduce the bias against requests located at extreme ends of the platters. When there are no more requests on a current sweep, the read/write head moves to the request closest to the outer cylinder (without servicing requests in between) 567 and begins the next sweep. The C-LOOK strategy is characterized by potentially lower variance of response times compared to LOOK, and by high throughput, although lower than that of LOOK. In many environments, there are often short periods when there are no waiting requests for the disk to service and the disk arm sits idle, waiting for the next request. If the disk arm is in the center of the disk, or at some hot spot of disk activity, the average seek time will be less. Moving the disk arm to a location that will hopefully minimize the next seek is known as disk arm anticipation. It can be useful in environments where processes's disk request patterns exhibit locality and where there is typically enough time to move the disk arm between disk requests. If a request is received during anticipatory movement, moving the head to the originally specified location and then to the requested cylinder increases response time. Allowing the seek to a hot spot to be interrupted to adjust to requests provides the greatest performance boost. Rotational optimization was used extensively with early fixed-head devices such as drums, and can improve performance in today's hard disks, which exhibit seek times and average latencies of the same order of magnitude. The shortest-latency-time-first (SLTF) strategy examines all pending requests and services the one with the shortest rotational delay first. Rotational optimization is sometimes referred to as sector queuing; requests are queued by sector position around the disk, and the nearest sectors are serviced first. The STLF strategy works well for fixed-head disks but does not incorporate the positioning time, which is the seek time plus latency, of moving-head disks. The shortestpositioning-time-first (SPTF) strategy next services the request that requires the shortest positioning time. SPTF results in high throughput and a low mean response time, similar to SSTF, and can also indefinitely postpone requests to the innermost and outermost cylinders. A variation of SPTF is the shortest-access-time-first (SATF) strategy, which next services the request that requires the shortest access time (i.e., positioning time plus transmission time). SATF exhibits higher throughput than SPTF, but large requests can be indefinitely postponed by a series of smaller requests. Requests to the innermost or outermost cylinders can be indefinitely postponed by requests to intermediate cylinders. SPTF and SATF can improve performance by implementing the "look-ahead" mechanisms employed by the LOOK and C-LOOK strategies. One reason why SPTF and SATF are rarely implemented is that they require knowledge of disk characteris- 568 Disk Performance Optimization tics including latency, track-to-track seek times and relative locations of sectors. Unfortunately, many of today's hard drives hide such geometry or expose misleading geometries due to hidden error-checking data and bad sectors. When disk storage proves to be a bottleneck, some designers recommend adding more disks to the system. This does not always solve the problem, because the bottleneck could be caused by a large request load on a relatively small number of disks. When this situation is detected, disk scheduling may be used as a means of improving performance and eliminating the bottleneck. Disk scheduling might not be useful in a batch-processing system with a relatively low degree of multiprogramming. Scheduling becomes more effective as the degree of multiprogramming increases, which increases the system load and leads to unpredictable disk request patterns. Nonuniform request distributions are common in certain situations and their consequences have been investigated. One study determined that the vast majority of disk references are to the same cylinder as the immediately preceding reference. Sophisticated file organization techniques can cause a proliferation of requests with large seek times. Such techniques are convenient for the applications designer but can complicate implementation and degrade performance. Many systems maintain a disk cache buffer, which is a region of main memory that the operating system reserves for disk data. In one context, the reserved memory acts as a cache, allowing processes quick access to data that would otherwise need to be fetched from disk. The reserved memory also acts as a buffer, allowing the operating system to delay writing the data to improve I/O performance. The disk cache buffer presents several challenges to operating system designers, because memory allocated to the cache buffer reduces the amount allocated to processes. Also, because memory is volatile, power or system failures can lead to

inconsistencies. Write-back caching writes buffered data to disk periodically, enabling the operating system to use a single request to batch multiple I/Os, which can improve system performance. Write-through caching writes data both to the disk cache buffer and to disk each time cached data is modified. This technique prevents the system from batching requests but reduces the possibility of inconsistent data in the event of a system crash. Many of today's hard disk drives maintain an independent high-speed buffer cache (often called an on-board cache) of several megabytes. If data requested is stored in the on-board cache, the hard drive can deliver the data at or near the speed of main memory. However, one study showed that disks that include on-board caches typically use replacement strategies that adapt poorly to random request patterns, leading to suboptimal performance. Disks tend to become fragmented as files and records are added and deleted. Many operating systems provide defragmentation (or disk reorganization) programs that can be used periodically to reorganize files. This allows the consecutive records of sequential files to be placed contiguously on the disk. Also, operating systems can place files that are likely to be modified near free space to reduce future fragmentation. Some operating systems allow users to partition a disk into separate areas. Files are then restricted to these areas, so fragmentation is reduced. Some systems use data compression techniques to reduce the amount of space required by information on disk. Data compression decreases the size of a record by replacing common bit patterns with shorter ones. This can reduce seeks, latency times and transmission times, but may require substantial processor time to compress the data for storage on the disk, and to decompress the data to make it available to applications. Systems that need to access certain information quickly benefit from placing multiple copies of that data at different positions on the disk. This can substantially reduce seek and rotation times, but the redundant copies can consume a significant portion of the disk. Also, if one copy is modified, all other copies must be modified, which can degrade performance. Additionally, record blocking can yield significant performance improvements by reading contiguous records as a single block, requiring only a singleseek. Systems that monitor disk access attempts can try to keep frequently accessed data in a favorable position in the memory hierarchy (i.e., in main or cache memory) while transferring infrequently referenced data to slower storage (such as a hard disk or compact disk). Redundant Arrays of fndependent Disks (RAID) is a family of techniques that use arrays of disks to address the problem of relatively slow disk transfer speeds. Each array organization, called a level, is characterized by data striping and redundancy. Storage is divided into fixed-size strips; contiguous strips on each disk generally store noncontiguous data or data from different files. A stripe consists of the set of strips at the same location on each disk in the array. When selecting a strip size, the system designer should consider the average size of disk requests. Smaller strip sizes, also called fine-grained strips, tend to spread file data across several disks. Fine-grained strips can reduce each request's access time and increase transfer rates Summary because multiple disks simultaneously retrieve portions of the requested data. While these disks service a request, they cannot be used to service other requests in the system's request queue. Large strip sizes, also called coarse-grained strips, enable more files to fit entirely within one strip. In this case, some requests can be serviced by only a portion of the disks in the array, so it is more likely that multiple requests can be serviced simultaneously. However, smaller requests are serviced by one disk at a time, which reduces transfer rates for individual requests. Systems such as Web servers and databases, which typically access multiple small records simultaneously, benefit from coarse-grained strips because several I/O operations can occur concurrently; systems such as supercomputers that require fast access to a small number of records benefit from fine-grained strips, which provide high transfer rates for individual requests. As the number of disks in the array increases, so does the likelihood that one of those disks will fail, measured by the mean-time-to-failure (MTTF). Most RAID systems maintain copies of data to help enable them to recover from errors and failures, a technique called redundancy. Disk mirroring is a redundancy technique that maintains copies of each data item on two disks. A drawback to mirroring is that only half of the storage capacity of the array can be used to store unique data. To realize the improved performance of striped disks with redundancy, the system must efficiently divide files into strips, form files from strips, determine the locations of strips in the array and implement redundancy. Many RAID systems contain special-purpose hardware called a RAID controller to perform such operations quickly. The designer must balance cost, performance and reliability when selecting a RAID level. Typically, improving one characteristic worsens the other two. RAID level 0 uses a striped disk array with no redundancy. Level 0 arrays are not fault tolerant; if one disk fails, all the data in the array that depend on the failed disk are lost. Depending on the array's strip size, all data stored in the array could become unusable with the loss of a single disk. Although RAID 0 is not fault tolerant, it is simple to implement, yields high transfer rates and does not incur any storage overhead. RAID level 1 employs disk mirroring (also called shadowing) to provide redundancy, so each disk in the array is duplicated. Stripes are not implemented in level 1, reducing both hardware complexity and system performance. While this results in the highest degree of fault tol- 569 erance of any RAID level, only half of the array's capacity can be used to store unique data, which increases cost. For each mirrored pair, two read requests can be serviced simultaneously, but only one request can be serviced at a time during a write. Recovering and rebuilding data from a failed disk, called data regeneration, entails copying the redundant data from the mirrored disk. If both disks in a mirrored pair fail, their data is unrecoverable. Some RAID systems contain "spare" disks (also called hot spare disks or online spares) that can replace failed disks, much like a spare tire for an automobile. Some systems also feature hot swappable disk drives that can to be replaced while the system is online. In these systems, data regeneration can occur while the system continues to run, which is necessary in high-availability environments such as online transaction processing systems. RAID 1 is best suited for business-critical and mission-critical environments in which reliability has higher priority than cost or performance. RAID level 2 arrays are striped at the bit level, so each strip stores one bit. Level 2 arrays are designed to reduce the storage overhead incurred by implementing fault tolerance using mirroring. Instead of maintaining redundant copies of each data item, RAID level 2 uses a technique of storing parity information called Hamming error-correcting codes (Hamming ECCs), which allow the system to detect up to two errors, correct up to one error and determine the location of the error in a stripe. The size of Hamming ECC codes, and thus the number of parity disks, increases in proportion to the logarithm (base 2) of the number of data disks. Thus, level 2 arrays containing a large number of disks incur significantly less storage overhead than level 1 arrays. Although the ECC disks reduce the overhead associated with redundancy when compared to mirroring, they can degrade performance, because multiple requests cannot be serviced simultaneously. Each read request requires the array to access all disks, compute the ECC, and compare it with the value retrieved from the parity disks. Similarly, when servicing a write request, the Hamming ECC must be calculated for each stripe that is written. Further, the system must perform a read-modify-write cycle, which degrades performance because the system must access the entire array twice for each write. One way to enable concurrency using a RAID level 2 array is to divide the system's disks into several small RAID level 2 arrays, but this increases storage overhead and cost. The primary reason that RAID 2 is not implemented in today's systems is that modern disk drives perform Hamming ECCs or comparable protection transparently in hardware. 570 Disk Performance Optimization RAID level 3 stripes data at the bit or byte level. RAID 3 uses XOR (exclusive or) error-correcting codes (XOR ECCs), which use the logical XOR operation to generate parity information. XOR ECC uses only one disk to hold parity information, regardless of the size of the array. The system can use the parity bits to recover from any single disk failure; if the parity disk fails, it can be rebuilt from the array's data disks. Due to parity checking RAID level 3 reads and writes require access to the entire array. Similar to RAID level 2, this yields high transfer rates when reading and writing large files, but only one request can be serviced at a time. RAID level 4 systems are striped using fixed-size blocks (typically larger than a byte) and use XOR ECC to generate parity data that is stored on a single parity disk. Because level 4 arrays enable coarse-grained striping, the system can potentially service multiple read requests simultaneously, if parity is not determined for each read. When servicing a write request, however, the system must update parity information to ensure that no data is lost in the event of a disk failure. This means that write requests must be performed one at a time, creating a write bottleneck. RAID level 5 arrays are striped at the block level and use XOR ECC for parity, but parity blocks are distributed throughout the array of disks. Because parity blocks are distributed across many disks, multiple parity blocks can be accessed simultaneously, removing the write bottleneck for many requests. Although RAID level 5 improves write per- formance by distributing parity, level 5 arrays must still perform a read-modify-write cycle for each write request, requiring at least four I/O operations to service each write request. Caching recently accessed data and parity blocks can reduce the number of I/O operations in the read-modifywrite cycle. Parity logging can improve RAID level 5 performance by storing the difference between the old parity and new parity (called the update image) in memory instead of performing a read-modify-write cycle. A Frequently Redundant Array of Independent Disks (AFRAID) improves performance by deferring parity data generation to times at which the system load is light. Although RAID level 5 increases performance relative to RAID levels 2-4, it is complex to implement, which increases its cost. Also, because parity is distributed throughout the array, data regeneration is more complicated than in other RAID levels. Despite its limitations. RAID level 5 is frequently adopted for its effective balance of performance, cost and reliability. Level 5 arrays are considered general-purpose arrays and are often found in file and application servers, enterprise resource planning (ERP) and other business systems. Many other RAID levels exist, each borrowing from one or more of RAID levels 0-5. Unfortunately, there is no standard naming convention for RAID levels, which often leads to confusing or misleading names. Key Terms A Frequently Redundant Array of Independent Disks (AFRAID) (RAID)—RAID implementation that defers parity data generation to times when the system's load is light to improve performance when a system performs many small writes. actuator—See disk arm. anticipatory movement—Movement of the disk arm during disk arm anticipation. block—Fixed-size unit of data, typically much larger than a byte. Placing contiguous data records in blocks enables the system to reduce the number of I/O operations required to retrieve them. blocking—Grouping of contiguous records into larger blocks that can be read using a single I/O operation. This technique reduces access times by retrieving many records with a single I/O operation. boom—See disk arm. bottleneck—Condition that occurs when a resource recerves requests faster than it can process them, which can slow process execution and reduce resource utilization. Hard disks are a bottleneck in most systems. circular LOOK (C-LOOK) disk scheduling—Disk scheduling strategy that moves the arm in one direction, servicing requests on a shortest-seek basis. When there are no more requests on a current sweep, the read-write head moves to the request closest to the cylinder opposite its current location (without servicing requests in between) and begins the next sweep. The C-LOOK policy is characterized by potentially lower variance of response times compared to LOOK; it offers high throughput (although lower than that of LOOK). circular SCAN (C-SCAN) disk scheduling—Disk scheduling strategy that moves the arm in one direction, servicing requests on a shortest-seek basis. When the arm has completed its a sweep, it jumps (without servicing requests) to Key Terms 571 the cylinder opposite its current location, then resumes its inward sweep processing requests. C-SCAN maintains high levels of throughput while further limiting variance of response times by avoiding the discrimination against the innermost and outermost cylinders. coarse-grained strip (RAID)—Strip size that enables average files to be stored in a small number of strips. In this case, some requests can be serviced by only a portion of the disks in the array, so it is more likely that multiple requests can be serviced simultaneously. If requests are small, they are serviced by one disk at a time, which reduces their average transfer rates. cylinder—Set of tracks that can be accessed by the read-write heads for a specific position of the disk arm. data compression—Technique that decreases the size of a data record by replacing repetitive patterns with shorter bit strings. This can reduce seeks, and transmission times, but may require substantial processor time to compress the data for storage on the disk, and to decompress the data to make it available to applications. data regeneration (RAID)—Reconstruction of lost data (due to disk errors or failures) in RAID systems. data striping (RAID)—Technique in RAID systems that divides contiguous data into fixed-size strips that can be placed on different disks. This enables multiple disks to service requests for data. defragmentation— Moving parts of files so that they are located in contiguous blocks on disk. This can reduce access times when reading from or writing to files sequentially. disk arm—Moving-head disk component that moves readwrite heads linearly, parallel to disk surfaces. disk arm anticipation—Moving the disk arm to a location that will minimize the next seek. Disk arm anticipation can be useful in environments where process disk request patterns exhibit locality and when the load is light enough that there is sufficient time to move the disk arm between disk requests without degrading performance. disk cache buffer—A region of main memory that the operating system reserves for disk data. In one context, the reserved memory acts as a cache, allowing processes quick access to data that would otherwise need to be fetched from disk. The reserved memory also acts as a buffer, allowing the operating system to delay writing the data to improve I/O performance by batching multiple writes into a small number of requests. disk mirroring (RAID)—Data redundancy technique in RAID that maintains a copy of each disk's contents on a separate disk. This technique provides high reliability and simplifies data regeneration but incurs substantial storage overhead, which increases cost. disk reorganization—Technique that moves file data on disk to improve its access time. One such technique is defragmentation, which attempts to place sequential file data contiguously on disk. Another technique attempts to place frequently requested data on tracks that result in low average seek times. disk scheduling—Technique that orders disk requests to maximize throughput and minimize response times and the variance of response times. Disk scheduling strategies improve performance by reducing seek times and rotational latencies. elevator algorithm—See SCAN disk scheduling. first-come-first-served (FCFS) disk scheduling—Disk scheduling strategy in which the earliest arriving request is serviced first. FCFS is a fair method of allocating service, but when the request rate (i.e., the load) becomes heavy, FCFS can result in long waiting times. FCFS exhibits a random seek pattern in which successive requests can cause time-consuming seeks from the innermost to the outermost cylinders. fine-grained strip (RAID)—Strip size that causes average files to be stored in multiple stripes. Fine-grained strips can reduce each request's access time and increase transfer rates because multiple disks simultaneously retrieve portions of the requested data. FSCAN disk scheduling—Disk scheduling strategy that uses SCAN to service only those requests waiting when a particular sweep begins (the " F " stands for "freezing" the request queue at a certain time). Requests arriving during a sweep are grouped together and ordered for optimum service during the return sweep. fragmented disk—Disk that stores files in discontinuous blocks as the result of file creation and deletion. Such disks exhibit high seek times when reading files sequentially. Disk defragmentation can reduce or eliminate the problem. Hamming error-correcting codes (Hamming ECCs)—Technique of generating parity bits that enables systems to detect and correct errors in data transmission. hard disk drive—Magnetic, rotational secondary storage device that provides persistent storage for, and random access to, data. hot spare disk (RAID)—Disk in a RAID system that is not used until a disk fails. Once the system regenerates the failed disk's data, the hot spare disk replaces the failed disk. 572 Disk Performance Optimization hot spot—Disk cylinder that contains frequently requested data. Some disk arm anticipation techniques move the disk head to hot spots to reduce average seek times. load—See request rate. level (RAID) —A particular organization of a RAID system, such as level 1 (mirroring) or level 2 (Hamming ECC parity). See also RAID level 0, RAID level 1, RAID level 2, RAID level 3, RAID level 4 and RAID level 5. LOOK disk scheduling—Variation of the SCAN disk scheduling strategy that "looks" ahead to the end of the current sweep to determine the next request to service. If there are no more requests in the current direction, LOOK changes the preferred direction and begins the next sweep, stopping when passing a cylinder that corresponds to a request in the queue. This strategy eliminates unnecessary seek operations experienced by other variations of SCAN by preventing the read-write head from moving to the innermost or outermost cylinder unless it needs to service a request at those cylinders. magnetic tape storage—Rewritable magnetic storage medium that accesses data sequentially. Its sequential nature makes it unsuitable for direct access applications. mean response time (disk scheduling)—Average time a system spends waiting for a disk request to be serviced. mean-time-to-failure (MTTF) (RAID)—Average time before a single-disk failure. mirroring (RAID) —See disk mirroring. moving-arm assembly—See disk arm. nonuniform request distribution—Set of disk requests that are not uniformly distributed across disk surfaces. This occurs because processes exhibit spatial locality, leading to localized request patterns. N-Step SCAN disk scheduling—Disk scheduling strategy that services the first n requests in the queue using the SCAN strategy. When the sweep is complete, the next n requests are serviced. Arriving requests are placed at the end of the request queue. N-Step SCAN offer good performance due to high throughput, low mean response times and a lower variance of response times than SSTF and SCAN. online spare—See hot spare disk. online transaction processing (OLTP)—Type of system that typically receives many disk requests to randomly distributed locations containing small amounts of data (e.g., databases and Web servers). Such systems significantly improve performance using disk scheduling algorithms. partition—Area of a disk whose boundaries cannot be crossed by file data. Partitions can reduce disk fragmentation. parity—Technique that detects an even number of errors in data transmission. Parity information is generated by determining whether the data contains an even (or odd) number of l's (or O's). This parity information is generated after transmission and compared to the value generated before transmission. Error-correction codes (ECCs). such as Hamming or XOR, use the parity of a string of bits to detect and correct errors. Parity enables RAID systems to provide fault tolerance with lower storage overhead than mirrored systems. parity logging (RAID)—Technique that increases write performance in RAID systems using parity by postponing writes to the parity disk while the system is busy. Because parity logging stores information in volatile memory, parity information could be lost if the system loses power. platter—Magnetic disk medium that stores bits on its surfaces. positioning time—Access time plus latency. Positioning time is used by the SPTF strategy to order requests. preferred direction—Direction in which the disk head is moving in SCAN-based scheduling algorithms. RAID (Redundant Array of Independent Disks)—Family of techniques that use an array of disks to improve disk transfer rates while providing fault tolerance. RAID controller—Special-purpose hardware that efficiently performs operations such as dividing files into strips. forming files from strips, determining the locations of strips in the array and implementing the array's fault-tolerance mechanism. RAID level 0—RAID system that uses a striped disk array with no redundancy. Level 0 arrays are not fault tolerant: if one disk fails, all the data in the array that depend on the failed disk are lost. Depending on the array's strip size, all data stored in the array could become unusable with the loss of a single disk. Although RAID 0 is not fault tolerant, it is simple to implement, yields high transfer rates and does not incur any storage overhead. Level 0 arrays are implemented in systems where performance is more important than fault tolerance, such as supercomputers. RAID level 1—RAID system that employs disk mirroring (also called shadowing) to provide redundancy, so that each disk in the array is duplicated. Stripes are not implemented in level 1, reducing both hardware complexity and system performance. While this results in the highest degree of fault tolerance of any RAID level, only half of the array's capacity can be used to store unique data, which increases cost. Level 1 arrays are implemented in systems where high availability is more important than cost, such as database systems. Key Terms 573 RAID level 2—RAID system that is striped at the bit level (i.e., each strip stores one bit). Level 2 arrays are designed to reduce the storage overhead incurred by implementing fault tolerance using mirroring. Instead of maintaining redundant copies of each data item, RAID level 2 uses a version of Hamming error-correcting codes (Hamming ECCs) to store parity information that allows the system to detect up to two errors, correct up to one error and determine the location of the error in a stripe. The size of Hamming ECC codes, and thus the number of parity disks, increases according to the logarithm (base 2) of the number of data disks. Thus, level 2 arrays containing a large number of disks incur significantly less storage overhead than level 1 arrays. random seek pattern—Series of requests to cylinders randomly distributed across disk surfaces. FCFS causes random seek patterns that result in high response times and low throughput. RAID level 3—RAID system that stripes data at the bit or byte level. RAID 3 uses XOR (exclusive or) error-correcting codes (XOR ECCs), which use the logical XOR operation to generate parity information. XOR ECC uses only one disk to hold parity information, regardless of the size of the array. The system can use the parity bits to recover from any single disk failure. Due to parity checking, RAID level 3 reads and writes require access to the entire array. Similar to RAID level 2, this yields high transfer rates when reading and writing large files, but only one request can be serviced at a time. redundancy—Technique that maintains multiple identical resources to enable recovery upon failure. RAID level 4—RAID system that is striped using fixed-size blocks (typically larger than a byte) and uses XOR ECC to generate parity data that is stored on a single parity disk. Because level 4 arrays enable coarse-grained striping, the system can potentially service multiple read requests simultaneously, if parity is not determined for each read. When servicing a write request, however, the system must update parity information to ensure that no data is lost in the event of a disk failure. This means that write requests must be performed one at a time, creating a write bottleneck. RAID level 5—RAID system that is striped at the block level and uses XOR ECC for parity, but distributes parity blocks throughout the array of disks. Because parity blocks are distributed across many disks, multiple parity strips can be accessed simultaneously, removing the write bottleneck for many requests. Although RAID level 5 increases performance relative to RAID levels 2-A, it is complex to implement and more costly. Level 5 arrays are considered general-purpose arrays and are often found in file and application servers, enterprise resource planning (ERP) and other business systems. RAMAC (Random Access Method of Accounting and Control—First commercial hard drive produced by IBM. read-modify-write cycle (RAID) —Operation that reads a stripe, modifies its contents and parity, then writes the stripe to the array. It is performed for each write request in RAID systems that use parity. Some systems reduce the cost of read-modify-write cycles by caching strips or by updating parity information only periodically. read-write head—Moving-head disk component that hovers over a disk surface, reading and writing bits as the disk moves. Redundant Array of Independent Disks (RAID) - See RAID. reliability—Measure of fault tolerance. The more reliable a resource, the less likely it is to fail. request rate—Measure of request frequency. The higher the request rate, the greater the system load. Systems experiencing a high disk request rate tend to benefit from disk scheduling. rotational latency—Time required for a disk to rotate a requested data item from its current position to a position adjacent to the read-write head. rotational optimization—Disk scheduling technique that reduces access times by next servicing the request to the nearest sector in the read-write head's current cylinder. SCAN disk scheduling—Disk scheduling strategy that reduces unfairness and variance of response times as compared to SSTF by servicing the request that requires the shortest seek distance in a preferred direction. SCAN behaves much like SSTF in terms of high throughput and good mean response times. However, because SCAN ensures that all requests in a given direction will be serviced before the requests in the opposite direction, it offers a lower variance of response times than SSTF. Also called the elevator algorithm. sector—Smallest portion of a track that can be accessed by an I/O request. sector queuing—See rotational optimization. seek operation—Operation that moves the disk head to a different cylinder. seek time—Time it takes for the read/write head to move from its current cylinder to the cylinder containing the requested data record. 574 Disk Performance Optimization seek optimization —Disk scheduling technique that reduces seek times by generally servicing requests to the cylinder near the read-write head. shadowing (RAID)—See mirroring. shortest-access-time-first (SATF) disk scheduling—Disk scheduling strategy that next services the request that requires the shortest access time (i.e., positioning time plus transmission time). SATF exhibits higher throughput than SPTF, but large requests can be indefinitely postponed by a series of smaller requests, and requests to the innermost or outermost cylinders can be indefinitely postponed by requests to intermediate cylinders. shortest-latency-time-first (SLTF) disk scheduling—Disk scheduling strategy that examines all of the waiting requests and services the one with the shortest rotational delay first. This strategy has been shown to be close to the theoretical optimum and is relatively easy to implement. shortest-positioning-time-first (SPTF) disk scheduling—Disk scheduling strategy that next services the request that requires the shortest positioning time. SPTF results in high throughput and a low mean response time, similar to SSTF, and can also indefinitely postpone requests to the innermost and outermost cylinders. shortest-seek-time-first (SSTF) disk scheduling—Disk scheduling strategy that next services the request that is closest to the read-write head's current cylinder (and thus incurs the shortest seek time), even if that is not the first one in the queue. By reducing average seek times, SSTF achieves higher throughput rates than FCFS, and mean response times tend to be lower for moderate loads. One significant drawback is that higher variances occur on response times because of the discrimination against the outermost and innermost tracks; in the extreme, starvation of requests far from the read-write heads could occur. spindle—Moving-head disk component that spins platters at high speeds. strip (RAID)—Smallest unit of data operated on by a RAID system. The set of strips at the same location on each disk is called a stripe. stripe (RAID)—Set of strips at the same location on each disk in a RAID system. Striping enables RAID systems to access files using multiple disks at once, which improves transfer times. track—Circular region of data on a platter. Sequential file data is typically placed contiguously on one track to improve access time by reducing seeking activity. transmission time—Time required for a data record to pass by the read-write head. UNIVAC 1 (UNIVersal Automatic Computer)—First computer to introduce a magnetic storage tape. update image (RAID)—Method to reduce parity computation time by storing the difference between new and old parities in memory instead of performing a read-modifywrite cycle. write-back caching—Technique that writes buffered data to disk periodically, enabling the operating system to batch multiple I/Os that are serviced using a single request, which can improve system performance. write-through caching—Technique that writes data both to the disk cache buffer and to disk each time cached data is modified. This technique prevents the system from batching requests, but reduces the possibility of inconsistent data in the event of a system crash. XOR (exclusive-or) operation—Operation on two bits that returns 1 if the two bits are not the same, 0 otherwise. RAID levels 3-5 use the XOR operation to generate parity bits. Exercises 12.1 What are the essential goals of disk scheduling? Why is each important? 12.2 Suppose that on a particular model of disk drive, average seek times approximately equal average latency times. How might this affect a designer's choice of disk scheduling strategies? 12.3 What makes a given disk scheduling discipline fair? Just how important is fairness compared to other goals of disk scheduling disciplines? 12.4 Disk scheduling disciplines that are not fair generally have a larger variance of response times than FCFS. Why is this true? 12.5 Under light loading conditions, virtually all the disk scheduling disciplines discussed in this chapter degenerate to which scheme? Why? 12.6 One criterion that influences the desirability of implementing disk scheduling is the disk scheduling mechanism's runtime overhead. What factors contribute to this overhead? Suppose t is the average seek time on a system with FCFS disk Exercises scheduling. Suppose s is the approximate time it would take to schedule an average disk request if another form of disk scheduling were in use. Assume all other factors are favorable to incorporating disk scheduling on this system. For each of the following cases, comment on the potential effectiveness of incorporating disk scheduling on the system. 575 immediate servicing of a new request for the same cylinder as the previous request? 12.15 Why does SCAN have a lower variance of response times than SSTF? 12.16 Compare the throughput of FSCAN with that of SCAN. a. 5 = 0.01t 12.17 Compare the throughput of C-SCAN with that of SCAN. b. s = 0.1t 12.18 How does a latency optimization scheme operate? c. s = t 12.19 A disk storage bottleneck may not always be removed by adding more disk drives. Why? d. s = 10r 12.7 Latency optimization usually has little effect on system performance except under extremely heavy loads. Why? 12.8 On interactive systems it is essential to ensure reasonable response times to users. Reducing the variance of response times is still an important goal, but it is not sufficient to prevent an occasional user from suffering indefinite postponement. What additional mechanism would you incorporate into a disk scheduling discipline on an interactive system to help ensure reasonable response times and to avoid the possibility of indefinite postponement? 12.9 Argue why some people refuse to consider FCFS to be a disk scheduling discipline. Is there a scheme for servicing disk requests that we would be even less inclined to call disk scheduling than FCFS? 12.10 In what sense is LOOK fairer than SSTF? In what sense is C-LOOK fairer than LOOK? 12.11 Give a statistical argument for why FCFS offers a small variance of response times. 12.12 Argue why SSTF tends to favor requests to midrange tracks at the expense of the requests to the innermost and outermost tracks. 12.13 Chapter 7, Deadlock and Indefinite Postponement, suggests a scheme for avoiding indefinite postponement. Suggest an appropriate modification to the SSTF scheme to create a "nonstarving SSTF." Compare this new version with regular SSTF with regard to throughput, mean response times, and variance of response times. 12.14 It is possible that while a disk request for a particular cylinder is being serviced, another request for that cylinder will arrive. Some disk scheduling disciplines would service this new request immediately after processing the current request. Other disciplines preclude the servicing of the new request until the return sweep of the disk arm. What dangerous situation might occur in a disk scheduling discipline that allowed 12.20 How does the level of multiprogramming affect the need for disk scheduling? 12.21 Suppose controller saturation is indicated. Might disk scheduling be useful? What other actions should be taken? 12.22 Why is it desirable to assume uniform request distributions when considering disk scheduling disciplines? In what types of systems might you expect to observe relatively uniform request distributions? Give several examples of systems that tend to have nonuniform request distributions. 12.23 Would disk scheduling be useful for a single-user, dedicated disk in a sequential file processing application? Why? 12.24 The VSCAN scheduling strategy combines SSTF and SCAN according to a variable R.91 VSCAN determines the next request to service using SSTF, but each request to a cylinder that requires the read/write head to change direction is weighted by a factor R. The "distances" to requests in the opposite direction are computed as follows. If C is the number of cylinders on the disk and S is the number of cylinders between the current location of the head and the next request, then VSCAN uses the distance, D = S + R x C. Thus, VSCAN uses R to add an extra cost to the requests in the unpreferred direction. 12.25 If R = 0, to which disk scheduling policy does VSCAN degenerate? 12.26 If R = 1, to which disk scheduling policy does VSCAN degenerate? 12.27 Discuss the benefit of choosing a value, R, somewhere between 0 and 1. 12.28 In what circumstances might disk scheduling actually result in poorer performance than FCFS? 12.29 Compare the essential goals of disk scheduling with those of processor scheduling. What are the similarities? What are the differences? 12.30 What factors contribute to the need for seeking? 576 Disk Performance Optimization 12.31 A sequential file "properly" stored on an empty disk can be retrieved with minimal seeking. But disks tend to become heavily fragmented as files are added and deleted. Thus, successive references to "adjacent" records in a disk file can result in substantial seeking throughout the disk. Discuss the notion of file reorganization as a scheme for minimizing seeks. Under what circumstances should a file be reorganized? Should every file on the disk be reorganized or should this operation be limited to certain files only? 12.32 A designer has proposed the use of multiple disk arms to greatly improve the response times of a disk subsystem. Discuss the advantages and disadvantages of such a scheme. 12.33 We mentioned that the LOOK algorithm is like the algorithm followed by elevators in buildings. In what ways is this an appropriate analogy? 12.34 Suggest other techniques besides disk scheduling and file reorganization for minimizing seeks. 12.35 The SCAN strategy often exhibits significantly higher throughput than the C-SCAN algorithm. However, system designers sometimes choose C-SCAN over SCAN. Why might this decision be justified? 12.36 In Section 12.10.6, we said that a RAID system can reduce the number of I/O operations when calculating parity by using the formula Ap' = (Ad XOR Ad') XOR Ap. Prove that this relation holds for all strips Ad,, Ad' and Ap. 12.37 For each of the following applications, indicate whether the system should use RAID level 0, RAID level 1, or RAID level 5 and explain why. 12.38 Storing account transactions for a financial institution 12.39 Storing high-resolution telescope images (that have been transferred from a tape drive) before processing on a supercomputer 12.40 Running a personal Web server Suggested Projects 12.41 Research disk scheduling in Linux. What scheduling strategies does Linux use? How are these strategies implemented? For an introduction to Linux disk scheduling, see Section 20.8.3, Block Device I/O. 12.42 Prepare a research paper discussing the techniques that hard disk manufacturers employ to improve performance. See the links in the Web Resources section. 12.43 Prepare a report briefly describing five RAID levels other than the 6 covered in detail in this chapter. Suggested Simulations 12.44 Develop monitors (see Chapter 6, Concurrent Programming) to implement each of the following disk scheduling disciplines. a. FCFS b. SSTF c. SCAN d. N-Step SCAN e. C-SCAN 12.45 In this chapter, we have investigated several cylinderoriented disk scheduling strategies including FCFS, SSTF, SCAN, TV-Step SCAN, and C-SCAN, F-SCAN, LOOK and CLOOK. Write a simulation program to compare the performance of these scheduling methods under light, medium, and heavy loads. Define a light load as one in which there is usually either 0 or 1 requests waiting for the entire disk. Define a medium load as one in which each cylinder has either 0 or 1 request waiting. Define a heavy load as one in which many cylinders are likely to have many requests pending. Use random number generation to simulate the arrival time of each request as well as the actual cylinder for each request. 12.46 The availability of moving-head disk storage has contributed greatly to the success of modern computing systems. Disks offer relatively fast and direct access to enormous amounts of information. Industry forecasters see a solid future for disk-based secondary storage systems, and researchers continue developing new and improved disk scheduling disciplines. Propose several new disk scheduling schemes and compare them with those presented in this chapter. Develop a program that simulates a disk that contains two platters (i.e., four surfaces), four read/write heads, 25 cylinders and 20 sectors per track. This disk should store the current position of the read/write head and provide functions such as read, write and seek. These functions should return the number of milliseconds required to service each request (track-to-track seek times of 2 ms, average seek times of 10 ms and average latencies of 5 ms are reasonable). Then create a simulation program that generates a uniform distribution of requests specifying a location by platter, cylinder and sector and a size by the number of requested sectors. These requests Recommended Reading 577 should be sent to your disk scheduler, which then orders them and sends them to the disk for service by calling its read, write and seek functions. Use the values returned by the disk to determine each scheduling algorithm's throughput, response time and variance of response times. 12.47 Extend the simulation in the previous exercise to incorporate rotational latency. For each strategy, did performance increase or decrease as a result? Recommended Reading Because disks are a bottleneck in most systems, disk performance optimization continues to be an active area of research and analysis in the literature. Denning discussed how disk scheduling algorithms can improve performance.92 As hard disks became the dominant form of secondary storage, research into improving their performance continued.93, 94, 95, 96, 97, 98, 99 Researchers regularly perform comparative analyses of disk scheduling algorithms, often using sophisticated simulations of hard disk activity.36-39 Patterson et al. introduced the notion of RAID in their seminal paper;100 subsequent research has further explored the trade-offs between various RAID levels.101-102 As the gap between main memory and hard disk speeds grows at an increasing pace, microelectromechanical storage (MEMS) devices, which access a rectangular media using chips containing thousands of read/write heads, show promise as a technology to replace the hard disk as high-capacity, low-cost storage. 103104 The bibliography for this chapter is located on our Web site at www.deitel.com/books/os3e/Bibliography.pdf. Works Cited i. Kozierok, C, "Life Without Hard Disk Drives," PCGuide, April 17,2001, . 2. Khurshudov, A., The Essential Guide to Computer Data Storage, Upper Saddle River, NJ: Prentice Hall PTR., 2001, p. 6. 3. "RAMAC," Whatis.com, April 30, age.techtarget.com/sDefinition/ 2001, . 4. Khurshudov, A., The Essential Guide to Computer Data Storage, Upper Saddle River, NJ: Prentice Hall PTR, 2001, p. 90. 5. "Cost of Hard Drive Space," September 20, 2002, . 6. Patterson, D.; G. Gibson; and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," Proceedings of the ACM SIGMOD, June 1998, p. 109. 7. Gotlieb, C, and G. MacEwen, "Performance of Movable-Head Disk Storage Devices," Journal of the ACM, Vol. 20, No. 4, October 1973, pp. 604-623. 8. Smith, A., "On the Effectiveness of Buffered and Multiple Arm Disks," Proceedings of the Fifth Symposium on Computer Architecture, 1978, pp. 109-112. 9. Pechura, M., and J. Schoeffler, "Estimated File Access Time of Floppy Disks," Communications of the ACM, Vol. 26, No. 10, October 1983, pp. 754-763. 10. Kozierok, C, "Spindle Speed," PCGuide, April 17, 2001, . 13. Kozierok, C, "Hard Disk Tracks, Cylinders and Sectors," modified April 17, 2001, . 14. "DiamondMax Plus 9," Maxtor Corporation, May 14, 2003, . 15. "WD Caviar Hard Drive 250GB 7200 RPM," October 28, 2003, . 16. "MK8025GAS," Toshiba America, Inc., October 28, 2003, . 17. "WD Raptor Enterprise Serial ATA Hard Drive 36.7 GB 10,000 RPM," October 28, 2003, . 18. "Cheetah 15K.3," Seagate Technology, March 27, 2003, . 19. Wilhelm, N., "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Accesses," Communications of the ACM, Vol. 19, No. 1, January 1976, pp. 13-17. 11. Walters, S., "Estimating Magnetic Disc Seeks," Computer Journal, Vol. 18, No. 1,1973, pp. 412-416. 20. Wong, C, "Minimizing Expected Head Movement in OneDimensional and Two-Dimensional Mass Storage Systems," ACM Computing Survey's, Vol. 12, No. 2,1980, pp. 167-178. 12. Kollias, J., "An Estimate of Seek Time for Batched Searching of Random or Indexed Sequential Structured Files," Computer Journal,Yol 21, No. 2,1978, pp. 21-26. 21. Frank, H, "Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems," Journal of the ACM, Vol. 16, No. 4, October 1969, pp. 602-620. 578 Disk Performance Optimization 22. Teorey, T., "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," Proceedings of AFIPS FJCC, Vol. 41,1972, pp. 1-11. 23. Wilhelm, N, "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Access," Communications of the ACM, Vol. 19, No. 1, January 1976, pp. 13-17. 24. Hofri, M., "Disk Scheduling: FCFS vs. SSTF Revisited," Communications of the ACM, Vol. 23, No. 11, November 1980, pp. 645653. 25. Denning, P., "Effects of Scheduling on File Memory Operations," Proceedings of AFIPS, SJCC, Vol. 30,1967, pp. 9-21. 26. Teorey, T, "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," Proceedings of AFIPS FJCC, Vol. 41,1972, pp. 1-11. 27. Wilhelm, N., "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Access," Communications of the ACM, Vol. 19, No. 1, January 1976, pp. 13-17. 28. Hofri, M., "Disk Scheduling: FCFS vs. SSTF Revisited," Communications of the ACM, Vol. 23, No. 11, November 1980, pp. 645653. 29. Denning, P., "Effects of Scheduling on File Memory Operations," Proceedings ofAFIPS, SJCC,Vol. 30,1967, pp. 9-21. 30. Teorey, T, "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," Proceedings ofAFIPS FJCC, Vol. 41,1972, pp. 1-11. 31. Gotlieb, C, and G. MacEwen, "Performance of Movable-Head Disk Storage Devices," Journal of the ACM, Vol. 20, No. 4, October 1973, pp. 604-623. 32. Teorey, T, and T. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies," Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 177-184. 33. Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Lookahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2, September 2002, pp. 36. 34. Teorey, T, "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," Proceedings of AFIPS FJCC, Vol. 41,1972, pp. 1-11. 35. Teorey, T, and T. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies," Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 177-184. 36. Worthington, B.; G. Ganger; and Y. Patt, "Scheduling Algorithms for Modern Disk Drives," Proceedings of the 1994 ACM SIGMETRICS Conference, May 1994, p. 243. 37. "Hard Disk Specifications," viewed October 2,2003, . 38. Stone, H., and S. Fuller, "On the Near Optimality of the ShortestLatency-Time-First Drum Scheduling Discipline," Communications oftheACM,\ol. 16, No. 6, June 1973, pp. 352-353. 39- Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Look-Ahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 1. September 2002, p. 33. 40. Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Look-Ahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2. September 2002, p. 33. 4 1 . Kozierok, C, "Logical Geometry," PCGuide, April 17, 2001. . 42. Khurshudov, A., The Essential Guide to Computer Data Storage (Upper Saddle River, NJ: Prentice Hall PTR..2001), pp. 106-107. 43. " Dell - Learn More - Hard Drives," 2003, . 44. Chaney, R., and B. Johnson, "Maximizing Hard-Disk Performance: How Cache Memory Can Dramatically Affect Transfer Rate," Byte, May 1984, pp. 307-334. 45. Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Look-Ahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2. September 2002, p. 38. 46. Wilhelm, N, "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Accesses," Communications of the ACM, Vol. 19, No. 1, January 1976, pp. 13-17. 47. Lynch, W., "Do Disk Arms Move?" Performance Evaluation Review, ACM Sigmetrics Newsletter, Vol. 1, December 1972, pp. 3-16. 48. Kozierok, C, "Cache Write Policies and the Dirty Bit," PCGuide. April 17, 2001, . 49. Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Look-Ahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2. September 2002, p. 32. 50. Kozierok, C, "Operating System and Controller Disk Caching." PCGuide, April 17, 2001, <www.pcguide.com/refhdd/perf/perf/extp/pcCaching-c.html>. 51. Thomasian, A., and C. Liu, "Special Issue on the PAPA 2002 Workshop: Disk Scheduling Policies with Look-Ahead," ACM SIGMETRICS Performance Evaluation Review, Vol. 30, No. 2. September 2002, p. 31. 52. "Disk Optimization Can Save Time and Resources in a Windows NT/2000 Environment," Raxco Software, . 53. King, R., "Disk Arm Movement in Anticipation of Future Requests," ACM Transactions in Computer Systems, Vol. 8, No. 3. August 1990, p. 215. 54. King, R., "Disk Arm Movement in Anticipation of Future Requests," ACM Transactions in Computer Systems, Vol. 8, No. 3, August 1990, p. 214. Works Cited 579 55- King, R., "Disk Arm Movement in Anticipation of Future Requests," ACM Transactions in Computer Systems, Vol. 8, No. 3, August 1990, pp. 220,226 75- Patterson, D.; G. Gibson; and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," Proceedings of the ACM SIGM O A June 1988, p. 113. 56. "RAID Overview," raid_overview.html>. 76. Patterson, D.; G. Gibson; and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," Proceedings of the ACM S1GMOD, June 1988, p. 113. . . . . 81. Wilson, R., "RISC Chips Explore Parallelism for Boost in Speed," Computer Design, January 1,1989, pp. 58-73. 82. Davidson, J. W., and R. A. Vaughan, "The Effect of Instruction Set Complexity on Program Size and Memory Performance," Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systerns, 1987, pp. 60-64. Works Cited 681 S3. Wilson, R., "RISC Chips Explore Parallelism for Boost in Speed," Computer Design, January 1,1989, pp. 58-73. 84. "IA-32 Intel Architecture Software Developer's Manual," Vol. 1: Basic Architecture, 2002. 85. Brehob, M., et al., "Beyond RISC - The Post-RISC Architecture," Michigan State University Department of Computer Science, Technical Report CPS-96-11, March 1996. 86. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvcl.html>. 87. Stokes, J., "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvcl.html>. 88. Brehob, M., et al., "Beyond RISC - The Post-RISC Architecture," Michigan State University Department of Computer Science, Technical Report CPS-96-11, March 1996. 89. DeMone, P., "RISC vs. CISC Still Matters," February 13, 2000, . 100. Stokes, J., "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 101. "Altivec Fact Sheet," Motorola Corporation, www.motorola.com/files/32bit/doc/fact_sheet/ ALTIVECCLANCE.pdf>. 2002, . 103. Schlansker, M., and B. Rau, "EPIC: Explicitly Parallel Instruction Computing," Computer, Vol. 33, No. 2, February 2000, pp. 37-38. 104. Lowney, P., et al., "The Multiflow Trace Scheduling Compiler," Journal of Supercomputing, Vol. 7, May 1993:51-142. 105. Hennessy, J., and D. Patterson, Computer Organization and Design, San Francisco, CA: Morgan Kaulmann Publishers, Inc., 1998, p. 528. 106. Schlansker, M., and B. Rau, "EPIC: Explicitly Parallel Instruction Computing," Computer, Vol. 33, No. 2, February 2000, pp. 38-39. 90. Hennessy, J., and D. Patterson, Computer Organization and Design, San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1998, p. 510. 107. Schlansker, M., and B. Rau, "EPIC: Explicitly Parallel Instruction Computing," Computer, Vol. 33, No. 2, February 2000, pp. 41-44. 91. Jouppi, N., "Superscalar vs. Superpipelined Machines," ACM SIGARCH Computer Architecture News, Vol. 16, No. 3, June 1988, pp. 71-80. 108. Schlansker, M., and B. Rau, "EPIC: Explicitly Parallel Instruction Computing," Computer, Vol. 33, No. 2, February 2000, pp. 41-44. 92. Ray, X; X Hoe; and B. Falsafi, "Dual Use of Superscalar Datapath for Transient-Fault Detection and Recovery," Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture, December 2001, pp. 214-224. 109. Lucas, H., "Performance Evaluation and Monitoring," ACM Computing Surveys, Vol. 3, No. 3, September 1971, pp. 79-91. 93. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 94. Hwu, W. W., and Y. Patt, "Checkpoint Repair for Out-of-Order Execution Machines," Proceedings of the 14th International Symposium on Computer Architecture, June 1987, pp. 18-26. no . Mogul, X, "Brittle Metrics in Operating System Research," Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, March 1999, pp. 90-95. in . Seltzer, M., et al., "The Case for Application-Specific Benchmarking," Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, March 1999, pp. 105-106. 112. "PC Magazine Benchmarks from VeriTest," PC Magazine . 95. Chuang, W., and B. Calder, "Predicate Prediction for Efficient Out-of-Order Execution," Proceedings of 17th Annual International Conference on Supercomputing, June 2003, pp. 183-192. 113. Hennessy, J., and D. Patterson, Computer Organization and Design, San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1998, p. 528. 96. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 114. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 97. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 115. Aletan, S., "An Overview of RISC Architecture," Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's, 1992, pp. 11-12. 98. Young, C, and M. Smith, "Static Correlation Branch Prediction," ACM Transactions on Programming Languages and Systems, Vol. 21, No. 5, September 1999, pp. 1028-1075. 116. Flynn, M. X; C. L. Mitchell; and X M. Mulder, "And Now a Case for More Complex Instruction Sets," Computer, Vol. 20, No. 9, September 1987, pp. 71-83. 99. Stokes, X, "Ars Technica: RISC vs. CISC in the Post RISC Era," October 1999, <arstechnica.com/cpu/4q99/risc-cisc/rvc1.html>. 117. Schlansker, M., and Rau, B., "EPIC: Explicitly Parallel Instruction Computing," Computer, Vol. 33, No. 2, February 2000, pp. 41-44. What's going to happen in the next decade is that we'll figure out how to make parallelism work. - David Kuck quoted in TIME,March 28,1988- "The question is," said Humpty Dumpty, "whish is to be master - that's all." — Lewis Carroll — I hope to see my Pilot face to face When I have crossed the bar. — Alfred, Lord Tennyson — "If seven maids with seven mops Swept it for half a year, Do you suppose," the Walrus said, "That they could get it clear?" —Lewis Carroll— The most general definition of beauty ... Multeity in Unity. —Samuel Taylor Coleridge— Chapter 15 Multiprocessor Management Objectives After reading this chapter, you should understand: • multiprocessor architectures and operating system organizations. • multiprocessor memory architectures. • design issues specific to multiprocessor environments. • algorithms for multiprocessor scheduling. • process migration in multiprocessor systems. • load balancing in multiprocessor systems. • mutual exclusion techniques for multiprocessor systems. Chapter Outline 15.1 Introduction 15.5 Multiprocessor Memory Sharing Mini Case Study: Supercomputers Biographical Note: Seymour Cray Operating Systems Thinking: Data 15.2 15.5.1 Cache Coherence 15.5.2 Page Replication and Migration 15.2.1 Classifying Sequential and Parallel Architectures 15.2.2 Processor Interconnection 15.5.3 Shared Virtual Memory Multiprocessor Architecture Schemes 15.2.3 Loosely Coupled vs. Tightly Coupled Systems 15.3 Replication and Coherency 15.6 Multiprocessor Scheduling 15.6.1 Job-Blind Multiprocessor Scheduling 15.6.2 Job-Aware Multiprocessor Scheduling Multiprocessor Operating System Organizations 15.3.1 Master/Slave 15.3.2 Separate Kernels 15.3.3 Symmetrical Organization Operating Systems Thinking: Graceful Degradation 15.7 Process Migration 15.7.1 Flow of Process Migration 15.7.2 Process Migration Concepts 15.7.3 Process Migration Strategies 15.4 Memory Access Architectures 15.4.1 Uniform Memory Access 15.4.2 Nonuniform Memory Access 15.4.3 Cache-Only Memory Architecture 15.4.4 No Remote Memory Access 684 15.8 Load Balancing 15.8.1 Static Load Balancing 15.8.2 Dynamic Load Balancing 15.9 Multiprocessor Mutual Exclusion 15.9.1 Spin Locks 15.9.2 Sleep/Wakeup Locks 15.9.3 Read/Write Locks Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited 685 686 Multiprocessor Management 15.1 Introduction For decades, Moore's Law has successfully predicted an exponential rise in processor transistor count and performance, yielding ever more powerful processors. Despite these performance gains, researchers, developers, businesses and consumers continue to demand substantially more computing power than one processor can provide. As a result, multiprocessing systems—computers that contain more than one processor —are employed in many computing environments. Large engineering and scientific applications that execute on supercomputers increase throughput by processing data in parallel on multiple processors. Businesses and scientific institutions use multiprocessing systems to increase system performance, scale resource usage to application requirements and provide a high degree of data reliability.1 For example, the Earth Simulator in Yokohama, Japan — the most powerful supercomputer as of June, 2003 —contains 5,f20 processors, each operating at 500MHz (see the Mini Case Study, Supercomputers). The system can Mini Case Study Supercomputers Supercomputer is simply a term for the most powerful contemporary computers. The early supercomputers would be no match for today's inexpensive PCs. The speed of a supercomputer is measured in Flops (Floating-point operations per second). The first supercomputer was the CDC 6600, manufactured by Control Data Corporation and designed by Seymour Cray, who became known as the father of supercomputing (see the Biographical Note, Seymour Cray).2, 3 This machine, released in the early 1960's, processed 3 million instructions per second. 4, 5 It was also the first computer to use the RISC (reduced instruction set comput- ing) architecture. 6 ' 7 Roughly 10 years and several machines later, Cray designed the Cray-1, one of the earliest vector-processor models.8 (Vector processors are discussed in Section 15.2.1, Classifying Sequential and Parallel Architectures.) The supercomputers that Cray designed dominated the high-performance field for many years. According to the supercomputer tracking organization Top500 (www.top500.org), the fastest computer in the world (at the time this book was published) is NEC's Earth Simulator, located in Japan. It operates at a peak speed of roughly 35 teraflops, more than twice the speed of the sec- ond-fastest computer and tens of thousands of times faster than a typical desktop machine. 9, 10 The Earth Simulator consists of 5,120 vector processors grouped into 640 units of eight processors each, where each unit has 16GB of shared main memory, totalling 10TB of memory.11, 12 These units are connected by 1500 miles of network cable and are linked to a 640TB disk system and a tape library that can hold over 1.5 petabytes (1,500,000,000,000,000 bytes).13, 14 This enormous computing capacity is used for research in predicting environmental conditions and events. 15 15.1 Introduction 687 execute 35.86 Tflops (trillion floating point operations per second), enabling researchers to model weather patterns, which can be used to predict natural disasters and evaluate how human activities affect nature. 16, 17 Multiprocessing systems must adapt to different workloads. In particular, the operating system should ensure that • all processors remain busy • processes are evenly distributed throughout the system • execution of related processes is synchronized • processors operate on consistent copies of data stored in shared memory • mutual exclusion is enforced. Techniques used to solve deadlock in multiprocessing and distributed systems are similar and are discussed in Chapter 17, Introduction to Distributed Systems. This chapter describes multiprocessor architectures and techniques for optimizing multiprocessing systems. These techniques focus on improving performance, Biographical Note Seymour Cray Seymour Cray is known as the father of supercomputing. Shortly after getting his Masters in Applied Mathematics from the University of Minnesota in 1951, he joined Engineering Research Associates (ERA), one of the first companies to build digital computers. 18, 19, 21 At ERA Cray designed the 1103, the first computer designed for scientific research that also sold well to the general market. 20, 22 In 1957, Cray left the company with William Norris (the founder of ERA) to start Control Data Corporation. 2 3 2 4 - 2 5 By 1960 Cray had designed the CDC 1604, the first computer that com- pletely replaced vacuum tubes with transistors. 26, 27, 28 In 1962 Cray designed the CDC 6600, the first true "supercomputer," followed by the CDC 7600.29, 30, 31 He always strove for simplicity, so the CDC 6600 featured a simplified architecture based on a smaller set of instructions—a design philosophy later called RISC (reduced instruction set computing). 32, 33, 34 Seymour Cray founded Cray Research in 1972 to continue building large scientific computers.35 The Cray-1, one of the earliest vector-processor computers, was released in 1976 (see Section 15.2.1, Classifying Sequential and Parallel Architectures). 36 The Cray-1 was much smaller yet more powerful than the CDC 7600. 37 It was also visually interesting, curving around to form a cylinder with a gap, and equipped with a bench running around the outside. 38 It ran at 133 megaflops, which was extraordinarily fast at that time, but slower than today's typical desktop PCs.39 The Cray-1 was followed by three more Cray models, each progressively faster and more compact than the last. The computational unit (not including the memory banks and cooling equipment) of the Cray-4 was smaller than the human brain. 40 688 Multiprocessor Management fairness, cost and fault tolerance. Often, improvement in one of these parameteroccurs at the expense of the others. We also consider how design decisions can affect multiprocessor performance. Self Review 1. Why are multiprocessors useful? 2. How do the responsibilities of a multiprocessor operating system differ from those of a uniprocessor system? Ans: 1) Many computer users demand more processing power than one processor can provide For example, businesses can use multiprocessors to increase performance and scale resource usage to application needs. 2) The operating system must balance the workload of various processors, enforce mutual exclusion in a system where multiple processes can execute truly simultaneously and ensure that all processors are made aware of modifications to shared memory. 15.2 Multiprocessor Architecture The term "multiprocessing system" encompasses any system containing more than one processor. Examples of multiprocessors include dual-processor personal computers, powerful servers that contain many processors and distributed groups of workstations that work together to perform tasks. Throughout this chapter we present several ways to classify multiprocessors. In this section, we categorize multiprocessing systems by their physical properties, such as the nature of the system's datapath (i.e., the portion of the processor that performs operations on data), the processor interconnection scheme and how processors share resources. 15.2.1 Classifyeing Sequential and Parallel Architectures Flynn developed an early scheme for classifying computers into increasingly parallel configurations. The scheme consists of four categories based on the different types of streams used by processors.41 A stream is simply a sequence of bytes that is fed to a processor. A processor accepts two streams —an instruction stream and a data stream. Single-instruction-stream, single-data-stream (SISD) computers are the simplest type. These are traditional uniprocessors in which a single processor fetches one instruction at a time and executes it on a single data item. Techniques such as pipelining, very long instruction word (VLIW) and superscalar design can introduce parallelism into SISD computers. Pipelining divides an instruction's execution path into discrete stages. This allows the processor to process multiple instructions simultaneously, as long as at most one instruction occupies each stage during a clock cycle. VLIW and superscalar techniques simultaneously issue multiple independent instructions (from one instruction stream) that execute in different execution units. VLIW relies on the compiler to determine which instructions to issue at any given clock cycle, whereas superscalar design requires that the processor make this decision.42 Additionally, Intel's Hyper-Threading technology introduces parallelism by creating two virtual processors from one physical processor. This gives a multiprocessor- 15.2 Multiprocessor Architecture 689 enabled operating system the impression that it is running on two processors that each execute at a little less than half the speed of the physical processor.43 Multiple-instruction-stream, single-data-stream (MISD) computers are not commonly used. A MISD architecture would have several processing units that act on a single stream of data. Each unit would execute a different instruction on the data and pass the result to the next unit.44 Single-instruction-stream, multiple-data-stream (SIMD) computers issue instructions that act on multiple data items. A SIMD computer consists of one or more processing units. A processor executes a SIMD instruction by performing the same instruction on a block of data (e.g., adding one to every element in an array). If there are more data elements than processing units, the processing units fetch additional data elements for the next cycle. This can increase performance relative to SISD architectures, which would require a loop to perform the same operation ome data element at a time. A loop contains many conditional tests, requires the SISD processor to decode the same instruction multiple times and requires the SISD processor to read data one word at a time. By contrast, SIMD architectures read in a block of data at once, reducing costly memory-to-register transfers. SIMD architectures are most effective in environments in which a system applies the same instruction to large data sets.45, 46 Vector processors and array processors use a SIMD architecture. A vector processor contains one processing unit that executes each vector instruction on a set of data, performing the same operation on each data element. Vector processors rely on extremely deep pipelines (i.e., ones containing many stages) and high clock speeds. Deep pipelines permit the processor to perform work on several instructions at a time so that many data elements can be manipulated at once. An array processor contains several processing units that perform the same instruction in parallel on many data elements. Array processors (often referred to as massively parallel processors) might contain tens of thousands of processing elements. Therefore, array processors are most efficient when manipulating large data sets. Vector and array processing are useful in scientific computing and graphics manipulation where the same operation must be applied to a large data set (e.g., matrix transformations).47, 48 The Connection Machine (CM) systems built by Thinking Machines, Inc., are examples of array processors.49 Multiple-instruction-stream, multiple-data-stream (MIMD) computers are multiprocessors in which the processing units are completely independent and operate on separate instruction streams.50 However, these systems typically contain hardware that allows processors to synchronize with each other when necessary, such as when accessing a shared peripheral device. Self Review 1. Thread-level parallelism (TLP) refers to the execution of multiple independent threads in parallel. Which multiprocessor architecture exploits TLP? 2. (T/F) Only SIMD processor architectures exploit parallelism. 690 Multiprocessor Management Ans: 1) Only MIMD and MISD architectures can execute multiple threads at once. However, the threads executed by a MISD computer manipulate the same data and are not independent. Therefore, only MIMD systems can truly exploit TLP. 2) False. SISD systems use techniques such as pipelines, VLIW and superscalar design to exploit parallelism; MISD processors execute multiple threads at once; and MIMD processors exploit TLP as described in the previous answer. 15.2.2 Processor Interconnection Schemes The interconnection scheme of a multiprocessor system describes how the system's components, such as processors and memory modules, are physically connected. The interconnection scheme is a key issue for multiprocessor designers because it affects the system's performance, reliability and cost. An interconnection scheme consists of nodes and links. Nodes are composed of system components and/or switches that route messages between components. A link is a connection between two nodes. In many systems, a single node might contain one or more processors, their associated caches, a memory module and a switch. In large-scale multiprocessors, we sometimes abstract the concept of a node and denote a group of nodes as a single supernode. Designers use several parameters to evaluate interconnection schemes. A node's degree is the number of nodes to which it is connected. Designers try to minimize a node's degree to reduce the complexity and cost of its communication interface. Nodes with larger degrees require more complex communication hardware to support communication between the node and its neighbor nodes (i.e., nodes connected to it). 51 One technique for measuring an interconnection scheme's fault tolerance is to count the number of communication links that must fail before the network is unable to function properly. This can be quantified using the bisection width—the minimum number of links that need to be severed to divide the network into two unconnected halves. Systems with larger bisection widths are more fault tolerant than those with smaller bisection widths because more components must fail before the entire system fails. An interconnection scheme's performance largely depends on communication latency between nodes. This can be measured in several ways, one of which is the average latency. Another performance measure is the network diameter—the shortest distance between the two most remote nodes in the interconnection scheme. To determine the network diameter, consider all pairs of nodes in the network and find the shortest path length for each pair—computed by totaling the number of links traversed—then find the largest of these paths. A small network diameter indicates low communication latency and higher performance. Finally, system architects attempt to minimize the cost of an interconnection scheme, which is equal to the total number of links in a network. 52 In the following subsections, we enumerate several successful interconnection models and evaluate them based on the preceding criteria. Many real systems implement variations of these models. For example, they might add extra communi- 15.2 Multiprocessor Architecture 691 cation links to increase fault tolerance (by increasing the bisection width) and performance (by decreasing the network diameter). Shared Bus The shared bus network organization uses a single communication path (i.e., route through which messages travel) between all processors and memory modules (Fig. 15.1).53 The components' bus interfaces handle transfer operations. The bus is

passive, and the components arbitrate among themselves to use the bus. Only one transfer can take place at a time on the bus, because the bus cannot convey two electrical signals at once. Therefore, before a component initiates a transfer, it must first check that both the bus and the destination component are available. One problem with shared buses—contention—arises when several components wish to use the bus at once. To reduce contention and bus traffic, each processor maintains its own local cache as shown in Fig. 15.1. When the system can fulfill a memory request from a processor's cache, the processor does not need to communicate over the bus with a memory module. Another option is to construct a multiple shared bus architecture, which reduces contention by providing multiple buses that service communication requests. However, these require complex bus arbitration logic and additional links, which increases the system's cost.54-55 The shared bus is a simple and inexpensive scheme to connect a small number of processors. New components can be added to the system by attaching them to the bus, and software handles the detection and identification of the bus components. However, due to contention for the single communication path, shared bus organizations do not scale beyond a small number of processors (in practice 16 or 32 is the maximum). 56 Contention is exacerbated by the fact that processor speed has increased faster than bus bandwidth. As processors get faster, it takes fewer processors to saturate a bus. Shared buses are dynamic networks, because communication links are formed and discarded (through the shared bus) during execution. Therefore, the criteria used to evaluate interconnection schemes that we discussed earlier do not apply; these criteria are based on static links, which do not change during execution. How- Figure 15.1 | Shared bus multiprocessor organization. 692 Multiprocessor Management ever, compared to other interconnection schemes, a shared bus with several processors is fast and inexpensive, but not particularly fault tolerant—if the shared bus fails, the components cannot communicate. 57 Designers can leverage the benefits of shared buses in multiprocessors with larger numbers of processors. In these systems, maintaining a single shared bus (or several buses) that connects all processors is impractical, because the bus becomes saturated easily. However, designers can divide the system's resources (e.g., processors and memory) into several small supernodes. The resources within a supernode communicate via a shared bus, and the supernodes are connected using one of the more scalable interconnection schemes described in the sections that follow. Such systems attempt to keep most communication traffic within a supernode to exploit the fast bus architecture, while enabling communication between supernodes. 58 Most multiprocessors with a small number of processors, such as dual-processor Intel Pentium systems, use a shared bus architecture. 59 Crossbar-Switch Matrix A crossbar-switch matrix provides a separate path from every processor to even memory module (Fig. 15.2).60 For example, if there are n processors and m memory modules, there will be n x m total switches that connect each processor to each memory module. We can imagine the processors as the rows of a matrix and the memory modules as the columns. In larger networks, nodes often consist of processor and memory components. This improves memory-access performance (for those accesses between a processor and its associated memory module). In the case of a crossbar-switch matrix, this reduces the interconnection scheme's cost. In this design, each node connects to a switch of degree of p - 1, where p is the number of processor-memory nodes in the system (i.e., in this case, m - n because each node contains the same number of processors and memory modules). 61, 62 A crossbar-switch matrix can support data transmissions to all nodes at once, but each node can accept at most one message at a time. Contrast this with the shared bus, which supports only one transmission at a time. A switch uses an arbitration algorithm such as "service the requesting processor that has been serviced least recently at this switch" to resolve multiple requests. The crossbar-switch design provides high performance. Because all nodes are linked to all other nodes and transmission through switch nodes has a trivial performance cost, the network diameter is essentially one. Each processor is connected to each memory module, so to divide a crossbar-switch matrix into two equal halves, half of the links between processors and memory modules must be cut. The number of links in the matrix is the product of n and m, so the bisection width is (n x m) / 2, yielding strong fault tolerance. As Fig. 15.2 shows, there are many paths a communication can take to reach its destination. A disadvantage of crossbar-switch matrices is their cost, which increases proportionally to nxm, making large-scale systems impractical.63 For this reason, crossbar-switches are typically employed in smaller multiprocessors (e.g., 16 processors). However, as hardware costs decline, they are being used more frequently in larger systems. The Sun UltraSPARC-Ill uses crossbar switches to share memory.64 15.2 Multiprocessor Architecture 693 Figure 15.2 | Crossbar-switch matrix multiprocessor organization. 2-D Mesh Network In a 2-D mesh network interconnection scheme, each node consists of one or more processors and a memory module. In the simplest case (Fig. 15.3), the nodes in a mesh network are arranged in a rectangle of n rows and m columns, and each node is connected with the nodes directly north, south, east and west of it. This is called a 4-connected 2-D mesh network. This design keeps each node's degree small, regardless of the number of processors in a system—the corner nodes have a degree of two, the edge nodes have a degree of three and the interior nodes have a degree of four. In Fig. 15.3, where n = 4 and m = 5, the 2-D mesh network can be divided into two equal halves by cutting the five links between the second and third rows of nodes. In fact, if n is even and m is odd the bisection width is m + 1 if m > n and is n otherwise. If the 2-D mesh contains an even number of rows and columns, the bisection width is the Figure 15.3 | 4-connected 2-D mesh Network. 694 Multiprocessor Management smaller of m and n. Although not as fault tolerant as a crossbar-switch matrix, a 2-D mesh network is more so than other simple designs, such as a shared bus. Because the maximum degree of a node is four, a 2-D mesh network will have a network diameter that is too substantial for large-scale systems. However, mesh networks have been used in large systems in which communication is kept mainly between neighboring nodes. For example, the Intel Paragon multiprocessor uses a 2-D mesh network.65 Hypercube An n-dimensional hypercube consists of 2n nodes, each linked with n neighbor nodes. Therefore, a two-dimensional hypercube is a 2 x 2 mesh network, and a three-dimensional hypercube is conceptually a cube.66 Figure 15.4 illustrates connections between nodes in three-dimensional (part a) and four-dimensional (part b) hypercubes. 67 Note that a three-dimensional hypercube is actually a pair of twodimensional hypercubes in which the corresponding nodes in each two-dimensional hypercube are connected. Similarly, a four-dimensional hypercube is actually a pair of three-dimensional hypercubes in which the corresponding nodes in each threedimensional hypercube are connected. A hypercube's performance scales better than that of a 2-D mesh network, because each node is connected by n links to other nodes. This reduces the network diameter relative to a 2-D mesh network. For example, consider a 16-node multiprocessor implemented as either a 4 x 4 mesh network or a 4-dimensional hypercube. A 4 x 4 mesh network has a network diameter of 6, whereas a 4-dimensional hypercube has a network diameter of 4. In some hypercubes, designers add communication links between nonneighbor nodes to further reduce the network diameter.68 A hypercube's fault tolerance also compares favorably with that of other designs. However, the increased number of links per node increases a hypercube's cost relative to that of a mesh network. 69 The hypercube interconnection scheme is efficient for connecting a modest number of processors and is more economical than a crossbar-switch matrix. The nCUBE system used in streaming media and digital-advertising systems employs hypercubes of up to 13 dimensions (8,192 nodes). 70 F i g u r e 15.4 | 3- and 4-dimensional hypercubes. 15.2 Multiprocessor Architecture Multistage Networks An alternative processor interconnection scheme is a multistage network.71 As in the crossbar-switch matrix design, some nodes are switches rather than processor nodes with local memory. Switch nodes are smaller, simpler and can be packed more tightly, increasing performance. To understand the benefits of a multistage network over a crossbar-switch matrix, consider the problem of flying between small cities. Instead of offering direct flights between every pair of cities, airlines use large cities as "hubs." A flight between two small cities normally consists of several "legs" in which the traveller first flies to a hub, possibly travels between hubs and finally travels to the destination airport. In this way, airlines can schedule fewer total flights and still connect any small city with an airport to any other. The switching nodes in a multistage network act as hubs for communication between processors, just as airports in large cities do for airlines. There are many schemes for constructing a multistage network. Figure 15.5 shows a popular multistage network called a baseline network.72 Each nodes on the left is the same as the node on the right. When one processor wants to communicate with another, the message travels through a series of switches. The leftmost switch corresponds to the least significant (rightmost) bit of the destination processor's ID; the middle switch corresponds to the middle bit; and the rightmost switch corresponds to the most significant (leftmost) bit. Figure 15.5 | Multistage baseline network. 695 696 Multiprocessor Management For example, consider the bold path in Fig. 15.5 from the processor with ID 001 to the processor with ID 110. Begin by following the path from processor 001 to the switch. Next, determine the least significant bit (i.e., 0) in the destination processor's ID (110) and follow that path from the current switch to the next switch. From the second switch, follow the path that corresponds to the middle bit (i.e.. 1) in the destination processor's ID. Finally, from the third switch, follow the path that corresponds to the most significant bit (i.e., 1) in the destination processor's ID. Multistage networks represent a cost-performance compromise. This design employs simple hardware to connect large numbers of processors. Any processor can communicate with any other without routing the message through intermediate processors. However, a multistage network has a larger network diameter, so communication is slower than with a crossbar-switch matrix—each message must pass through multiple switches. Also, contention can develop at switching elements, which might degrade performance. IBM's SP series multiprocessor, which evolved to the POWER4, uses a multistage network to connect its processors. The IBM ASCI-White, which can perform over 100 trillion operations per second, is based on the POWER3-II multiprocessor.73, 74 Self Review 1. An 8-connected 2-D mesh network includes links to diagonal nodes as well as the links in Fig. 15.3. Compare 4-connected and 8-connected 2-D mesh networks using the criteria discussed in this section. 2. Compare a multistage network to a crossbar-switch matrix. What are the benefits and drawbacks or each interconnection scheme? Ans: 1) An 8-connected 2-D mesh network has approximately double the degree for each node and, therefore, approximately double the cost compared to a 4-connected 2-D mesh network. However, the network diameter decreases and the bisection width correspondingly increases, because there are more links in the network. Thus, an 8-connected 2-D mesh network exhibits lower network latency and higher fault tolerance at a higher cost than a 4-connected 2-D mesh network. 2) A multistage network is cheaper to build than a crossbar-switch matrix. However, multistage networks exhibit reduced performance and fault tolerance. 15.2.3 Loosely Coupled vs. Tightly Coupled Systems Another defining characteristic of multiprocessors is how the processors share system resources. In a tightly coupled system (Fig. 15.6), the processors share most system resources. Tightly coupled systems often employ shared buses, and processors usually communicate via shared memory. A centralized operating system typically manages the system's components. Loosely coupled systems (Fig. 15.7) normally connect components indirectly through communication links.75 Processors sometimes share memory, but often each processor maintains its own local memory, to which it has much faster access than to the rest of memory. In other cases, message passing is the only form of communication between processors, and memory is not shared. In general, loosely coupled systems are more flexible and scalable than tightly coupled systems. When components are loosely connected, designers can easily add 15.2 Multiprocessor Architecture 697 components to or remove components from the system. Loose coupling also increases fault tolerance because components can operate independently of one another. However, loosely coupled systems typically are less efficient, because they communicate by passing messages over a communications link, which is slower than communicating through shared memory. This also places a burden on operating systems programmers, who typically hide most of the complexity of message passing from application programmers. Japan's Earth Simulator is an example of a loosely coupled system.76 By contrast, tightly coupled systems perform better but are less flexible. Such systems do not scale well, because contention for shared resources builds up quickly as processors are added. For this reason, most systems with a large number of processors are loosely coupled. In a tightly coupled system, designers can optimize interactions between components to increase system performance. However, this decreases the system's flexibility and fault tolerance, because one component relies on other components. A dual-processor Intel Pentium system is an example of a tightly coupled system.77 Figure 15.6 | Tightly coupled system Figure 15.7 | Loosely coupled system. 698 Multiprocessor Management Self Review 1. Why are many small multiprocessors constructed as tightly coupled systems? Why are many large-scale systems loosely coupled? 2. Some systems consist of several tightly coupled groups of components connected together in a loosely coupled fashion. Discuss some motivations for this scheme. Ans: 1) A small system would typically be tightly coupled because there is little contention for shared resources. Therefore, designers can optimize the interactions between system components, thus providing high performance. Typically, large-scale systems are loosely coupled to eliminate contention for shared resources and reduce the likelihood that one failed component will cause systemwide failure. 2) This scheme leverages the benefits of both interconnection organizations. Each group is small, so tightly coupled systems provide the best performance within a group. Making the overall system loosely coupled reduces contention and increases the system's flexibility and fault tolerance. 15.3 Multiprocessor Operating System Organizations In organization and structure, multiprocessor operating systems differ significantly from uniprocessor operating systems. In this section, we categorize multiprocessors based on how they share operating system responsibilities. The basic operating system organizations for multiprocessors are master/slave, separate kernels for each processor and symmetric (or anonymous) treatment of all processors. 15.3.1 Master/Slave The master/slave multiprocessor organization designates one processor as the master and the others as slaves (Fig. 15.8).78 The master executes operating system code; the slaves execute only user programs. The master performs input/output and computation. The slaves can execute processor-bound jobs effectively, but I/Obound jobs running on the slaves cause frequent calls for services that only the master can perform. From a fault-tolerance standpoint, some computing capability is lost when a slave fails, but the system continues to function. Failure of the master processor is catastrophic and halts the system. Master/slave systems are tightly coupled because all slave processors depend on the master. The primary problem with master/slave multiprocessing is the hardware asymmetry, because only the master processor execute the operating system. When Figure 15.8 | Master/Slave multiprocessing. 15.3 Multiprocessor Operating System Organizations 699 a process executing on a slave processor requires the operating system's attention, the slave processor generates an interrupt and waits for the master to handle the interrupt. The nCUBE hypercube system is an example of a master/slave system. Its many slave processors handle computationally intensive tasks associated with manipulating graphics and sound.79 Self Review 1. (T/F) Master/slave multiprocessors scale well to large-scale systems. 2. For what type of environments are master/slave multiprocessors best suited? Ans 1) False. Only one processor can execute the operating system. Contention will build between processes that are awaiting services (such as I/O) that only the master processor can provide. 2) Master/slave processors are best suited for environments that mostly execute processor-bound processes. The slave processors can execute these processes without intervention from the master processor. 15.3.2 Separate Kernels In the separate-kernels multiprocessor organization, each processor executes its own operating system and responds to interrupts from user-mode processes running on that processor.80 A process assigned to a particular processor runs to completion on that processor. Several operating system data structures contain global system information, such as the list of processes known to the system. Access to these data structures must be controlled with mutual exclusion techniques, which we discuss in Section 15.9, Multiprocessor Mutual Exclusion. Systems that use the separate-kernels organization are loosely coupled. This organization is more fault tolerant than the master/slave organization—if a single processor fails, system failure is unlikely. However, the processes that were executing on the failed processor cannot execute until they are restarted on another processor. In the separate-kernels organization, each processor controls its own dedicated resources, such as files and I/O devices. I/O interrupts return directly to the processors that initiate those interrupts. This organization benefits from minimal contention over operating system resources, because the resources are distributed among the individual operating systems for their own use. However, the processors do not cooperate to execute an individual process, so some processors might remain idle while one processor executes a multithreaded process. The Tandem system, which is popular in businesscritical environments, is an example of a separate-kernels multiprocessor; because the operating system is distributed throughout the loosely coupled processing nodes, it is able to achieve nearly 100 percent availability.81 700 Multiprocessor Management Self Review 1. Why is the separate-kernels organization more fault tolerant than the master/slave organization? 2. For what type of environment would separate-kernels multiprocessor be useful? Ans: 1) The separate-kernels organization is loosely coupled. Each processor has its own resources and does not interact with other processors to complete its tasks. If one processor fails, the remainder continue to function. In master/slave organizations, if the master fails none of the slave processors can perform operating system tasks that must be handled by the master. 2) A separate-kernels multiprocessor is useful in environments where processes do not interact, such as a cluster of workstations in which users execute independent programs. 15.3.3 Symmetrical Organization Symmetrical multiprocessor organization is the most complex organization to implement, but is also the most powerful.82-83 [Note: This organization should not be confused with symmetric multiprocessor (SMP) systems, which we discuss in Section 15.4.1.] The operating system manages a pool of identical processors, any one of which can control any I/O device or reference any storage unit. The symmetry makes it possible to balance the workload more precisely than with the other organizations. Because many processors might be executing the operating system at once, mutual exclusion must be enforced whenever the operating system modifies shared data structures. Hardware and software conflict-resolution techniques are important. Symmetrical multiprocessor organizations are generally the most fault tolerant. When a processor fails, the operating system removes that processor from its pool of available processors. The system degrades gracefully while repairs are made (see the Operating Systems Thinking feature, Graceful Degradation). Also, a process running on a symmetrical system organization can be dispatched to any processor. Therefore, a process does not rely on a specific processor as in the separatekernels organization. The operating system "floats" from one processor to the next. One drawback of the symmetrical multiprocessor organization is contention for operating system resources, such as shared data structures. Careful design of system data structures is essential to prevent excessive locking that prevents the operating system from executing on multiple processors at once. A technique that minimizes contention is to divide the system data structures into separate and independent entities that can be locked individually. Even in completely symmetrical multiprocessing systems, adding new processors does not cause system throughput to increase by the new processors' rated capacities. There are many reasons for this, including additional operating system overhead, increased contention for system resources and hardware delays in switching and routing transmissions between an increased number of components. Several performance studies have been performed; one early BBN Butterfly system with 256 processors operated 180 to 230 times faster than a single-processor system.84 15.4 Memory Access Architectures 701 Self Review 1. Why would doubling the number of processors in a symmetrical organization not double the total processing power? 2. What are some benefits of the symmetrical organization over the master/slave and separate-kernels organizations? Ans: 1) Adding processors increases contention for resources and increases operating system overhead, offsetting some of the performance gains attained from adding processors. 2) The symmetrical organization is more scalable than master/slave because all processors can execute the operating system; this also makes the system more fault tolerant. Symmetrical organization provides better cooperation between processors than separate kernels. This facilitates IPC and enables systems to exploit parallelism better. 15.4 Memory Access Architectures So far, we have classified multiprocessor systems by hardware characteristics and by how processors share operating system responsibilities. We can also classify multiprocessor systems by how they share memory. For example, consider a system with tew processors and a small amount of memory. If the system contains a group of memory modules that is easily accessible by all processors (e.g., via a shared bus), the system can maintain fast memory access. However, systems with many processors and memory modules will saturate the bus that provides access to these memory modules. In this case, a portion of memory can be closely tied to a processor so Operating Systems Thinking Graceful Degradation Things do go wrong. Individual components of systems do fail. Entire systems fail. The question is, "What should a system do after experiencing some degree of failure?" Many systems are designed to degrade gracefully (i.e., continue operating after a failure, but at reduced levels of service). A classic example of graceful degradation occurs in a symmetric multiprocessing system in which any processor can run any process. If one of the processors fails, the system can still function with the remaining processors, but with reduced performance. Graceful degradation occurs in many disk systems that, upon detecting a failed portion of the disk, simply "map around" the failed areas, enabling the user to continue storing and retrieving information on that disk. When a router fails on the Internet, the Internet continues to handle new transmissions by sending them to the remaining functioning routers. Providing for graceful degradation is an important task for operating systems designers, especially as people grow increasingly reliant on hardware devices that eventually fail. 702 Multiprocessor Management that the processor can access its memory more efficiently. Designers, therefore, must weigh concerns about a system's performance, cost and scalability when determining the system's memory-access architecture. The following sections describe several common memory architectures for multiprocessor systems. 15.4.1 Uniform Memory Access Uniform-memory-access (LIMA) multiprocessor architectures require all processors to share the system's main memory (Fig. 15.9). This is a straightforward extension of a uniprocessor memory architecture, but with multiple processors and memory modules. Typically, each processor maintains its own cache to reduce bus contention and increase performance. Memory-access time is uniform for any processor accessing any data item, except when the data item is stored in a processor's cache or there is contention on the bus. UMA systems are also called symmetric multiprocessor (SMP) systems because any processor can be assigned any task and all processors share all resources (including memory, I/O devices and processes). UMA multiprocessors with a small number of processors typically use a shared bus or a crossbarswitch matrix interconnection network. I/O devices are attached directly to the interconnection network and are equally accessible to all processors.85 UMA architectures are normally found in small multiprocessor systems (typically two to eight processors). UMA multiprocessors do not scale well—a bus quickly becomes saturated when more than a few processors access main memory simultaneously, and crossbar-switch matrices become expensive even for modestsized systems.86-87 Self Review 1. Why are mesh networks and hypercubes inappropriate interconnection schemes for UMA systems? 2. How is a UMA system "symmetric?" Ans: 1) Mesh networks and hypercubes place processors and memory at each node, so local memory can be accessed faster than remote memory; thus memory-access times are not uni- Figure 1 5 . 9 | UMA multiprocessor. 15.4 Memory Access Architectures 703 form. 2) It is symmetric because any processor can be assigned any task and all processors share all resources (including memory, I/O devices and processes). 15.4.2 Nonuniform Memory Access Nonuniform-memory-access (NUMA) multiprocessor architectures address UMA's scalability problems. The primary bottleneck in a large-scale UMA system is access to shared memory—performance degrades due to contention from numerous processors attempting to access shared memory. If a crossbar-switch matrix is used, the interconnection scheme's cost might increase substantially to facilitate multiple paths to shared memory. NUMA multiprocessors handle these problems by relaxing the constraint that memory-access time should be uniform for all processors accessing any data item. NUMA multiprocessors maintain a global shared memory that is accessible by all processors. The global memory is partitioned into modules, and each node uses one of these memory modules as the processor's local memory. In Fig. 15.10, each node contains one processor, but this is not a requirement. Although the interconnection scheme's implementation might vary, processors are connected directly to their local memory modules and connected indirectly (i.e., through one of the interconnection schemes discussed in Section 15.2.2) to the rest of global memory. This arrangement provides faster access to local memory than to the rest of global memory, because access to global memory requires traversing the interconnection network. The NUMA architecture is highly scalable because it reduces bus collisions when a processor's local memory services most of the processor's memory requests. A NUMA system can implement a strategy that moves pages to the processor on which those pages are accessed most frequently —a technique called page migration, which is discussed in detail in Section 15.5.2, Page Replication and Migration. Typically, NUMA systems can support a large number of processors, but they are more complex to design than UMAs, and systems with many processors can be expensive to implement.88-89 Figure 15.10 | NUMA multiprocessor. 704 Multiprocessor Management Self Review 1. List some advantages of NUMA multiprocessors over UMA. List some disadvantages. 2. What issues does the NUMA design raise for programmers and for operating systems designers? Ans: 1) NUMA multiprocessors are more scalable than UMA multiprocessors because NUMA multiprocessors remove the memory-access bottleneck of UMA systems by using large numbers of processors. UMA multiprocessors are more efficient for small systems because there is little memory contention and access to all memory in a UMA system is fast. 2) If two processes executing on processors at separate nodes use shared memory for IPC, then at least one of the two will not have the memory item in its local memory, which degrades performance. Operating system designers place processes and their associated memory together on the same node. This might require scheduling a process on the same processor each time the process executes. Also, the operating system should be able to move pages to different memory modules based on demand. 15.4.3 Cache-Only Memory Architecture As described in the previous section, each node in a NUMA system maintains its own local memory, which processors from other nodes can access. Often, local memory access is dramatically faster than global memory access (i.e., access to another node's local memory). Cache-miss latency—the time required to retrieve data that is not in the cache —can be significant when the requested data is not present in local memory. One way to reduce cache-miss latency is to reduce the number of memory requests serviced by remote nodes. Recall that NUMA systems place data in the local memory of the processor that accesses the data most frequently. This is impractical for a compiler or programmer to implement, because data access patterns change dynamically. Operating systems can perform this task, but can move only page-size chunks of data, which can slow data migration. Also, different data items on a single page often are accessed by processors at different nodes.90 Cache-Only Memory Architecture (COMA) multiprocessors (also called Cache-Only Memory Access multiprocessors) use a slight variation of NUMA to address this memory placement issue (Fig. 15.11). COMA multiprocessors include one or more processors, each with its associated cache, and a portion of the global shared memory. However, the memory associated with each node is organized as a large cache known as an attraction memory (AM). This allows the hardware to migrate data efficiently at the granularity of a memory line—equivalent to a cache line, but in main memory, and typically four or eight bytes.91 Also, because each processor's local memory is viewed as a cache, different AMs can have copies of the same memory line. With these design modifications, data often resides in the local memory of the processor that uses the data most frequently, which reduces the average cache-miss latency. The trade-offs are memory overhead from duplicating data items in multiple memory modules, and complicated hardware and protocols to ensure that memory updates are reflected in each processor's AM. This overhead results in higher latency for those cache misses that are serviced remotely.92 15.4 Memory Access Architectures 705 Figure 15.11 | COMA multiprocessor. Self Review 1. What problems inherent in NUMA multiprocessors does the COMA design address? 2. (T/F) The COMA design always increases performance over a NUMA design. Ans: 1) Cache misses serviced at remote nodes in both NUMA and COMA systems typically require much more time than cache misses serviced by local memory. Unlike NUMA systems, COMA multiprocessors use hardware to move copies of data items to a processor's local memory (the AM) when referenced. 2) False. COMA multiprocessors reduce the num ber of cache misses serviced remotely, but they also add overhead. In particular, cache-miss latency increases for cache misses that are serviced remotely, and synchronization is required for data that is copied into several processors' attraction memory. 15.4.4 No Remote Memory Access UMA, NUMA and COMA multiprocessors are tightly coupled. Although NUMA (and COMA to a lesser extent) multiprocessors scale well, they require complex software and hardware. The software controls access to shared resources such as memory; the hardware implements the interconnection scheme. NO-RemoteMemory-Access (NORMA) multiprocessors are loosely coupled multiprocessors that do not provide any shared global memory (Fig. 15.12). Each node maintains its own local memory, and often, NORMA multiprocessors implement a common shared virtual memory (SVM). On an SVM system, when a process requests a page that is not in its processor's local memory, the operating system loads the page into local memory from another memory module (i.e., from a remote computer over a network) or from secondary storage (e.g., a disk). 93 - 94 Nodes in NORMA systems that do not support SVM must share data through explicit message passing. Google, which powers its service using 15,000 low-end servers located across the world, is an example of a distributed NORMA multiprocessor system.95 706 Multiprocessor Management Figure 15.12 | NORMA multiprocessor. NORMA systems are the simplest to build, because they do not require a complex interconnection scheme. However, the absence of shared global memory requires application programmers to implement IPC via message passing and remote procedure calls. On many systems, using shared virtual memory is inefficient, because the system would have to send entire data pages from one processor to the next, and these processors often are not in the same physical machine.96 We discuss SVM in more detail in Section 15.5.3, Shared Virtual Memory. Because NORMA multiprocessors are loosely coupled, it is relatively easy to remove or add nodes. NORMA multiprocessors are distributed systems governed by a single operating system, rather than networked computers each with its own operating system. If a node in a NORMA system fails, a user can simply switch to another node and continue working. If one percent of the nodes fails in a NORMA multiprocessor, the system merely runs one percent slower.97 Self Review 1. Why are NORMA multiprocessors ideal for workstation clusters? 2. In what environments are NORMA multiprocessors not useful? Ans: 1) Different users in a workstation cluster usually do not share memory. However, they share resources such as a file system and processing power, which NORMA systems provide. Also, it is relatively easy to add or remove nodes in NORMA multiprocessors, which can be useful for scaling the system's processing capabilities. 2) NORMA multiprocessors are not useful in shared-memory environments, especially systems with a single user or a few processors. These can be implemented more efficiently with UMA or NUMA designs. 15.5 Multiprocessor Memory Sharing 15.5 Multiprocessor Memory Sharing When multiple processors with private caches or local memories access shared memory, designers must address the problem of memory coherence. Memory is coherent if the value obtained from reading a memory address is always the value that was most recently written to that address.98 For example, consider two processors that maintain a separate copy of the same page in their local memories. If one processor modifies its local copy, the system should ensure that the other copy of the page is updated to reflect the latest changes. Multiprocessors that allow only one copy of a page to be resident in the system at a time (such as UMA and small NUMA systems) must still ensure cache coherence —reading a cache entry reflects the most recent update to that data. Large-scale systems (such as NORMA or large-scale NUMA systems) often permit the same memory page to be resident in the local memory of several processors. These systems must integrate memory coherence into their implementation of shared virtual memory. Other important design considerations are page placement and replication. Local memory access is much faster than global memory access, so ensuring that data accessed by a processor resides in that processor's local memory can improve performance. There are two common strategies for addressing this issue. Page replication maintains multiple copies of a memory page, so that the page can be accessed quickly at multiple nodes (see the Operating Systems Thinking feature, Data Replication and Coherency). Page migration transfers pages to the node (or nodes, when used with page replication) where processors access a page most. In Section 15.5.2, Page Replication and Migration, we consider implementations of these two strategies. Self Review 1. Why is memory coherence important? 2. What are the advantages of allowing multiple copies of the same page to exist in one system? Operating Systems Thinking Data Replication and Coherency We will see several examples of data being duplicated for various reasons in operating systems. Sometimes this is done for backup to ensure recoverability if one copy of the data is lost. It is also done for performance where some copies of the data may be accessed faster than others. Often, it is crucial to ensure coher- ency, (i.e., that all copies of the data either are identical or will be made so before any differences could cause a problem). 707 708 Multiprocessor Management Ans: 1) Without memory coherence, there is no guarantee that a particular copy of data is the most up-to-date version. Data must be coherent to ensure that applications produce correct results. 2) If two processes executing on different processors repeatedly read the same memory page, replicating the page allows both processors to keep the page in local memory so that it can be accessed quickly. 15.5.1 Cache Coherence Memory coherence became a design consideration when caches appeared, because computer architectures allowed different access paths to data (i.e., through the cache copy or the main memory copy). In multiprocessor systems, coherence is complicated by the fact that each processor maintains a private cache. UMA Cache Coherence Implementing cache coherence protocols for UMA multiprocessors is simple because caches are relatively small and the bus connecting shared memory is relatively fast. When a processor updates a data item, the system must also update or discard all instances of that data in other processors' caches and in main memory.99 This can be accomplished by bus snooping (also called cache snooping). In this protocol, a processor "snoops" the bus by determining whether a requested write from another processor is for a data item in the processor's cache. If the data resides in the processor's cache, the processor removes the data item from its cache. Bus snooping is simple to implement, but generates additional traffic on the shared bus. Alternatively, the system can maintain a centralized directory that records the items that reside in each cache and indicates when to remove stale data (i.e., data that does not reflect the most recent update) from a cache. Another option is for the system to allow only one processor to cache a particular memory item. 100 Cache-Coherent NUMA (CC-NUMA) Cache-coherent NUMAs (CC-NUMAs) are NUMA multiprocessors that impose cache coherence. In a typical CC-NUMA architecture, each physical memory address is associated with a home node, which is responsible for storing the data item with that main-memory address. (Often, the home node is simply determined by the higher-order bits in the address.) When a cache miss occurs at a node, that node contacts the home node associated with the requested memory address. If the data item is clean (i.e., no other node has a modified version of the data item in its cache), the home node forwards it to the requesting processor's cache. If the data item is dirty (i.e., another node has written to the data item since the last time the main memory entry was updated), the home node forwards the request to the node with the dirty copy; this node sends the data item to both the requestor and the home node. Similarly, requests to modify data are performed via the home node. The node that wishes to modify data at a particular memory address requests exclusive ownership of the data. The most recent version of the data (if not already in the modifying node's cache) is obtained in the same manner as a read request. After the 15.5 Multiprocessor Memory Sharing 709 modification, the home node notifies other nodes with copies of the data that the data was modified.101 This protocol is relatively simple to implement, because all reads and writes first contact the home node. Although this might seem inefficient, this coherence protocol requires a maximum of only three network communications. (Consider how much traffic would be generated if a writing node had to contact all other nodes.) This protocol also makes it easy to distribute the load throughout the system—by assigning each node as the home node for approximately the same number of addresses—which increases fault tolerance and reduces contention. However, this protocol can perform poorly if most data accesses come from remote nodes. Self Review 1. What are the advantages and disadvantages of bus snooping? 2. Why is it difficult for the system to establish a home node for memory that is referenced by several different nodes over time? How can the CC-NUMA home-based protocol be modified to support such behavior? Ans: 1) Bus snooping is easy to implement and it enables cache coherence. However, it generates additional bus traffic. 2) If memory-access patterns for a memory region change frequently, then it is difficult to decide which node should be the home node for that region. Although one node might make the majority of the references to that item now, this will soon change and most of the references will come from a remote node. A remedy would be to dynamically change the home node of a data item (this, of course, would imply changing the item's physical memory address). 15.5.2 Page Replication and Migration NUMA systems have higher memory-access latency than UMA multiprocessors, which limits NUMA's performance. Therefore, maximizing the number of cache misses serviced by local memory is an important NUMA design consideration. 102 The COMA design, described in Section 15.4.3, is one attempt to address the NUMA latency issue. CC-NUMA systems address the latency issue by implementing page-migration or page-replication strategies.103 Replicating a page is straightforward. The system copies all data from a remote page to a page in the requesting processor's local memory. To maintain memory coherence, the system uses a data structure that records where all duplicate pages exist. We discuss different strategies for memory coherence in the next section. Page migration occurs in a similar fashion, except that after a page has been replicated, the original node deletes the page from its memory and flushes any associated TLB or cache entries.104 Although migrating and replicating pages has obvious benefits, these strategies can degrade performance if they are not implemented correctly. For example, remotely referencing a page is faster than migrating or replicating that page. Therefore, if a process references a remote page only once, it would be more efficient not to migrate or replicate the page.105 In addition, some pages are better candidates for replication than for migration, and vice versa. For example, pages that are read fre- 710 Multiprocessor Management quently by processes at different processors would benefit from replication. by enabling all processes to access that page from local memory. Also, the drawback of replication—maintaining coherency on writes—would not be a problem for a readonly page. A page frequently modified by one process is a good candidate for migration; if other processes are reading from that page, replicating the page would not be viable because these processes would have to fetch updates continually from the writing node. Pages that are frequently written by processes at more than one processor are not good candidates for either replication or migration, because they would be migrated or replicated after each write operation. 106 To help determine the best strategy, many systems maintain each page's access history information. A system might use this information to determine which pages are frequently accessed —these should be considered for replication or migration The overhead of replicating or migrating a page is not justified for pages that are not frequently accessed. A system also might maintain information on which remote processors are accessing the page and whether these processors are reading from, or writing to, the page. The more page-access information the system gathers, the better decision the system can make. However, gathering this history and transferring it during migration incurs overhead. Thus, the system must collect enough information to make good replication or migration decisions without incurring excessive overhead from maintaining this information.107 Self Review 1. How could an ineffective migration/replication strategy degrade performance? 2. For what types of pages is replication appropriate? When is migration appropriate? Ans: 1) Migration and replication are more costly than remotely referencing a page. They are beneficial only if the page will be referenced remotely multiple times from the same node. An ineffective strategy might migrate or replicate pages that are, for example, modified by many nodes, requiring frequent updates and thus reducing performance. Also, manipulating pages access history information imposes additional overhead that can degrade performance. 2) Pages that are read frequently by multiple processors (but not modified) should be replicated. Pages that are written by only one remote node should be migrated to that node. 15.5.3 Shared Virtual Memory Sharing memory on small, tightly coupled multiprocessors, such as UMA multiprocessors, is a straightforward extension of uniprocessor shared memory because all processors access the same physical addresses with equal access latency. This strategy is impractical for large-scale NUMA multiprocessors due to remote-memory-access latency, and it is impossible for NORMA multiprocessors, which do not share physical memory. Because IPC through shared memory is easier to program than IPC via message passing, many systems enable processes to share memory located on different nodes (and perhaps in different physical address spaces) through shared virtual memory (SVM). SVM extends uniprocessor virtual memory concepts by ensuring memory coherence for pages accessed by various processors.108 15.5 Multiprocessor Memory Sharing 711 Two issues facing SVM designers are selecting which coherence protocol to use and when to apply it. The two primary coherence protocols are invalidation and write broadcast. First we describe these protocols, then we consider how to implement them. Invalidation In the invalidation memory-coherence approach, only one processor may access a page while it is being modified —this is called page ownership. To obtain page ownership, a processor must first invalidate (deny access to) all other copies of the page. After the processor obtains ownership, the page's access mode is changed to read/ write, and the processor copies the page into its local memory before accessing it. To obtain read access to a page, a processor must request that the processor with read/write access change its access mode to read-only access. If the request is granted, other processors can copy the page into local memory and read it. Typically, systems employ the policy of always granting requests unless a processor is waiting to acquire page ownership; in the case that a request is denied, the requestor must wait until the page is available. Note that multiple processes modifying a single page concurrently leads to poor performance, because competing processors repeatedly invalidate the other processors' copies of the page.109 Write Broadcast In the write broadcast memory-coherence approach, the writing processor broadcasts each modification to the entire system. In one version of this technique, only one processor may obtain write ownership of a page. Rather than invalidating all other existing copies of the page, the processor updates all copies.110 In a second version, several processors are granted write access to increase efficiency. Because this scheme does not require writers to obtain write ownership, processors must ensure that various updates to a page are applied in the correct order, which can incur significant overhead. 111 Implementing Coherence Protocols There are several ways to implement a memory-coherence protocol, though designers must attempt to limit memory-access bus traffic. Ensuring that every write is reflected at all nodes as soon as possible can decrease performance; however, weakening coherence constraints can produce erroneous results if programs use stale data. In general, an implementation must balance performance and data integrity. Systems that use sequential consistency ensure that all writes are immediately reflected in all copies of a page. This scheme does not scale well to large systems due to communication cost and page size. Internode communication is slow compared to accessing local memory; if a process repeatedly updates the same page, many wasteful communications will result. Also, because the operating system manipulates pages, which often contain many data items, this strategy might result in false sharing, where processes executing at separate nodes might need access to unrelated data on the same page. In this case, sequential consistency effectively requires 712 Multiprocessor Management the two processes to share the page, even though their modifications do not affect one another. 112 Systems that use relaxed consistency perform coherence operations periodically. The premise behind this strategy is that delaying coherence for remote data by a few seconds is not noticeable to the user and can increase performance and reduce the false sharing effect.113 In release consistency, a series of accesses begin with an acquire operation and end with a release operation. All updates between the acquire and the release are batched as one update message after the release. 114 In lazy release consistency, the update is delayed until the next acquire attempt for the modified memory. This reduces network traffic, because it eliminates coherence operations for pages that are not accessed again.115 One interesting consequence of lazy release consistency is that only a node that attempts to access a modified page receives the modification. If this node also modifies the page, a third node would need to apply two updates before it can access the page. This can lead to substantial page-update overhead. One way to reduce this problem is to provide periodic global synchronization points at which all data must be consistent.116 An alternative way to reduce this problem is to use a home-based consistency approach similar to that used for cache coherence in a CC-NUMA. The home node for each page is responsible for keeping an updated version of the page. Nodes that issue an acquire to a page that has been updated since that its last release must obtain the update from the home node. 117 The delayed consistency approach sends update information when a release occurs, but nodes receiving updates do not apply them until the next acquisition. These updates can be collected until a new acquire attempt, at which point the node applies all update information. This does not reduce network traffic, but it improves performance because nodes do not have to perform coherence operations as frequently 118 Lazy data propagation notifies other nodes that a page has been modified at release time. This notification does not supply the modified data, which can reduce bus traffic. Before a node acquires data for modification, it determines if that data has been modified by another node, If so, the former node retrieves update information from the latter. Lazy data propagation has the same benefits and drawbacks as lazy release consistency. This strategy significantly reduces communication traffic, but requires a mechanism to control the size of update histories, such as a global synchronization point.119 Self Review 1. What is a benefit of relaxed consistency? A drawback? 2. In many relaxed consistency implementations, data may not be coherent for several seconds. Is this is a problem in all environments? Give an example where it might be a problem. 15.6 Multiprocessor Scheduling 713 Ans: 1) Relaxed consistency reduces network traffic and increases performance. However, the system's memory can be incoherent for a significant period of time, which increases the likelihood that processors will operate on stale data. 2) This is not a problem in all environments, considering that even sequential consistency schemes may use networks exhibiting latencies of a second or two. However, latency of this magnitude is not desirable for a supercomputer in which many processes interact and communicate through shared memory. The extra latency degrades performance due to frequent invalidations. 15.6 Multiprocessor Scheduling Multiprocessor scheduling has goals similar to those of uniprocessor scheduling— the system attempts to maximize throughput and minimize response time for all processes. Further, the system must enforce scheduling priority Unlike uniprocessor scheduling algorithms which only determine in what order processes are dispatched, multiprocessor scheduling algorithms also must determine to which processor they are dispatched, which increases scheduling complexity. For example, multiprocessor scheduling algorithms should ensure that processors are not idle while processes are waiting to execute. When determining the processor on which a process is executed, the scheduler considers several factors. For example, some strategies focus on maximizing parallelism in a system to exploit application concurrency. Systems often group collaborating processes into a job. Executing a job's processes in parallel improves performance by enabling these processes to execute truly simultaneously. This section introduces several timesharing scheduling algorithms that attempt to exploit such parallelism by scheduling collaborative processes on different processors.120 This enables the processes to synchronize their concurrent execution more effectively. Other strategies focus on processor affinity—the relationship of a process to a particular processor and its local memory and cache. A process that exhibits high processor affinity executes on the same processor through most or all of its life cycle. The advantage is that the process will experience more cache hits and, in the case of a NUMA or NORMA design, possibly fewer remote page accesses than if it were to execute on several processors throughout its life cycle. Scheduling algorithms that attempt to schedule a process on the same processor throughout its life cycle maintain soft affinity, whereas algorithms that schedule a process on only one processor maintain hard affinity.121 Space-partitioning scheduling algorithms attempt to maximize processor affinity by scheduling collaborative processes on a single processor (or single set of processors) under the assumption that collaborative processes will access the same shared data, which is likely stored in the processor's caches and local memory.122 Therefore, space-partitioning scheduling increases cache and local memory hits. However, it can limit throughput, because these processes typically do not execute simultaneously.123 Multiprocessor scheduling algorithms are generally classified as job blind or job aware. Job-blind scheduling policies incur minimal scheduling overhead, 714 Multiprocessor Management because they do not attempt to enhance a job's parallelism or processor affinity. Job-aware scheduling evaluates each job's properties and attempts to maximize each job's parallelism or processor affinity, which increases performance at the cost of increased overhead. Many multiprocessor scheduling algorithms organize processes in global run queues.124 Each global run queue contains all the processes in the system that are ready to execute. Global run queues might be used to organize processes by priority, by job or by which process executed most recently.125 Alternatively, systems might use a per-processor run queue. This is typical of large, loosely coupled systems (such as NORMA systems) in which cache hits and references to local memory should be maximized. In this case, processes are associated with a specific processor and the system implements a scheduling policy for that processor. Some systems use per-node run queues; each node might contain more than one processor. This is appropriate for a system in which a process is tied to a particular group of processors. We describe the related issue of process migration, which entails moving processes from one per-processor or per-node run queue to another, in Section 15.7, Process Migration. Self Review 1. What types of processes benefit from timesharing scheduling? From space-partitioning scheduling? 2. When are per-node run queues more appropriate than global run queues? Ans: 1) Timesharing scheduling executes related processes simultaneously, improving performance for processes that interact frequently because processes can react to messages or modifications to shared memory immediately. Space-partitioning scheduling is more appropriate for processes that must sequentialize access to shared memory (and other resources) because their processors are likely to have cached shared data, which improves performance. 2) Per-node run queues are more appropriate than global run queues in loosely coupled systems, where processes execute much less efficiently when accessing remote memory. 15.6.1 Job-Blind Multiprocessor Scheduling Job-blind multiprocessor scheduling algorithms schedule jobs or processes on any available processor. The three algorithms described in this section are examples of job-blind multiprocessor scheduling algorithms. In general, any uniprocessor scheduling algorithm, such as those described in Chapter 8, can be implemented as a jobblind scheduling multiprocessor algorithm. First-Come-First-Served (FCFS) Process Scheduling First-come-first-served (FCFS) process scheduling places arriving processes in a global run queue. When a processor becomes available, the scheduler dispatches the process at the head of the queue and runs it until the process relinquishes the processor. FCFS treats all processes fairly by scheduling them according to their arrival times. However, FCFS might be considered unfair because long processes make 15.6 short processes wait, and low-priority processes can make high-priority processes wait—although a preemptive version of FCFS can prevent the latter from occurring. Typically, FCFS scheduling is not useful for interactive processes, because FCFS cannot guarantee short response times. However, FCFS is easy to implement and eliminates the possibility of indefinite postponement—once a process enters the queue, no other process will enter the queue ahead of it.126-127 Round-Robin Process (RRprocesss) Multiprocessor Scheduling Round-robin process (RRprocess) scheduling places each ready process in a global run queue. RRprocess scheduling is similar to uniprocessor round-robin scheduling—a process executes for at most one quantum before the scheduler dispatches a new process for execution. The previously executing process is placed at the end of the global run queue. This algorithm prevents indefinite postponement, but does not facilitate a high degree of parallelism or processor affinity because it ignores relationships among processes.128-129 Shortest-Process-First (SPF) Multiprocessor Scheduling A system can also implement the shortest-process-first (SPF) scheduling algorithm, which dispatches the process that requires the least amount of time to run to completion.130 Both preemptive and nonpreemptive versions of SPF exhibit lower average waiting times for interactive processes than FCFS does, because interactive processes typically are "short" processes. However, a longer process can be indefinitely postponed if shorter processes continually arrive before it can obtain a processor. As with all job-blind algorithms, SPF does not consider parallelism or processor affinity. Self Review 1. Is a UMA or a NUMA system more suitable for job-blind multiprocessor scheduling? 2. Which job-blind scheduling strategy discussed in this section is most appropriate for batch processing systems and why? Ans: 1) A UMA system is more appropriate for a job-blind algorithm. NUMA systems often benefit from scheduling algorithms that consider processor affinity, because memory access time depends on the node at which a process executes, and nodes in a NUMA system have their own local memory. 2) SPF is most appropriate because it exhibits high throughput, an important goal for batch-processing systems. 15.6.2 Job-Aware Multiprocessor Scheduling Although job-blind algorithms are easy to implement and incur minimal overhead, they do not consider performance issues specific to multiprocessor scheduling. For example, if two processes that communicate frequently do not execute simultaneously, they might spend a significant amount of their time busy waiting, which degrades overall system performance. Furthermore, in most multiprocessor systems, Multiprocessor Scheduling 715 716 Multiprocessor Management each processor maintains its own private cache. Processes in the same job often access the same memory items, so scheduling one job's processes on the same processor tends to increase cache hits and improves memory-access performance. In general, job-aware process-scheduling algorithms attempt to maximize parallelism or processor affinity, at the cost of greater scheduling algorithm complexity. Smallest-Number-of-Processes-First (SNPF) Scheduling The smallest-number-of-processes-first (SNPF) scheduling algorithm, which can be either preemptive or nonpreemptive, uses a global job priority queue. A job's priority is inversely proportional to the number of processes in the job. If jobs containing the same number of processes compete for a processor, the job that has waited the longest receives priority. In nonpreemptive SNPF scheduling, when a processor becomes available, the scheduler selects a process from the job at the head of the queue and allows it to execute to completion. In preemptive SNPF scheduling, if a new job arrives with fewer processes, it receives priority and its processes are dispatched immediately 131 - 132 SNPF algorithms improve parallelism, because processes that are associated with the same job can often execute concurrently. However, the SNPF algorithms do not attempt to improve processor affinity. Further, it is possible for jobs with many processes to be postponed indefinitely. Round-Robin Job (RRJob) Scheduling Round-robin job (RRJob) scheduling employs a global job queue from which each job is assigned to a group of processors (although not necessarily the same group each time the job is scheduled). Every job maintains its own process queue. If the system contains p processors and uses a quantum of length q, then a job receives a total of p x q of processor time when dispatched. Typically, a job does not contain exactly p processes that each exhaust one quantum (e.g., a process may block before its quantum expires). Therefore, RRJob uses round-robin scheduling to dispatch the job's processes until the job consumes the entire p x q quanta, the job completes or all of the job's processes block. The algorithm also can divide the p x q quanta equally among the processes in the job, allowing each to execute until it exhausts its quantum, completes or blocks. Alternatively, if a job has more than p processes, it can select p processes to execute for a quantum length of q.133 Similar to RRprocess scheduling, this algorithm prevents indefinite postponement. Further, because processes from the same job execute concurrently, this algorithm promotes parallelism. However, the additional context-switching overhead of round-robin scheduling can reduce job throughput.134-135 Cischeduling Coscheduling (or gang scheduling) algorithms employ a global run queue that is accessed in round-robin fashion. The goal of coscheduling algorithms is to execute processes from the same job concurrently rather than maximize processor affinity.136 There are several coscheduling implementations—matrix, continuous and 15.6 Multiprocessor Scheduling 717 undivided. We present only the undivided algorithm, because it corrects some of the deficiencies of the matrix and continuous algorithms. The undivided coscheduling algorithm places processes from the same job in adjacent entries in the global run queue (Fig. 15.13). The scheduler maintains a "window" equal to the number of processors in the system. All processes within a window execute in parallel for at most one quantum. After scheduling a group of processes, the window moves to the next group of processes, which also execute in parallel for one quantum. To maximize processor utilization, if a process in the window is suspended, the algorithm extends the sliding window one process to the right to allow another runnable process to run for the given quantum, even if it is not part of a job that is currently executing. Because coscheduling algorithms use a round-robin strategy, they prevent indefinite postponement. Furthermore, because processes from the same job often run at the same time, coscheduling algorithms allow programs that are designed to run in parallel to take advantage of a multiprocessing environment. Unfortunately, the processor on which a process might be dispatched to a different processor each time, which can reduce processor affinity.137 Dynamic Partitioning Dynamic partitioning minimizes the performance penalty associated with cache misses by maintaining high processor affinity.138 The scheduler evenly distributes processors in the system among jobs. The number of processors allocated to a job is always less than or equal to the job's number of runnable processes. For example, consider a system that contains 32 processors and executes three jobs—the first with eight runnable processes, the second with 16 and the third with 20. If the scheduler were to divide the processors evenly between jobs, one job would receive 10 processors and the other two jobs 11. In this case, the first job has only eight processes, so the scheduler assigns eight processors to that job and evenly distributes the remaining 24 processors (12 each) to the second and third jobs (Example 1 in Fig. 15.14). Therefore, a particular job always executes on a certain Figure 15.13 | Coscheduling (undivided version). 718 Multiprocessor Management Figure 15.14 | Dynamic partitioning. subset of processors as long as no new jobs enter the system. The algorithm can be extended so that a particular process always executes on the same processor. If every job contains only one process, dynamic partitioning reduces to a round-robin scheduling algorithm.139 As new jobs enter the system, the system dynamically updates the processor allocation. Suppose a fourth job containing 10 runnable processes (Example 2 in Fig. 15.14) enters the system. The first job retains its allocation of eight processors, but the second and third jobs each relinquish four processors to the fourth job. Thus, the processors are evenly distributed among the jobs—eight processors per job.140 The algorithm updates the number of processors each job receives whenever jobs enter or exit the system, or a process within a job changes state from running to waiting or vice versa. Although the number of processors allocated to jobs changes, a job still executes on either a subset or superset of its previous allocation, which helps maintain processor affinity.141 For dynamic partitioning to be effective, the performance increase from cache affinity must outweigh the cost of repartitioning.142 Self Review 1. How are RRJob and undivided coscheduling similar? How are they different? 2. Describe some of the trade-offs between implementing a global scheduling policy that maximizes processor affinity, such as dynamic partitioning, and per-processor run queues. 15.7 Process Migration 719 Ans: 1) RRJob and undivided coscheduling are similar in that both schedule processes of a job to execute concurrently in round-robin fashion. Undivided coscheduling simply places processes of the same job next to each other in the global run queue where they wait for the next available processor. RRJob schedules only entire jobs. 2) Global scheduling is more flexible because it reassigns processes to different processors depending on system load. However, per-processor run queues are simpler to implement and can be more efficient than maintaining global run queue information. 15.7 Process Migration Process migration entails transferring a process between two processors.143-144 This might occur if, for example, a processor fails or is overloaded. The ability to execute a process on any processor has many advantages. The most obvious is that processes can move to processors that are underutilized to reduce process response times and increase performance and throughput. 145 - 146 (We describe this technique, called load balancing, in more detail in Section 15.8, Load Balancing.) Process migration also promotes fault tolerance. 147 For example, consider a program that must perform intensive, uninterrupted computation. If the machine running the program needs to be shut down or becomes unstable, the program's progress might be lost. Process migration allows the program to move to another machine to continue computation perhaps in a more stable environment. In addition, process migration promotes resource sharing. In large-scale systems, some resources might not be replicated at every node. For example, in a NORMA system, processes might execute on machines with different hardware device support. A process might require access to a RAID array that is available through only one computer. In this case, the process should migrate to the computer with access to the RAID array for better performance. Finally, process migration improves communication performance. Two processes that communicate frequently should execute on or near the same node to reduce communication latency. Because communication links are often dynamic, process migration can be used to make process placement dynamic.148 Self Review 1. What are some benefits provided by process migration? 2. On which types of systems (UMA, NUMA or NORMA) is process migration most appropriate? Ans: 1) Process migration promotes fault tolerance by moving processes away from malfunctioning nodes, supports load balancing, can reduce communication latency and promotes resource sharing. 2) Process migration is appropriate for large-scale NUMA or NORMA systems that use per-processor (or per-node) run queues. Process migration enables these systems to perform load balancing and share resources local to each node. 15.7.1 F l o w of Process Migration Although process migration implementations vary across architectures, many implementations follow the same general steps (Fig. 15.15). First, a node issues a 720 Multiprocessor Management Figure 15.15 | Process migration. migration request to a remote node. In most schemes, the sender initiates the migration because its node

is overloaded or a specific process needs to access a resource located at a remote node. In some schemes, an underutilized node might request processes from other nodes. If the sender and receiver agree to migrate a process, the sender suspends the migrating process. The sender creates a message queue to hold all messages destined for the migrating process. Then, the sender extracts the process's state; this includes copying the process's memory contents (i.e., pages marked as valid in the process's virtual memory), register contents, state of open files and other process-specific information. The sender transmits the extracted state to a "dummy" process that the receiver creates. The two nodes notify all other processes of the migrated process's new location. Finally, the receiver dispatches the new instance of the process, the sender forwards the mes- 15.7 Process Migration 721 sages in the migrated process's message queue and the sender destroys the local instance of the process.149-150 Self Review 1. List some items that compose a process's state and that must be migrated with a process. 2. What overhead is incurred in process migration? Ans: 1) The sending process must transfer such items as the process's memory contents, register contents and state of open files. 2) The overhead includes migrating the process and its state, rerouting messages to the receiving node, maintaining two instances of the process for a short time (the old instance and the "dummy process"), suspending the process for a short time and sending messages to other nodes to inform them about the migrated process's new location. 15.7.2 Process Migration Concepts Memory transfer is the most time-consuming element of migration. 151 To minimize the process migration's performance cost, residual dependency—a process's dependency on its former node—must be minimized. For example, the process's former node might contain part of the process's working set or might be executing other processes with which the migrated one was communicating. If migration results in many residual dependencies, a receiving node must communicate with the sending node while executing the migrated process. This increases IPC, which generates more traffic on the interconnection network and degrades performance due to high latencies. Also, residual dependencies decrease fault tolerance, because the migrated process's execution depends on both nodes functioning correctly.152 Often, the strategies that result in the highest degree of residual dependency transfer pages from the sender only when the process at the receiving node references them. These lazy (or on-demand) migration strategies reduce the initial process migration time —the duration for which the migrating process is suspended. In lazy migration, the process's state (and other required) information is transferred to the receiver, but the original node retains the process's pages. As the process executes on the remote node, it must initiate a memory transfer for each access to a page that remains at the sending node. Although this technique yields fast initial process migration, memory access can severely decrease an application's performance. Lazy migration is most useful when the process does not require frequent access to the remote address space.153-154 For successful migration, processes should exhibit several characteristics. A migrated process should exhibit transparency—i.e., it should not be adversely affected (except, perhaps, for a slight delay in response time) by migration. In other words, the process should lose no interprocess messages or open file handles.155 A system also must be scalable—if process's residual dependencies grow with each migration, then a system could quickly be overwhelmed by network traffic as processes request remote pages. Finally, advances in communication technology across multiple architectures have created the need for heterogeneous process migra- 722 Multiprocessor Management tions—processes should be able to migrate between two different processor architectures in distributed systems. This implies that process state should be stored in a platform-independent format.156 Section 17.3.6, Process Migration in Distributed Systems, discusses heterogeneous process migration architectures in more detail. Self Review 1. Why is residual dependency undesirable? Why might some residual dependency be beneficial? 2. How might a migration strategy that results in significant residual dependency not be scalable? Ans: 1) Residual dependency is undesirable because it decreases fault tolerance and degrades performance after the transfer. One reason for allowing residual dependency is that it reduces initial transfer time. 2) If a process that already has a residual dependency migrates again, it will now rely on three nodes. With each migration, the process's state grows (so it can find its memory pages) and its fault tolerance decreases. 15.7.3 Process Migration Strategies Process migration strategies must balance the performance penalty of transferring large amounts of process data with the benefit of minimizing a process's residual dependency. In some systems, designers assume that most of a migrated process's address space will be accessed at its new node. These systems often implement eager migration, which transfers all of a process's pages during initial process migration. This enables the process to execute as efficiently at its new node as it did at its previous one. However, if the process does not access most of its address space, the initial latency and the bandwidth required by eager migration are largely overhead.157, 158 To mitigate eager migration's initial cost, dirty eager migration transfers only a process's dirty pages. This strategy assumes that there is common secondary storage (e.g., a disk to which both nodes have access). All clean pages are brought in from the common secondary storage as the process references them from the remote node. This reduces initial transfer time and eliminates residual dependency. However, each access to a nonresident page takes longer than it would have using eager migration.159 One disadvantage of dirty eager migration is that the migrated process must use secondary storage to retrieve the process's clean pages. Copy-on-reference migration is similar to dirty eager migration, except that the migrated process can request clean pages from either its previous node or the common secondary storage. This strategy has the same benefits as dirty eager migration but gives the memory manager more control over the location from which to request pages —access to remote memory may be faster than disk access. However, copy-on-reference migration can add memory overhead at the sender.160-161 The lazy copying implementation of copy-on-reference transfers only the minimum state information at initial migration time; often, no pages are transferred. This creates a large residual dependency, forces the previous node to keep pages in memory for the migrated process and increases memory-access latency over strategies that migrate dirty pages. However, the lazy copying strategy eliminates most 15.8 Load Balancing 723 initial migration latency.162' 163 Such latency might be unacceptable for real-time processes and is inappropriate for most processes.164 All the strategies discussed so far either create residual dependency or incur a large initial migration latency. A method that eliminates most of the initial migration latency and residual dependency is the flushing strategy. In this strategy, the sender writes all pages of memory to shared secondary storage when migration begins; the migrating process must be suspended while the data is being written to secondary storage. The process then accesses the pages from secondary storage as needed. Therefore, no actual page migration occurs to slow initial migration, and the process has no residual dependency on the previous node. However, the flushing strategy reduces process performance at its new node, because the process has no pages in main memory, leading to page faults.165-166 Another strategy for eliminating most of the initial migration latency and residual dependency is the precopy method, in which the sender node begins transferring dirty pages before the original process is suspended. Any transferred page that the process modifies before it migrates is marked for retransmission. To ensure that a process eventually migrates, the system defines a lower threshold for the number of dirty pages that should remain before the process migrates. When that threshold is reached, the process is suspended and migrated to another processor. With this technique, the process does not need to be suspended for long (compared to other techniques such as eager and copy-on-reference), and memory access at the new node is fast because most data has already been copied. Also, residual dependency is minimal. The process's working set is duplicated for a short time (it exists at both nodes involved in migration), but this a minor drawback. 167168 Self Review 1. Which migration strategies should be used with real-time processes? 2. Although initial migration time is minimal and there is little residual dependency for the precopy strategy, can you think of any "hidden" performance costs this strategy incurs? Ans: 1) Real-time processes cannot be suspended for long because this would slow response time. Therefore, strategies such as lazy copying, flushing, copy-on-reference and precopy are best for soft real-time processes. However, hard real-time processes should not be migrated because migration always introduces some indeterministic delay. 2) The process must continue executing on the sender node while its state is being copied to the receiving node. Because a migration decision was made, it is reasonable to assume that executing on the sending node is no longer desirable and therefore, precopy includes this additional cost. 15.8 Load Balancing One measure of efficiency in a multiprocessor system is overall processor utilization. In most cases, if processor utilization is high, the system is performing more efficiently. Most multiprocessor systems (especially NUMA and NORMA systems) attempt to maximize processor affinity. This increases efficiency, because the processes do not need to access remote resources as often, but it might reduce processor 724 Multiprocessor Management utilization if all the processes assigned to a particular processor complete. This processor will idle while processes are dispatched to other processors to exploit affinity. Load balancing is a technique in which the system attempts to distribute processing loads evenly among processors. This increases processor utilization and shortens run queues for overloaded processors, reducing average process response times.169 A load balancing algorithm might assign a fixed number of processors to a job when the job is first scheduled. This is called static load balancing. This method yields low runtime overhead, because processors spend little time determining the processors on which a job should execute. However, static load balancing does not account for varying process populations within a job. For example, a job might include many processes initially, but maintain only a few processes throughout the rest of its execution. This can lead to unbalanced run queues, which may lead to processor idling.170 Dynamic load balancing attempts to address this issue by adjusting the number of processors assigned to a job throughout its life. Studies have shown that dynamic load balancing performs better than static load balancing when contextswitch time is low and system load is high.171 Self Review 1. What are some of the benefits of implementing load balancing? 2. What scheduling algorithms benefit from load balancing? Ans: 1) Some of the benefits of load balancing include higher processor utilization, which leads to higher throughput, and reduced process response time. 2) Load balancing is appropriate for scheduling algorithms that maximize processor affinity, such as dynamic partitioning. Also, it can benefit systems that maintain per-processor or per-node run queues. 15.8.1 Static Load Balancing Static load balancing is useful in environments in which jobs repeat certain tests or instructions and therefore exhibit predictable patterns (e.g., scientific computing). 172 These patterns can be represented as graphs that can be used to model scheduling. Consider the processes in a system as vertices in a graph, and communications between processes as edges. For example, if there is an application in which one process continually analyzes similar data, then feeds that data to another process, this can be modeled as two nodes connected by one edge. Because this relationship will be consistent throughout the application's life, there is no need to adjust the graph. Due to shared cache and physical memory, communication between processes at the same processor is much faster than communication between processes at different processors. Therefore, static load balancing algorithms attempt to divide the graph into subgraphs of similar size (i.e., each processor has a similar number of processes) while minimizing edges between subgraphs to reduce communication between processors.173 However, this technique can incur significant overhead for a large number of jobs.174 Consider the graph in Fig. 15.16. Two dashed lines represent 15.8 Load Balancing 725 Figure 15.16 | Static load balancing using graphs. possible cuts to divide the processes somewhat evenly. Cut #1 yields four interprocessor communication channels, whereas cut #2 yields only two, thus representing a better grouping of processes. Static load balancing can be inadequate when communication patterns change dynamically and when processes complete with no new processes to take their places. The first case might perform less efficiently due to high communication latency. In the second case, processor utilization could decrease even when there are processes waiting to obtain a processor. In these cases, dynamic load balancing can improve performance. 175 Self Review 1. When is static load balancing useful? When is it not useful? 2. In Fig. 15.16, there is a substantial difference between cut #1 and cut #2. Considering that it is difficult to find the most effective cut when there are large numbers of jobs, what does this say about the limitations of static load balancing? Ans: 1) Static load balancing is useful in environments where processes might exhibit predictable communication patterns. It is not useful in environments where communication patterns change dynamically and processes are created or terminated unpredictably. 2) The difference between Cut #1 (four IPC channels) and Cut #2 (two IPC channels) illustrates the performance implications of making bad decisions. Most large systems will have to estimate to find a solution; a wrong estimate could severely degrade performance. 15.8.2 Dynamic Load Balancing Dynamic load balancing algorithms migrate processes after they have been created in response to system load. The operating system maintains statistical processor load information such as the number of active and blocked processes at a processor, average processor utilization, turnaround time and latency.176-177 If many of a processor's processes are blocked or have a large turnaround times, the processor most likely is overloaded. If a processor does not have a high processor utilization rate, it probably is underloaded. Several policies may be used to determine when to migrate processes in dynamic load balancing. The sender-initiated policy activates when the system determines that a processor contains a heavy load. Only then will the system search 726 Multiprocessor Management for underutilized processors and migrate some of the overloaded processor's jobs to them. This is best for systems with light loads, because process migration is costly and, in this case, the policy will rarely be activated.178, 179 Conversely, a receiver-initiated policy is better for overloaded systems. In this environment, the system initiates process migration when a processor's utilization is low.180, 181 Most systems experience heavy and light loads over time. For these systems, the symmetric policy, which combines the previous two methods, provides maximum versatility to adapt to environmental conditions.182, 183 Finally, the random policy, in which the system arbitrarily chooses a processor to receive a migrated process, has shown decent results due to its simple implementation and (on average) even distribution of processes. The motivation behind the random policy is that the migrating process's destination will likely have a smaller load than its origin, considering that the original processor is severely overloaded. 184, 185 The subsections that follow describe common algorithms that determine how processes are migrated. For the purposes of this discussion, consider that the multiprocessor system can be represented by a graph in which each processor and its memory are a vertex and each link is an edge. Bidding Algorithm The bidding algorithm is a simple sender-initiated migration policy. Processors with smaller loads "bid" for processes on overloaded processors, much as in an auction. The value of a bid is based on the current load of the bidding processor and the distance between the underloaded and overloaded processors in the graph. To reduce the process migration cost, more direct communication paths to the overloaded processor receive higher bid values. The overloaded processor accepts bids from processors that are within a certain distance on the graph. If the overloaded processor receives too many bids, it decreases the distance; if it receives too few, it increases the distance and checks again. The process is sent to the processor with the highest bid.186 Drafting Algorithm The drafting algorithm is a receiver-initiated policy that classifies the load at each processor as low, normal or high. Each processor maintains a table describing the other processors' loads using these classifications. Often in large-scale or distributed systems, processors maintain only information about their neighbors. Even time a processor's load changes classification, the processor broadcasts its updated information to the processors in its load table. When a processor receives one of these messages, it appends its own information and forwards the message to its neighbors. In this way, information about load levels eventually reaches all nodes in the network. Underutilized processors use this information to request processes from overloaded processors.187 15.8 Load Balancing 727 Communications Issues Inefficient or incorrect communication strategies can overwhelm a system. For example, some process migration implementations employ systemwide broadcasts. The flood of broadcast messages can overwhelm communication channels. Due to communication delays, many overloaded processors could receive a request for a process at the same time and all send their processes to one underloaded processor.188 Several strategies have been devised to avoid these problems. For example, the algorithm could restrict processors to communicate only with then immediate neighbors, which reduces the number of messages that are transmitted, but increases the time required for information to reach every node in the system.189 Alternatively, processors could periodically select a random processor with which to exchange information. In this case, processes are migrated from the processor with the higher load to the one with a lower load.190 In cases where one processor is severely overloaded and the rest are underloaded, this results in rapid process diffusion. Figure 15.17 illustrates process diffusion in a system in which nodes communicate only with their neighbors. The overloaded processor, represented by the middle vertex, has 17 processes while all others have one process. After one iteration, the overloaded processor communicates with its neighbors and sends three processes to each. Now those processors have four processes, which is three more than some of their neighbors have. In the second iteration, the processors with four processes send some to their neighbors. Finally, in the third iteration, the overloaded processor once again sends some processes to its neighbors so that now the processor with the heaviest load has only three processes. This example illustrates that even when communications are kept among neighboring processors, load balancing can effectively distribute processing responsibilities throughout the system. System at start: Figure 15.17 | Processor load diffusion. (Part 1 of 2.) 728 Multiprocessor Management . . . After one iteration ...After two iterations ...After three iterations: Figure 15.17 | Processor load diffusion. (Part 2 of 2.) 15.9 Multiprocessor Mutual Exclusion 729 Self Review 1. In what type of environment should a sender-oriented policy be used? A receiver-oriented policy? Justify your answers. 2. How does high communication latency hinder the effectiveness of load balancing? Ans: 1) A system with a light load should use a sender-oriented policy, whereas a system with a heavy load should use a receiver-oriented policy. In both cases this will reduce unnecessary migrations, because there will be few senders in a system with a light load and few receivers in one with a heavy load. 2) A node sending a message that it is underloaded might have become overloaded by the time another node receives the message. Thus, the receiving node could further unbalance the system's load by migrating additional processes to the sender. 15.9 Multiprocessor Mutual Exclusion Many of the mutual exclusion mechanisms we described in Chapter 5 are not adequate for multiprocessor systems. For example, disabling interrupts, which prevents other processes from executing by disabling preemption on uniprocessor systems, does not guarantee mutual exclusion on multiprocessors because several processes can execute simultaneously on different processors. Instructions such as test-and-set can be inefficient when the data these instructions reference is located in remote memory. Test-and-set has another weakness, namely that each of the pending operations requires a separate memory access; these are performed sequentially, because only one access to a memory location may normally occur at a time. Such collisions can quickly saturate various kinds of interconnection networks, causing serious performance problems. Because of nontrivial communication costs and the fact that multiple processes execute at the same time, designers have developed a number of mutual exclusion techniques for multiprocessors. This section introduces mutual exclusion techniques for nondistributed multiprocessing systems. We present distributed mutual exclusion in Section 17.5, Mutual Exclusion in Distributed Systems. 15.9.1 Spin Locks Multiprocessor operating systems, such as Windows XP and Linux, frequently use spin locks for multiprocessor mutual exclusion. A spin lock is called a "lock" because a process that holds a spin lock claims exclusive access to the resource the spin lock protects (e.g., a shared data structure in memory or a critical section of code). In effect, other processes are "locked out" of the resource. Other processes that wish to use the resource "spin" (i.e., busy wait) by continually testing a condition to determine whether the resource is available. In uniprocessing systems, spin locks are wasteful because they consume processor cycles that could be used by other processes, including the one that holds the lock, to do useful work. In multiprocessing systems, the process holding the lock could release it while another process is busy waiting. In this case, it is more efficient for a process to busy wait if the time required for the system to perform a context switch to schedule another process is longer than the average busy-wait time. 191 Also, if a processor contains no 730 Multiprocessor Management other runnable processes in its run queue, it makes sense to keep the process spinning to minimize the response time when the spin lock is released. 192 ' 193 When a process holds a spin lock for a long duration (relative to contextswitch time), blocking is more efficient. Delayed blocking is a technique in which a process spins for a short period of time; if it does not acquire the lock in that time, the process blocks.194 An advisable process lock (APL) presents an alternative solution. When a process acquires an APL, it specifies the amount of time it will hold the lock. Based on the time specified, other processes waiting to acquire the APL can determine if it is more efficient to busy wait for the lock or to block.195 Adaptive locks (also called configurable locks) add flexibility to a mutual exclusion implementation. At certain times, such as when there are few active processes, spin locks are preferable because a process can acquire a resource soon after it becomes available. At times when system load is high, spinning wastes valuable processor cycles and blocking is preferable. Adaptive locks permit a process to dynamically change the type of lock being used. These locks can be set to blocking, spinning or delayed blocking and can incorporate APL features to customize the lock according to system load and application needs. 196 When multiple processes simultaneously wait for a spin lock to be released, indefinite postponement can occur. Operating systems can prevent indefinite postponement by granting the spin lock to processes in first-come-first-served order, or by aging processes waiting on the spin lock, as discussed in Section 7.3. Self Review 1. In what way can spinning be useful in a multiprocessor? Under what conditions is it not desirable in a multiprocessor? Why is spinning not useful in a uniprocessor system? 2. Why might an APL not be useful in all situations? Ans: 1) Spinning can be useful because it minimizes the time between when a resource becomes available and when a new process acquires the resource. Spinning is not desirable when system load is high and the expected time before the resource is available is greater than context-switch time; in these cases, spinning uses valuable processor cycles. Spinning is not useful in uniprocessor systems because it blocks the execution of the process holding the resource. 2) An APL is not useful when a process cannot determine how long it will hold the lock. 15.9.2 Sleep/Wakeup Locks A sleep/wakeup lock provides synchronization similar to a spin lock, but reduces wasted processor cycles and bus traffic. Consider processes P1 and P 2 , both of which request use of a resource protected by a sleep/wakeup lock. P1 requests the resource first and obtains the lock. When P2 requests the resource owned by P1 and does not receive the resource, P 2 responds by sleeping (i.e., blocking). Once P 1 releases the lock, P1 wakes the highest-priority process waiting for the resource (in this case, P 2). Unlike spin locks (which give the lock to the next waiting process), sleep/wakeup locks can use the processor scheduler to enforce process priorities. 15.9 Multiprocessor Mutual Exclusion 731 Sleep/wakeup locks can cause waiting processes to be indefinitely postponed, depending on the system's scheduling policy. Note that unlike a uniprocessor implementation, only the process with the highest priority is awakened. When all threads are awakened, a race condition can result because two or more threads may access a resource associated with a lock in a nondeterministic order. Race conditions should be avoided because they can cause subtle errors in applications and are difficult to debug. In uniprocessors, even if all processes are alerted, there can be no race condition because only one process (the one with highest priority in most scheduling algorithms) obtains control of the processor and acquires the lock. In a multiprocessing environment, a broadcast could wake many processes that are competing for the lock, creating a race condition. This would also result in one process obtaining the lock and many processes testing, reblocking and consequently going back to sleep, thus squandering processor time due to context switching. This phenomenon is known as the thundering herd.197 Self Review 1. What are an advantage and a disadvantage of a sleep/wakeup lock compared to a spin lock? 2. How is the implementation of a sleep/wakeup lock different in a multiprocessor environment than in a uniprocessor environment? Ans: 1) A sleep/wakeup lock eliminates wasted processor cycles incurred by spinning. Also, a sleep/wakeup lock ensures that the highest-priority waiting process obtains the lock next. A disadvantage is that reaction to the lock's availability is slower because the new acquirer must awaken and obtain a processor. 2) In a multiprocessor environment, a releasing process awakens only one waiting process, which prevents a thundering herd that would waste processor cycles and flood communication channels. This is not a problem on uniprocessor systems. 15.9.3 Read/Write Locks Enforcing mutually exclusive access to shared memory in a multiprocessor system can degrade performance. Only writers need exclusive access to the resource, whereas, in general, multiple readers can access the same memory location at once. Therefore, many systems protect shared memory with a more versatile read/write lock rather than a generic mutual exclusion lock.198 A read/write lock provides mutual exclusion similar to that presented in the readers/writers problem of Section 6.2.4. A read/write lock permits multiple reader processes (i.e., processes that will not alter shared data) to enter their critical sections. Unlike the example in Section 6.2.4, however, read/write locks require that a writer (i.e., a process that will alter shared data) wait until there are no reader or writer processes in their critical sections before entering its critical section.199 To implement this locking approach efficiently, the mutual exclusion mechanism must use shared memory. However, in environments in which memory is not shared, such as NORMA systems, message passing must be used. We discuss such techniques in Section 17.5, Mutual Exclusion in Distributed Systems. 732 Multiprocessor Management Self Review 1. In what situations are read/write locks more efficient than spin locks? 2. A naive implementation of a read/write lock allows any reader to enter its critical section when no writer is in its critical section. How can this lead to indefinite postponement for writers? What is a fairer implementation? Ans: 1) Read/write locks are more efficient than spin locks when multiple processes read a memory location, but do not write to it. In this case, multiple processes can be in critical sections at once. 2) This implementation leads to indefinite postponement if readers continually enter their critical sections before a writer can enter its critical section. A fairer implementation would allow a reader to enter its critical section only if no writer is in or is waiting to enter its critical section as discussed in Section 6.2.4. Web Resources www.winntmag.com/Articles/Index.cfm?ArticleID=303&pg=l&show=495 This article describes multiprocessor scheduling in Windows NT. www.aceshardware.com/Spades/read.php?article_id= 30000187 This article describes how the AMD Athlon (with two processors) implements cache coherence. It also discusses bus snooping. www.mosix.org This site provides information about MOSIX, a software bundle used on UNIX systems, which implements automatic process migration and load balancing. www.teamten.com/lawrence/242.paper/242.paper.html This paper discusses mutual exclusion algorithms for multiprocessors. users.win.be/W0005997/UNIX/LINUX/IL/atomicityeng.html This site explains how to use spin locks to implement mutual exclusion in Linux. Summary Many applications demand substantially more computing power than one processor can provide. As a result, multiprocessing systems—computers that use more than one processor to meet a system's processing needs—are often employed. The term "multiprocessing system" encompasses any system with more than one processor. This includes dual-processor personal computers, powerful servers that contain many processors and distributed groups of workstations that work together to perform tasks. There are several ways to classify multiprocessors. Flynn developed an early scheme for classifying computers into increasingly parallel configurations based on the types of streams used by processors. SISD (single instruction stream, single data stream) computers are traditional uniprocessors that fetch one instruction at a time and execute it on a single data item. MISD (multiple instruction stream, single data stream) computers (which are not commonly used) have multiple processing units, each operating on a piece of data and then passing the result to the next pro- cessing unit. SIMD (single instruction stream, multiple data stream) computers, which include array and vector processors, execute instructions on multiple data items in parallel. MIMD (multiple instruction stream, multiple data stream) computers have multiple processing units that operate independently on separate instruction streams. The interconnection scheme of a multiprocessor systern describes how the system physically connects its components, such as processors and memory modules. The interconnection scheme affects the system's performance, reliability and cost, so it is a key issue for multiprocessor designers. A shared bus provides low cost and high performance for a small number of processors but does not scale well, owing to contention on the bus as the number of processors increases. A crossbar-switch matrix provides high fault tolerance and performance but is inappropriate for small systems in which UMA (uniform memory access) is more cost effective. A 2-D mesh network is a simple design that provides adequate performance and fault tolerance at Summary a low cost but that also does not scale well. A hypercube is more scalable than a 2-D mesh network and provides better performance, but at a higher cost. Multistage networks compromise between cost and performance and can be used to build extremely large-scale multiprocessors. In a tightly coupled system, the processors share most system resources. Loosely coupled systems connect components indirectly through communication links and do not share most resources. Loosely coupled systems are more scalable, flexible and fault tolerant but do not perform as well as tightly coupled systems. Loosely coupled systems also place more burden on programmers, who must implement applications that communicate via IPC rather than shared memory. Multiprocessors can also be categorized based on how the processors share operating system responsibilities. In the master/slave organization, only the master processor can execute the operating system; the slaves execute only user programs. In the separate-kernels organization, each processor executes its own operating system, and operating system data structures maintain the system's global information. In the symmetrical organization, all processors can control any device and reference any storage unit. This organization enables systems to balance their workloads more precisely. Multiprocessor systems use several architectures to share memory. UMA multiprocessors require all processors to share all the system's main memory equally owing to shared-memory contention; these systems are not scalable beyond a few processors. NUMA (nonuniform memory access) multiprocessors partition memory into modules, assigning one to each processor as its local memory. COMA (cache-only memory architecture) multiprocessors are similar to NUMA multiprocessors, but treat all memory as a large cache to increase the likelihood that requested data resides in the requesting processor's local memory. NORMA (no remote memory access) multiprocessors are loosely coupled and do not provide any globally shared main memory. NORMA multiprocessors are used to build large-scale distributed systems. Memory is coherent if the value obtained from reading a memory address is always the same as the value most recently written to that address. UMA cache-coherence protocols include bus snooping and directory-based coherence. CC-NUMAs (cache-coherent NUMAs) often use a homebased approach in which a home node for each memory address is responsible for keeping that data item coherent throughout the system. Systems that use page replication maintain multiple copies of a memory page so it can be accessed quickly by 733 multiple processors. Systems that use page migration transfer pages to the processor that accesses the pages most. These techniques can be combined to optimize performance. Shared virtual memory (SVM) provides the illusion of shared physical memory in large-scale multiprocessor systems. Invalidation (in which a writer voids other copies of a page) and write broadcast (in which a writer notifies other processors of updates to a page) are two approaches for implementing memory coherence in SVM systems. These protocols can be strictly applied, but this can be inefficient. Relaxed consistency, in which the system might not be coherent for a few seconds, improves efficiency but sacrifices some data integrity. Multiprocessor scheduling algorithms must determine both when and where to dispatch a process. Some algorithms maximize processor affinity (the relationship of a process to a particular processor and its local memory) by executing related processes on the same processors. Others maximize parallelism by executing related processes on separate processors simultaneously. Some systems use global run queues, whereas others (typically large-scale systems) maintain per-processor or per-node run queues. Process migration is the act of transferring a process between two processors or computers. Process migration can increase performance, resource sharing and fault tolerance. To migrate a process, nodes must transfer the process's state, which includes the process's memory pages, register contents, state of open files and kernel context. Migration policies that allow residual dependency (a process's dependency on its former node) decrease fault tolerance and the process's performance after migration. However, eliminating residual dependency makes initial migration slow. Various migration policies balance the goals of minimal residual dependency and fast initial migration. Load balancing is a technique in which the system attempts to distribute processing responsibility equally among processors to increase processor utilization and decrease process response times. Static load balancing algorithms assign a fixed number of processors to a job when the job is first scheduled. These algorithms are useful in environments, such as scientific computing, where process interactions and execution times are predictable. Dynamic load balancing algorithms change the number of processors assigned to a job throughout the job's life. These algorithms are useful when process interactions are unpredictable, and when processes can be created and terminate at any time. Many uniprocessor mutual exclusion mechanisms are either inefficient or ineffective for multiprocessors. Design- 734 Multiprocessor Management ers have developed mutual exclusion techniques specific to multiprocessors. A spin lock is a mutual exclusion lock in which waiting processes spin (i.e., busy wait) for the lock. This is appropriate when system load is low and spinning time is short relative to context-switch time. Spin locks reduce reaction time when resources become available. With a sleep/wakeup lock, a process releasing the lock wakes up the highest-priority waiting process, which acquires the lock. This reduces the use of processor cycles typical in spin locks. Read/write locks allow one writer or many readers to be in their critical sections at once. Key Terms 2-D mesh network—Multiprocessor interconnection scheme that arranges nodes in an m x n rectangle. 4-connected 2-D mesh network—2-D mesh network in which nodes are connected with the nodes directly to the north, south, east and west. acquire operation—In several coherence strategies, an operation indicating that a process is about to access shared memory. adaptive lock—Mutual exclusion lock that allows processes to switch between using a spin lock or a blocking lock, depending on the current condition of the system. advisable process lock (APL)—Locking mechanism in which an acquirer estimates how long it will hold the lock; other processes can use this estimate to determine whether to block or spin when waiting for the lock. cache-coherent NUMA (CC-NUMA)—NUMA multiprocessor that maintains cache coherence, usually through a home-based approach. cache miss latency—Extra time required to access data that does not reside in the cache. cache-only memory architecture (COMA) multiprocessor Multiprocessor architecture in which nodes consist of a processor, cache and memory module; main memory is organized as a large cache. cache snooping—Bus snooping used to ensure cache coherency. configurable lock—See adaptive lock contention—In multiprocessing a situation in which several processors compete for the use of a shared resource. array processor—SIMD (single instruction stream, multiple data stream) system consisting of many (possibly tens of thousands) simple processing units, each executing the same instruction in parallel on many data elements. copy-on-reference migration—Process migration technique in which only a process's dirty pages are migrated with the process, and the process can request clean pages either from the sending node or from secondary storage. attraction memory (AM)—Main memory in a COMA (cacheonly memory architecture) multiprocessor, which is organized as a cache. cost of an interconnection scheme—Total number of links in a network. baseline network—Type of multistage network. bidding algorithm—Dynamic load balancing algorithm in which processors with smaller loads "bid" for jobs on overloaded processors; the bid value depends on the load of the bidding processor and the distance between the underloaded and overloaded processors. coscheduling—Job-aware process scheduling algorithm that attempts to execute processes from the same job concurrently by placing them in adjacent global-run-queue locations. crossbar-switch matrix—Processor interconnection scheme that maintains a separate path from every sender node to every receiver node. bisection width—Minimum number of links that need to be severed to divide a network into two unconnected halves. degree of a node—Number of other nodes with which a node is directly connected. bus snooping—Coherence protocol in which processors "snoop" the shared bus to determine whether a requested write is to a data item in that processor's cache or, if applicable, local memory. delayed blocking—Technique whereby a process spins on a lock for a fixed amount of time before it blocks; the rationale is that if the process does not obtain the lock quickly, it will probably have to wait a long time, so it should block. cache coherence—Property of a system in which any data item read from a cache has the value equal to the last write to that data item. delayed consistency—Memory coherence strategy in which processors send update information after a release, but a Key Terms 735 receiving node does not apply this information until it performs an acquire operation on the memory. node forwards update information to other nodes that subsequently access the page. dirty eager migration—Process migration method in which only a process's dirty pages are migrated with the process; clean pages must be accessed from secondary storage. home node—Node that is the "home" for a physical memory address or page and is responsible for maintaining the data's coherence. drafting algorithm—Dynamic load balancing algorithm that classifies each processor's load as low, normal or high; each processor maintains a table of the other processors' loads, and the system uses a receiver-initiated policy to exchange processes. hypercube—Multiprocessor interconnection scheme that consists of 2n nodes (where n is an integer); each node is linked with n neighbor nodes. dynamic load balancing—Technique that attempts to distribute processing responsibility equally by changing the number of processors assigned to a job throughout the job's life. dynamic partitioning—Job-aware process scheduling algorithm that divides processors in the system evenly among jobs, except that no single job can be allocated more processors than runnable processes; this algorithm maximizes processor affinity. eager migration—Process migration strategy that transfers the entire address space of a process during the initial phases of migration to eliminate a migrated process's residual dependencies on its original node. false sharing—Situation that occurs when processes on separate processors are forced to share a page because they are each accessing a data item on that page, although not the same data item. first-come-first-served (FCFS) process scheduling—Job-blind multiprocessor scheduling algorithm that places arriving processes in a queue; the process at the head of the queue executes until it freely relinquishes the processor. flushing—Process migration strategy in which the sending node writes all of the process's memory pages to a shared disk at the start of migration; the process then accesses the pages from the shared disk as needed on the receiving node. gang scheduling—Another name for coscheduling. global run queue—Process scheduling queue used in some multiprocessor scheduling algorithms, which is independent of the processors in a system and into which every process or job in the system is placed. hard affinity—Type of processor affinity in which the scheduling algorithm guarantees that a process only executes on a single node throughout its life cycle. home-based consistency—Memory-coherence strategy in which processors send coherence information to a home node associated with the page being written; the home interconnection scheme—Design that describes how a multiprocessor system physically connects its components, such as processors and memory modules. invalidation—Memory-coherence protocol in which a process first invalidates—i.e., voids—all other copies of a page before writing to the page. job-aware scheduling—Multiprocessor scheduling algorithms that account for job properties when making scheduling decisions; these algorithms typically attempt to maximize parallelism or processor affinity. job-blind scheduling—Multiprocessor scheduling algorithms that ignore job properties when making scheduling decisions; these algorithms are typically simple to implement. lazy copying—Process migration strategy that transfers pages from the sender only when the process at the receiving node references these pages. lazy data propagation—Technique in which writing processors send coherence information after a release, but not the data; a processor retrieves the data when it accesses a page that it knows is not coherent. lazy migration—Process migration strategy in multiprocessor systems that does not transfer all pages during initial migration. This increases residual dependency but reduces initial migration time. lazy release consistency—Memory-coherence strategy in which a processor does not send coherence information after writing to a page until a new processor attempts an acquire operation on that memory page. load balancing—Technique which attempts to distribute the system's processing responsibility equally among processors. loosely coupled system—System in which processors do not share most resources; these systems are flexible and fault tolerant but perform worse than tightly coupled systems. massively parallel processor—Processor that performs a large number of instructions on large data sets at once; array processors are often called massively parallel processors. master/slave multiprocessor organization—Scheme for delegating operating system responsibilities in which only one 736 Multiprocessor Management processor (the "master") can execute the operating system, and the other processors (the "slaves") can execute only user processes. page migration—Technique in which the system transfers pages to the processor (or processors when used with page replication) that accesses the pages most. memory coherence—State of a system in which the value obtained from reading a memory address is always the same as the most-recently written value at that address. page replication—Technique in which the system maintains multiple copies of a page at different nodes so that it can be accessed quickly by multiple processors. memory line—Entry in memory that stores one machine word of data, which is typically four or eight bytes. per-processor run queue—Process scheduling queue associated with a specific processor; processes entering the queue are scheduled on the associated processor independently of the scheduling decisions made in rest of the system. multiple-instruction-stream, multiple-data-stream (MIMD) computer—Computer architecture consisting of multiple processing units, which execute independent instructions and manipulate independent data streams; this design describes multiprocessors. multiple-instruction-stream, single-data-stream (MISD) computer—Computer architecture consisting of several processing units, which execute independent instruction streams on a single stream of data; these architectures have no commercial application. multiple shared bus architecture—Interconnection scheme that employs several shared buses connecting processors and memory. This reduces contention, but increases cost compared to a single shared bus. multiprocessing system—Computing system that employs more than one processor. multistage network—Multiprocessor interconnection scheme that uses switch nodes as hubs for communication between processor nodes that each have their own local memory. network diameter—Shortest path between the two most remote nodes in a system. network link—Connection between two nodes. node—System component, such as a processor, memory module or switch, attached to a network; sometimes a group of components might be viewed as a single node. nonuniform-memory-access (NUMA) multiprocessor—Multiprocessor architecture in which each node consists of a processor, cache and memory module. Access to a processor's associated memory module (called local memory) is faster than access to other memory modules in the system. no-remote-memory-access (NORMA) multiprocessor—Multiprocessor architecture that does not provide global shared memory. Each processor maintains its own local memory. NORMA multiprocessors implement a common shared virtual memory. on-demand migration—Another name for lazy process migration. per-node run queue—Process scheduling queue associated with a group of processors; processes entering the queue are scheduled on the associated node's processors independently of the scheduling decisions made in rest of the system. precopy—Process migration strategy in which the sender begins transferring dirty pages before the original process is suspended; once the number of untransferred dirty pages at the sender reaches some threshold, the process migrates. process migration—Transferring a process and its associated state between two processors. processor affinity—Relationship of a process to a particular processor and a corresponding memory bank. race condition—Occurs when multiple threads simultaneously compete for the same serially reusable resource, and that resource is allocated to these threads in an indeterminate order. This can cause subtle program errors when the order in which threads access a resource is important. random policy—Dynamic load balancing policy in which the system arbitrarily chooses a processor to receive a migrated process. read/write lock—Lock that allows a single writer process or multiple reading processes (i.e., processes that will not alter shared variables) to enter a critical section. receiver-initiated policy—Dynamic load balancing policy in which processors with low utilization attempt to find overloaded processors from which to receive a migrated process. relaxed consistency—Category of memory-coherence strategies that permit the system to be in an incoherent state for a few seconds after a write, but improve performance over strict consistency. release consistency—Memory-coherence strategy in which multiple accesses to shared memory are considered a single access; these accesses begin with an acquire and end Key Terms 737 with a release, after which coherence is enforced throughout the system. release operation—In several coherence strategies, this operation indicates that a process is done accessing shared memory. residual dependency—Dependency of a migrated process on its original node after process migration because some of the process's state remains on the original node. round-robin job (RRJob) scheduling—Job-aware process scheduling algorithm employing a global run queue in which jobs are dispatched to processors in a round-robin fashion. round-robin process (RRprocess) scheduling—Job-blind multiprocessor scheduling algorithm that places each process in a global run queue and schedules these process in a round-robin manner. sender-initiated policy—Dynamic load balancing policy in which overloaded processors attempt to find underloaded processors to which to migrate a process. separate kernels multiprocessor organization—Scheme for delegating operating system responsibilities in which each processor executes its own operating system, but the processors share some global system information. sequential consistency—Category of memory-coherence strategies in which coherence protocols are enforced immediately after a write to a shared memory location. shared bus—Multiprocessor interconnection scheme that uses a single communication path to connect all processors and memory modules. shared virtual memory (SVM)—An extension of virtual memory concepts to multiprocessor systems; SVM presents the illusion of shared physical memory between processors and ensures coherence for pages accessed by separate processors. shortest-process-first (SPF) scheduling (multiprocessor)—Jobblind multiprocessor scheduling algorithm, employing a global run queue, that selects the process with the smallest processor time requirement to execute on an available processor. single-instruction-stream, multiple-data-stream (SIMD) computer—Computer architecture consisting of one or more processing elements that execute instructions from a single instruction stream that act on multiple data items. single-instruction-stream, single-data-stream (SISD) computer—Computer architecture in which one processor fetches instructions from a single instruction stream and manipulates a single data stream; this architecture describes traditional uniprocessors. sleep/wakeup lock—Mutual exclusion lock in which waiting processes block, and a releasing process wakes the highest-priority waiting process and gives it the lock. smallest-number-of-processes-first (SNPF) scheduling—Jobaware process scheduling algorithm, employing a global job-priority queue, where job priority is inversely proportional to the number of processes in a job. soft affinity—Type of processor affinity in which the scheduling algorithm tries, but does not guarantee, to schedule a process only on a single node throughout its life cycle. space-partitioning scheduling—Multiprocessor scheduling strategy that attempts to maximize processor affinity by scheduling collaborative processes on a single processor (or single set of processors); the underlying assumption is that these processes will access the same shared data. spin lock—Mutual exclusion lock in which a waiting process busy waits for the lock; this reduces the process's reaction time when the protected resource's becomes available. static load balancing—Category of load balancing algorithms that assign a fixed number of processors to a job when it is first scheduled. stream—Sequence of objects fed to the processor. switch—Node that routes messages between component nodes. symmetric multiprocessor (SMP)—Multiprocessor system in which processors share all resources equally, including memory, I/O devices and processes. symmetrical multiprocessor organization—Scheme for delegating operating system responsibilities in which each processor can execute the single operating system. symmetric policy—Dynamic load balancing policy that combines the sender-initiated policy and the receiver-initiated policy to provide maximum versatility to adapt to environmental conditions. thundering herd—Phenomenon that occurs when many processes awaken when a resource becomes available; only one process acquires the resource, and the others test the lock's availability and reblock, wasting processor cycles. tightly coupled system—System in which processors share most resources; these systems provide higher performance but are less fault tolerant and flexible than loosely coupled systems. 738 Multiprocessor Management timesharing scheduling—Multiprocessor scheduling technique that attempts to maximize parallelism by scheduling collaborative processes concurrently on different processors. of main memory; in general, memory-access time is constant, regardless of which processor requests data, except when the data is stored in a processor's cache. undivided coscheduling algorithm—Job-aware process scheduling algorithm in which processes of the same job are placed in adjacent locations in the global run queue, and processes are scheduled round-robin. vector processor—Type of SIMD computer containing one processing unit that executes instructions that operate on multiple data items. uniform-memory-access (UMA) multiprocessor—Multiprocessor architecture that requires all processors to share all write broadcast—Technique for maintaining memory coherence in which the processor that performs a write broadcasts the write throughout the system. Exercises 15.1 A multiprocessor's interconnection scheme affects the system's performance, cost and reliability. a. Of the schemes presented in this chapter, which are best suited for small systems and which for largescale systems? b. Why are some interconnection schemes good for small networks and others for large networks, but none are optimal for all networks? 15.2 For each of the following multiprocessor organizations, describe an environment in which the organization is useful; also, list a drawback for each. a. Master/slave multiprocessor organization b. Separate-kernels multiprocessor organization c. Symmetrical organization 15.3 For each of the following environments, suggest whether the UMA, NUMA or NORMA memory-access architecture would be best and explain why. a. An environment consisting of a few interactive processes that communicate using shared memory b. Thousands of workstations performing a common task c. Large-scale multiprocessor containing 64 processors in a single machine d. Dual-processor personal computer 15.4 As we described in the chapter, one important, but difficult, goal of a NUMA multiprocessor designer is maximizing the number of page accesses that can be serviced by local memory. Describe the three strategies, COMA, page migration and page replication, and discuss the advantages and disadvantages of each. a. COMA multiprocessor. b. Page migration. c. Page replication. 15.5 For each of the following multiprocessor scheduling algorithms, use the classifications discussed early in this chapter to describe the type of multiprocessor system that would likely employ it. Justify your answers. a. Job-blind multiprocessor scheduling b. Job-aware multiprocessor scheduling c. Scheduling using per-processor or per-node run queues 15.6 For each of the following system attributes, describe how process migration can increase it in a system and an inefficient implementation reduce it. a. Performance b. Fault tolerance c. Scalability 15.7 In our load balancing discussion, we described how processors decide when and with whom to migrate a process. However, we did not describe how processors decide which process to migrate. Suggest some factors that could help determine which process to migrate. [Hint: Consider the benefits of process migration other than load balancing.] 15.8 Can a process waiting for a spin lock be indefinitely postponed even if all processes guarantee to leave their critical sections after a finite amount of time? A process waiting for a sleep/wakeup lock? If you answered yes, suggest a way to prevent indefinite postponement. Suggested Projects 15.9 Prepare a research paper describing how the Linux operating system supports CC-NUMA multiprocessors. Describe Linux's scheduling algorithm, how Linux maintains cache and memory coherency and the various mutual exclusion mechanisms provided by the operating system. Make sure your research entails reading some source code. Recommended Reading 739 15.10 The cost and performance of different hardware devices are changing at different rates. For example, hardware continues to get cheaper, and processor speed is increasing faster than bus speed. Write a research paper describing the trends in interconnection schemes. Which schemes are becoming more popular? Which less popular? Why? algorithm? For what type of environment is each intended? Are most algorithms static or dynamic? 15.12 Prepare a research paper describing how operating systems implement memory coherence. Include a precise description of the coherence protocol and when it is applied for at least one real operating system. 15.11 Prepare a research paper surveying the load balancing algorithms in use today. What are the motivations behind each Suggested Simulations 15.13 Use Java threads to simulate a multiprocessor, with each thread representing one processor. Be sure to synchronize access to global (i.e., shared) variables. Implement shared memory as an array with memory addresses as subscripts. Define an object class called Process. Randomly create Process objects and implement a scheduling algorithm for them. A Process object should describe how long it runs before blocking each time, and when it terminates. It should also spec- ify when it requires access to a shared memory location and to which location. 15.14 Expand your simulation. Implement per-processor run queues and a load balancing algorithm. Maintain a data structure for a processor's local memory, and implement a function in the Process object to randomly add memory locations to the local memory. Be sure to migrate this local memory when migrating a Process. Recommended Reading Flynn and Rudd provide a synopsis of parallel and sequential architecture classifications in their 1996 paper "Parallel Architectures."200 Crawley, in his paper "An Analysis of MIMD Processor Interconnection Networks for Nanoelectronic Systems," surveys processor interconnection schemes.201 As multiprocessing systems have become more mainstream, much research has been devoted to optimizing their performance. Mukherjee et al. present a survey of multiprocessor operating system concepts and systems in "A Survey of Multiprocessor Operating Systems."202 More recently Tripathi and Karnik, in their paper "Trends in Multiprocessor and Distributed Operating System Designs," summarize many important topics such as scheduling, memory access and locks.203 For more information regarding scheduling algorithms in multiprogrammed environments, see Leutenegger and Ver- non's "The Performance of Multiprogrammed Multiprocessor Scheduling Policies,"204 and more recently, Bunt et al's "Scheduling in Multiprogrammed Parallel Systems."205 Lai and Hudak discuss "Memory Coherence in Shared Virtual Memory Systems."206 Iftode and Singh survey different implementation strategies for maintaining memory coherence in "Shared Virtual Memory: Progress and Challenges."207 A comprehensive survey, "Process Migration," has been published by Milojicic et al.,208 and the corresponding problem of load balancing has been treated thoroughly by Langendorfer and Petri in "Load Balancing and Fault Tolerance in Workstation Clusters: Migrating Groups of Processes."209 The bibliography for this chapter is located on our Web site at www.deitel.com/ books/os3e/Bibliography.pdf. Works Cited 1. Mukherjee, B.; K. Schwan; and P. Gopinath, "A Survey of Multiprocessor Operating Systems," GIT-CC-92/05, November 5, 1993, p. 1. 2. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30,1990, . 3. Breckenridge, C, "Tribute to Seymour Cray," November 19, 1996,. 4. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30, 1990, . 5. Breckenridge, C, "Tribute to Seymour Cray," November 19, 1996, . 6. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30,1990, . 740 Multiprocessor Management 7. Breckenridge, C, "Tribute to Seymour Cray," November 19, 1996, . 28. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30,1990, . 8. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30,1990, . 29. Allison, D., "Interview with Seymour Cray," May 9,1995, . 9. Top500, "TOP500 Supercomputer Sites: TOP500 List 06-2003," June 2003, . 30. Breckenridge, C, "Tribute to Seymour Cray," November 19. 1996,. 1o. van der Steen, A., "TOP500 Supercomputer Sites: Intel Itanium 2," . 31. Mitchell, R., "The Genius: Meet Seymour Cray, Father of the Supercomputer," BusinessWeek, April 30,1990, . I I . Habata, S.; M. Yokokawa; and S. Kitawaki, "The Earth Simulator System," NEC Research and Development, January 2003, . 12. JAMSTEC, "Earth Simulator Hardware," stec.go.jp/esc/eng/ES/hardware.html>. . . . . 53. Doughs, E; J. K. Ousterhout; M. F Kaashoek; and A. S. Tanenbaum, "A Comparison of Two Distributed Systems: Amoeba and Sprite," Computing Systems, Vol. 4, No. 3, December 1991, pp. 12-14. . 54. Doughs, E; J. K. Ousterhout; M. F Kaashoek; and A. S. Tanen baum, "A Comparison of Two Distributed Systems: Amoeba and Sprite," Computing Systems, Vol. 4, No. 3, December 1991, pp. 12-14, . 55. "The Common Object Request Broker: Architecture and Specifi cation," OMG Document Number 93.12.43, December 1993. 56. Agrawala, A. K., and G. Ricart, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Communications of theACM,Vol. 24, No. 1, January 1981. 57. Rosenkrantz, D. J.; R. E. Sterns; and P. M. Lewis, "System Level Concurrency Control for Distributed Database Systems," ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978. 58. Johnston, B.; R Javagal; A. Datta; and S. Ghosh, "A Distributed Algorithm for Resource Deadlock Detection," Proceedings of the Tenth Annual Phoenix Conference on Computers and Com munication March, 1991, pp. 253-255. 59. Obermarck, R., "Distributed Deadlock Detection Algorithm." ACM Transactions on Database Systems, Vol. 7, No. 2, June 1982, p. 198. The humblest is the peer of the most powerful. —John Marshall Harlan O! call back yesterday, bid time reeturn. William Shakespeare— Chapter 18 Distributed Systems and Web Services Objectives After reading this chapter, you should understand: • characteristics and examples of networked and distributed file systems. • types, benefits and examples of clustering. • the peer-to-peer distributed computing model. • grid computing. • Java distributed computing technologies. • Web services and the Microsoft .NET and Sun ONE platforms. Chapter Outline 18.1 lntroduction 18.5.2 Centralized vs. Decentralized P2P Applications 18.5.3 Peer Discovery and Searching 18.5.4 18.2 Distributed File Systems 18.2.1 Distributed File System Concepts JXTA 18.6 18.2.2 Network File System (NFS) 18.3.3 Andrew File System (AFS) Grid Computing 18.4.4 Coda File System 18.2.5 Sprite File System 18.3 Mukticomputer Systems 18.4 Clustering 18.4.1 Clustering Types 18.4.2 Clustering Benefits 18.7 Java Distributed Computing 18.7.1 Java Servlets and JavaServer Pages (JSP) 18.7.2 Jini 18.7.3 JavaSpaces 18.7.4 Java Management Extensions (JMX) 18.4.3 Clustering Examples 18.5 Peer-to-Peer Distributed Computing 18.5.1 Client/Server and Peer-to-Peer Applications 18.8 Web Services 18.8.1 Microsoft's .NET Platform 18.8.2 Sun Microsystems and the Sun ONE Platform Web Resources | Summary | Key Terms | Exercises | Recommended Reading | Works Cited 816 18.1 Introduction 817 18.1 Introduction In the preceding chapter, we discussed the basic concepts of distributed systems, including the middleware technologies that help to build transparent and scalable systems. We also discussed how to ensure mutual exclusion in distributed systems and how to prevent and detect distributed deadlocks. In this chapter, we look at another fundamental element of distributed systems—file sharing. Accessing files stored on file servers using a distributed file system is similar to accessing files stored on the user's local computer. We discuss several distributed file systems and give an overview of how data and files are shared among distributed computers. Then, we introduce clustering, which takes advantage of distributed systems and parallel systems to build powerful computers. We discuss the peer-to-peer distributed computing model which is used to remove many central points of failure in applications like instant messengers. Another distributed computing model we discuss is grid computing, which uses unused computer power to solve complex problems. We continue our presentation of Java distributed systems technologies, with discussions of servlets, JSP, Jini, JavaSpaces and JMX, and discuss Web services and how they improve interoperability in distributed systems. 18.2 Distributed File Systems Networked file systems allow clients to access resources stored on remote computers as the combination of a server name and the path to the resource within that server. Distributed file systems, discussed in detail in this chapter, are special examples of networked file systems that allow transparent access to remote files.1 They take the form of one global file system to which all machines in the system have access. This is achieved by allowing users to mount other file systems (see Chapter 13). Once a remote file system has been mounted, the user can access the files in it as if the files were local. This section addresses the key concepts of distributed file systems and presents case studies on various key distributed file systems. 18.2.1 Distributed File System Concepts A distributed file server can be either stateful or stateless. A stateful server maintains state information about client requests—such as the file name, a pointer to the file and the current position in the file—so that subsequent access to the file is faster and easier. In a stateless system, the client must specify this state information in each request. There are benefits to both types of systems: fault tolerance is easier to implement in stateless systems, but stateful systems can reduce the size of request messages and hence offer better performance. Distributed file systems can be characterized by transparency, scalability, security, fault tolerance, and consistency. Transparency Distributed file systems provide complete file location transparency; the user sees the distributed file system as a single, global file system. For example, consider a 818 Distributed Systems and Web Services company with offices located in Boston and New York. If a user is in Boston, the system should allow the user to access files located in either Boston or New York. However, the physical location of the files is hidden from the user. To achieve high degrees of transparency, robust naming mechanisms are required.2 Scalability Distributed file systems should also be scalable; allowing new computers to be added to the system easily. Designers of scalable distributed file systems must consider the size of the file system, file management, data duplication and concurrent user access. Other design issues, such as narrow bandwidth, could also limit scalability. Further, if any operations are restricted to a single machine, adding clients could make that machine a bottleneck.3 Security Two key security concerns in distributed file systems are ensuring secure communications and guaranteeing correct file-access permissions. Providing secure communications can be done by encrypting file content. Ensuring correct file access permissions can be complex. In many operating systems, the kernel can determine what files a process has access to based on the identity of the process' user. However, in a distributed system, users may have different user identifications on different machines. This can allow users to access files that they should not and prevent users from accessing files that they are authorized to access.4 Fault Tolerance Fault tolerance is more complex in distributed systems than in conventional systems. When a machine crashes, the system should provide mechanisms to leave the file system in a stable and consistent state. For example, if a client requests a file and blocks while waiting for a server's response and the server then crashes, the client must have some way to guarantee that it will not wait

indefinitely. Often, both the client and server will set a timer and abort a call after a timeout, but this introduces complications. What happens if a client sends a request and does not receive a response from the server? It could be that the server has crashed. It could also be that the server is slow or that its response was lost. In the latter case, if the client's request was idempotent, i.e., several calls to perform the same operation return the same result (such as when checking a bank balance), then the client can resend its request without introducing inconsistency into the system. However, if the client's request was not idempotent, such as when withdrawing money from a bant account, inconsistencies might arise.5 A well-designed distributed system makes sure that the same remote procedure calls are executed exactly once regardless of server failures. High availability is closely related to fault tolerance. In a fault-tolerant system, the failure of a component causes no down time; a backup component immedi- 18.2 Distributed File Systems 819 ately takes the place of the failed component and users will not be aware that a failure occurred. In a high-availability system, the failure of a component causes minimal down time. In such systems, often the backup components are used during normal execution for better performance.6 Consistency Consistency is another concern for the designers of distributed systems. Many distributed systems implement client caching to avoid the overhead of multiple RPCs. In such cases, clients will keep a local copy of a file and flush it to the server from :ime to time. Because there are multiple copies of the same file, files can become inconsistent. To avoid inconsistencies, each client must invalidate its copy of the file against the copy on the server. The client checks the timestamp of the copy of the file on the server, ff it has been updated since the client last copied the file, then the client must download the latest version. If the file has not been updated since the client copied it, then the client can work on its cached copy. Invalidation guarantees that each client will have the same view of the file, but it can lead to inefficiency if it results in the overwriting of work performed by the client. To reduce file access times, sometimes clients will also cache information about files, such as creation or modification dates and location.7 Servers can guarantee consistency by issuing locks to clients, giving them exclusive access to a file. When another client needs to access the file, that client must wait for the client that owns the lock to release it. The server still has ultimate control over the file and can revoke the lock if necessary. In fact, most servers lease locks and automatically revoke them when the leases expire. If a user still requires a lock, the user must renew the lease before it expires. In some cases, a server may even delegate complete control over the file to the client for a brief period of time. In this situation, other clients must request file access from that client instead of from the server, thereby reducing the load on the server.8 Designing a distributed file system involves trade-offs. One means of implementing fault tolerance is replication. But replication requires more components and introduces the problem of consistency. To ensure consistency, systems can employ client invalidation and file locks. But these increase complexity and, in some cases, restrict scalability; and these mechanisms can reduce performance. Self Review 1. Why is fault tolerance difficult to implement in stateful systems? 2. What problem is raised by client caching? Ans: 1) In stateful systems, if the server crashes, it must reconstruct the state information of each client when it restarts. The state information after the server restarts must be consistent with the state information before the crash. 2) Client caching may cause file inconsistency because multiple copies of a file may be distributed to multiple clients, enabling them to modify the same file at the same time. 820 Distributed Systems and Web Services 18.2.2 Network File System (NFS) Sun Microsystem's Network File System (NFS), the current de facto standard for network file sharing, is supported natively in most varieties of UNIX and with client and server software available for other common platforms. NFS was originally designed around TCP/IP, so it has been easily adapted to link file systems across the Internet and other wide area networks.9 The original version of NFS was released by Sun in 1984 and licensed free to the industry. NFS was revised in 1985 to version 2, which became integrated into various operating systems. In 1989 NFS-2 became the first version of NFS to be standardized by the Internet Engineering Task Force (IETF), becoming an Internet protocol.10 NFS was revised again with the release of version 3 in 1995.11 Figure 18.1 illustrates how NFS works. Before an NFS client can access a remote file, it must use the mount protocol—a mechanism that translates a file path name to a file handle—to mount the directory that holds the remote file. A file handle identifies a remote file by its type, location and access permissions. Mounting involves mapping a remote file directory to a client's local directory. To mount the file system, the NFS client makes an RPC (the mount request, which contains the file path of the remote file to be mounted) to the client stub. Upon receiving the mount request, the server skeleton passes the call to the NFS server. The NFS server looks up its local file system and exports the local file directory that satisfies the client request, which means the server makes the local directory of files available to the remote client. The server skeleton then returns a file handle of the exported local file directory, which allows the client to access the exported file Figure 18.1 I NFS architecture. 18.2 Distributed File Systems 821 directory remotely. The remote client uses this file handle in all subsequent requests to files in the exported file directory. NFS-2 and NFS-3 assume a stateless server implementation, which makes fault tolerance easier to implement. If a stateless server crashes, the client can simply retry until the server responds. The NFS-2 and NFS-3 permission checks simply rely on the authentication information from RPC, which determines whether the client can access certain information and functions. NFS-2 uses UDP as its transfer protocol, since UDP performs well over local area networks. TCP becomes the protocol of choice on error-prone, high-latency and wide-area networks. Both NFS-2 and NFS-3 support TCP; NFS-3 optimizes its use by allowing transfers of large streams of data. Neither version 2 nor version 3 defines a caching policy to guarantee the consistency of the server and client copies of the file.12 Although NFS-2 and NFS-3 are structurally similar, NFS-3 introduces several key differences. First, it supports larger files—file sizes can be stored in 64 bits, while NFS-2 supports only 32-bit file sizes. Second, NFS-3 supports safe asynchronous writes, which allow a server to continue executing before a write has been completed. Finally, NFS-3 removes the 8KB limit defined by NFS-2 for any single request or reply.13 This limit was chosen as a reasonable upper bound for the size of UDP packets being transferred over local area Ethernet. Following the release of NFS-3, Sun enhanced NFS with a service called WebNFS, which allows NFS clients to access WebNFS-enabled servers with a minimum of protocol overhead. WebNFS is specifically designed to take advantage of NFS-3's use of TCP to support efficient file operations over the Internet. Many features provided by WebNFS have been integrated into version 4 of NFS, which entered development in 1999 as an IETF proposal. NFS version 4 maintains most of the features of prior versions of NFS and provides efficient operation over wide area networks, strong security implementation, stronger interoperability and client caching.14 Like its predecessors, NFS-4 uses RPCs to communicate between client and server. NFS-4 offers a new feature, allowing multiple related RPCs to be packaged together into one request. This limits network traffic and increases performance. Also, NFS-4 is stateful, which allows faster access to files. With earlier versions of NFS, the client used the mount protocol to receive the file handle. In NFS-4, the client obtains the file handle without the mount protocol. The client gets a file handle referring to the root of the file system on the server, then traverses the file system to locate the requested file. By removing the mount protocol, the client does not need to mount the different file systems of a server separately.15 NFS-4 extends the client-caching scheme through delegation, whereby the server temporarily transfers control of a file to a client. When the server grants a read delegation, no other clients can write to that particular file but they can read from it. When the server grants a write delegation, then no other clients can read from or write to that particular file. If another client requests write access to a file with the read or write delegation, or the client requests read access to the file with 822 Distributed Systems and Web Services write delegation, the server will revoke the delegation and request that the original client flush the file to disk.16 As discussed earlier, the only security provided by previous versions of NFS was weak authentication from RPC. The developers of NFS-4 were able to take advantage of new RPC security which also supports several cryptographic algorithms which are used when NFS-4 sets up communication sessions and encrypts each RPC. 17 Self Review 1. Does NFS-2 require stateful server implementation? What is the benefit of such a design? 2. When an NFS-4 client owns the read delegation of a file, can other clients read or write to that file? Ans: 1) No, NFS-2 servers must be stateless. With a stateless server, it is easy to implement fault tolerance. 2) Other clients can read from, but not write to, the file. 18.2.3 Andrew File System (AFS) Development of the Andrew File System (AFS), a scalable distributed file system, began in 1983 at Carnegie Mellon University. The goal was to provide a distributed file system that would grow to support a large user community while being secure. The designers hoped to exploit advances in workstation processing capabilities to facilitate scalability, efficiency and security. AFS is a global file system that appears as a branch of a traditional UNIX file system at each workstation, ft is based on the client-server model and relies on RPCs for communication. Andrew went through three major revisions: AFS-1, AFS-2 and AFS-3. 18 - 19 The entity that governs distributed file access in AFS is called Vice (Fig. 18.2). Vice processes run on distributed file servers and interact with a user-level process called Venus at each client. Venus interacts with the kernel's virtual file system Figure 18.2 | AFS structure. 18.2 Distributed File Systems 823 (VFS), which provides the abstraction of a common file system at each client and is responsible for all distributed file operations. AFS obtains scalability from its caching method. The local nonvolatile storage (disk) at each workstation serves primarily as a cache to the distributed file system. The data stored in nonvolatile storage is not lost when a machine loses power. When a user process tries to open a file, it makes a request to the VFS which consequently checks to see if the file is in the cache. If it is, the VFS simply accesses the file. If not, the VFS communicates through Venus to Vice, which retrieves the file from the correct server.20 In AFS-1 and AFS-2, the cache was updated only at the file level—block updates were not possible. This guaranteed that users working locally on a file would not have to wait long for network access once the file was open. AFS-3 was also upgraded to support block-level caching, which allows large files to be manipulated by clients with small caches. In AFS-1, each time the VFS tried to open a cached file it would have Venus contact the server to make sure that the file was still valid. This generated many network calls, even when files had not been modified. In AFS-2 and AFS-3, the server becomes responsible for ensuring each client has the most up-to-date copy of its riles. Each server keeps track of the files that clients cached. Each time a client closes a file, Venus flushes it to Vice, so the server always keeps the most up-to-date copy of the files. When a file is flushed to Vice, the server performs a callback to clients that have that file cached. When a client receives a callback, it knows that its cached version is no longer valid and requests the newest version from the server. In AFS-1 the server had one process listening on a dedicated port for connection requests. Upon connection, this process would fork another process to handle the Vice functions for the duration of the client's session. Because processes on different machines cannot share main memory, all shared data had to be written to the server's local file system. Furthermore, servers suffered performance problems due to the time associated with the context switches that were required to service different clients. In AFS-2 and AFS-3 there is one dedicated process that services all client requests. This process spawns a separate thread for each client and allows all state information to be loaded into virtual memory. Because the state information tends to be small for each client, it can generally be held in main memory. This eliminates context switches and allows interthread communication through shared memory. The RPC mechanism is incorporated into the thread package to further improve performance. 21 The overall AFS structure also changed with each revision. In AFS-1 different subtrees were divided among different servers, which were known as the custodians. Each server also had a directory that would allow Vice to direct requests for different subtrees to different servers. Venus located resources based on pathname. This had several drawbacks—resolving an entire path could generate multiple calls to Vice and could span multiple custodians. AFS-2 introduced volumes to manage subtrees. Volumes are primarily of administrative value, allowing replication and isolation of certain subtrees, and are transparent to users. Generally each user is 824 Distributed Systems and Web Services assigned a volume. Making the idea of volumes concrete also allowed the implementation of file identifiers (fids) in Venus. A fid specifies a volume, an index within a volume and an identifier to guarantee object uniqueness within a volume. The AFS-2 implementation of Venus resolved filenames in this manner, which simplified file lookups and allowed the renaming of directories. Each server had a directory to allow it to find volumes on any other server. AFS-3 was designed to be a large distributed file system. In fact, one continuous AFS-3-based namespace spanned multiple geographically distant universities in the United States. Obviously, there cannot be centralized administration of such a file system as each university typically has its own managerial requirements. Therefore, AFS-3 introduced cells, which preserve namespace continuity while allowing different systems administrators to oversee each cell.22, 23 The security of AFS is much stronger than that of earlier versions of NFS, partially due to the underlying RPC structure. Because AFS is designed to grow to service many computers, its designers decided that it would not be safe to trust the network or any client. Therefore, every RPC is encrypted and must be authenticated. In AFS-1 and AFS-2 the permissions of a given user are related to the permissions of each group to which the user belongs. In AFS-3 permissions must be specified individually. Upon system login, users must specify a password. Vice sends the encrypted password to an authentication server, which responds with tokene. Secure RPCs use these tokens to provide secure communication. File access is governed by access control lists (ACLs). Section 19.4.3, Access Control Mechanisms, discusses ACL in detail. Each user is allowed to perform certain operations on certain files. Authentication is guaranteed by matching the credentials on each token to the permitted operations in the ACL. 24 18.2.4 Coda File System AFS is scalable, indeed able to service thousands of computers, but it has one large drawback—it is not fault tolerant. The crash of one server or disruption of network service can render hundreds of computers inoperable. To deal with this, the designers of AFS began research on Coda in 1987.25 Coda is derived from AFS-2. As in AFS-2, Coda uses local nonvolatile storage (e.g., disks) as file caches, and Vice uses callbacks to keep clients up to date. Volumes are the basic division in the file system and Coda performs file lookups in the same way as AFS-2. The goal of fault tolerance led to the addition and modification of several of AFS's features, including the idea of a volume.26 In Coda, volumes are logical pieces of the file system and are replicated physically across multiple file servers. Servers that hold the same volume are known as a volume storage group (VSG). Although the multiple servers are transparent to users, the Venus process on each client periodically probes the VSGs of each of its volumes. The members of the VSG with which the client can connect and communicate are known collectively as the client's available volume storage group (AVSG) (Fig. 18.3). Coda uses AVSGs to implement fault tolerance. 27 18.2 Figure 18.3 | Coda volume structure. When a client process tries to read a file, its request is directed to the local VFS. If the file is in cache, then the read continues as in AFS-2. Otherwise, Venus contacts a member of the AVSG known as the preferred server (PS). Venus downloads a copy of the file from the PS and downloads file version information from each member in the AVSG. If the versions on each server agree, then the client is allowed to use the file. If the versions from different servers do not agree, then the calling process aborts and the outdated copies are updated. After the members of the AVSG have been updated, the requesting client downloads a copy of the file. Note that if the PS or any member of the AVSG crashes, the client can still operate, whereas in AFS-2 the client would be stuck.28 Upon closing a file, Coda clients write a copy of the file to each member of the AVSG The write occurs in two steps using the Coda Optimistic Protocol. The first step, COP1, writes the file as we have discussed. The second step, COP2 writes an update set, which specifies the members of the AVSG that have successfully performed the COP1, to each of the members of the AVSG. If any of the members are not in the update set, it is assumed that they have not updated (even if they have and their reply was lost) and this is noted to remind the system to update them the next time communication is established. To decrease write delays, COP2 messages are performed asynchronously and are appended on future COPf messages. If other clients have a copy of the file in cache, then servers will have to perform callbacks.29 These operations will provide a consistent view of a file within an AVSG. When a file is opened, it will be the same as the last copy that was written to Coda. However, Coda's replication scheme is optimistic, which means it assumes that files in different AVSGs are consistent and it allows clients to access and modify such files. This guarantees that files will always be available, but it might lead to inconsistencies and the loss of work. Distributed File Systems 825 826 Distributed Systems and Web Services We now discuss Coda's replication scheme and potential consistency issues. Consider a VSG and two clients whose AVSGs are mutually exclusive (Fig. 18.4). It is possible for each client to download, modify and upload the same copy of a file to its AVSG. Now the view of that file within the VSG will not be consistent. Coda provides some mechanisms to fix inconsistencies. If the file in conflict is a directory, the Coda servers deal with the problem. Each member of the VSG locks the volume that holds the document and one is designated the leader. Logs of the transactions of each server are sent to the leader, which combines the logs and redistributes them. Based on the logs, each server updates its view of the directory. Finally, the volumes are unlocked. When Coda clients are connected to Coda, they enter a stage called the hoarding stage, where they prepare for a possible disconnection from the system by caching any requested files. When the client becomes disconnected, it enters the emulation stage, during which all file read requests are serviced from the cache. The files could be stale, but the system is still operable. If the user tries to access a file that is not in cache, the system reports an error. Writes during the emulation stage occur in two steps. First, the file is updated on disk. Second, the client modifkation log (CML) is updated to reflect file changes. Venus modifies the CML so that when a log entry is undone by a later entry, the earlier log entry is removed. The CML also facilitates updates during the reintegration stage. Reintegration occurs immediately after the client reconnects to the system when Venus asynchronously updates the server using the CML. 30 A client may become disconnected for several reasons: All of the servers in a VSG could crash, a network connection could be faulty or the user could intentionally disconnect from the network. Coda handles each of these cases in the same way, and the transition between connected and disconnected states is transparent. Figure 18.4 | File consistency issues in mutually exclusive AVSGs. 18.2 Distributed This makes Coda particularly attractive for the growing number of mobile computers, and it is one of the primary reasons that Coda is still undergoing improvement. Self Review 1. In Coda, if client A and client B belong to the same AVSG, does it imply that they belong to the same VSG? If client A and client B belong to the same VSG, does it imply that they belong to the same AVSG? 2. What is the problem with Coda's replication scheme? Ans: 1) If client A and client B belong to the same AVSG, then they belong to the same VSG. If client A and client B belong to the same VSG, then they may not belong to the same AVSG. 2) Coda's replication scheme provides a consistent view of a file within an AVSG but does not guarantee consistency within the VSG. Coda's replication scheme assumes that files in different AVSGs are consistent and allows clients to access and modify such files within different AVSGs, which causes inconsistency within the VSG. 18.2.5 Sprite File System In Chapter 17, we overviewed the Sprite operating system. This section focus on the Sprite file system. A primary goal for the Sprite file system was transparency. Sprite's developers attained a better level of transparency than that offered by NFS, AFS or Coda. A second goal was to implement complete consistency.31 The Sprite file system has the perceived form of a UNIX file system. Unlike NTS or Coda, however, it gives every client the exact same view of the hierarchy. The Sprite file system also goes one step further in UNIX file system emulation by allowing transparent remote access to I/O devices (which are represented as files in UNIX). Although hidden from the user, the Sprite file system is divided into domains. Each domain represents a piece of the global file hierarchy and is stored at one server (Fig. 18.5). Except for the domain at the top of the global file hierarchy, the root of one domain is the deepest child of another. 32 Figure 18.5 | Sprite file system domains. File Systems 827 828 Distributed Systems and Web Services The kernel of each client keeps a private copy of a prefix table, which stores domain information to aid in file lookups (Fig. 18.5). Each entry in the table represents a separate domain and consists of the absolute path to the root directory within the domain, the server that houses the domain and a token that identifies the domain. To look up a file, a client takes an absolute pathname and locates the longest matching prefix within the prefix table. If no such prefix can be found, the client broadcasts a request to every server. The server that houses the domain that matches the path responds with information to populate the client's prefix table, and the client continues. The client then removes the matched prefix from the name and sends what is left to the indicated server along with the domain identification token. The server resolves the name within the specified domain and returns a token pointing to the requested file. If the resolution cannot be completed because the path spans multiple domains, the server returns the pathname of the new domain and the client repeats the process. Sprite processes that access files keep the server address and token identifying the domain of the working directory as part of their state. Processes that access files via relative paths provide this state information in addition to the relative path to allow servers to resolve the entire path. 33 In Sprite, both the client and the server keep a cache in main memory. To open a file, the client first checks its cache. If the file is not in the client's cache, the client then makes a request to its backing store (which is generally a server). If the server is unable to satisfy the request from its cache, it reads the data in from disk (Fig. 18.6, step a). The server then stores the data in its cache and passes the data t< the client. The client retains a copy of the data in its cache as well (Fig. 18.6, step b). When a client writes a block, it writes into its cache. Sprite flushes updated pages to the server every 30 seconds though the cache management algorithm may choose to flush the page sooner. The page will then remain in the server's cache for 30 seconds before it is flushed to disk (Fig. 18.6, step c). The lazy write-back strategy was chosen to trade an increase in performance for a decrease in fault tolerance. As in UNIX systems, modified pages in a cache will be lost during a crash if they have not been flushed to the server.34 Sprite's complete consistency is achieved by its caching protocols. The clients may have inconsistent versions of a file if two or more clients have a copy of the file in cache and at least one client is modifying its copy. This can be divided into two cases: sequential write-sharing and concurrent write-sharing. Sequential write-sharing occurs when one client modifies a file then another client tries to read or write its own cached copy of the file. To guarantee the highest level of consistency, before a client opens a file it contacts the server and requests the version of that file. If the file on the server is newer than the cached version, the client removes the file from cache and downloads the version from the server. To ensure that delayed writebacks do not hinder this algorithm, the server keeps track of the last client that writes a file. On receiving a read request, the server forces the last client that modifies the file to flush all modified pages to disk immediately, before servicing the request. Concurrent write-sharing occurs when two clients modify cached copies of 18.2 Distributed File Systems 829 Figure 18.6 | Reading and writing in the Sprite file system. the same file at the same time. To ensure consistency, Sprite disables caching of that file for all clients. Reads and writes are sent directly to the server, and the state of the file therefore appears the same to all clients. Note that although each client will have a consistent view of the file, there could be synchronization issues associated with this algorithm. Sprite provides locking system calls for applications that cause the inconsistency. This solution also hinders file system performance. The developers of Sprite justified their implementation by observing that concurrent write-sharing is rare.35 Cache (in main memory) and virtual memory contend for the same system resources in Sprite—physical memory. Whereas devoting a large portion of physical memory to the virtual memory system increases processing efficiency, devoting a large portion to cache increases network and file system efficiency. Furthermore, some applications rely heavily on virtual memory while others rely on the file system. Sprite allows its cache size to vary dynamically to deal with these issues. Physical memory pages are divided into two sets, a set for cache and a set for virtual memory. The replacement algorithm for each set uses the age of the pages as a guideline for removal when new pages must be swapped in. The set that has the older page loses that page to the set that is swapping in a new page. For example, consider the situation where 16 pages of physical memory are devoted to cache, 16 pages are devoted to virtual memory and one of the virtual memory's pages is the 830 Distributed Systems and Web Services oldest of all 32 pages. If a process tries to bring a page into cache, the oldest virtual memory page is removed and the cache is then allocated 17 total pages. This introduces a subtle point of inefficiency. What happens if a page is in both virtual memory and cache? Clearly the system is wasting space. To avoid this, virtual memory writes directly to the server when updating files. We have seen that Sprite is more concerned with performance than with fault tolerance. This point is strengthened when considering how Sprite deals with server crashes. Sprite does not use replication servers. Instead, in the hope that crashes would be infrequent, Sprite's designers implemented a server that would recover quickly from crashes. To do this they placed the burden of state control on the clients, using a log-structured file system with checkpointing on the server (as discussed in Section 13.10.2). The motivation for log-based file system was to improve I/O performance. When clients realize that a server has crashed, they wait until it reboots and then send it information about its files that they have cached. The server uses this information to rebuild its state.36 Self Review 1. How is complete consistency achieved in sequential write-sharing? 2. What problem might result from placing the burden of state control on the clients? Propose a solution to this problem. Ans: 1) Sequential write-sharing occurs when one client modifies a file and then another chent tries to read or write its own cached copy of the file. Before a client opens a file, it contacts the server and requests the version of that file. If the file on the server is newer than the cached version, the client removes the file from cache and downloads the version from the server. To ensure complete consistency, the server keeps track of the last client that writes a file. On receiving a read request, the server forces the last client that modifies the file to flush all modified pages to disk immediately, before servicing the request. 2) All of the clients will bombard the server with information when it reboots, possibly swamping the network. Sprite's developers built a negative acknowledgement into their implementation.This allows the server to tell the client that it is busy now, but has not crashed and to come back later. 18.3 Multicomputer Systems As discussed in Chapter 15, multiprocessing is the use of multiple processors to execute separate portions of a computation in parallel. Several classes of machines are generally referred to as multiprocessors, but the most popular are machines that have similar or identical processors and that share a common memory. In contrast, multicomputer systems do not share a common memory or bus (although they may share a virtual memory). Each processing unit has access to its own resources. These independent units are connected in a network to operate cooperatively to form a multicomputer system. Multicomputer systems can be either homogenous or heterogeneous. A homogeneous multicomputer system consists of computers that have the same hardware, run the same operating systems and communicate across the same network architectures. A heterogeneous multicomputer system consists of 18.4 computers that may have different hardware, run different operating systems and communicate across different network architectures.37 Modern supercomputers are massively parallel processing (MPP) machines capable of hundreds of gigaflops or even teraflops. A flop is a floating point operation per second.38 Currently under construction at Los Alamos is the supercomputer Q which will top 100 teraflops. It will be used by scientists to simulate the effects of aging on America's nuclear arsenal, using three-dimensional models of explosions.39 Los Alamos scientists are also working on a low-end supercomputer called Green Destiny built from off-the-shelf components. Green Destiny provides 160 gigaflops of computing power using 100Base-T Ethernet and 240 667-megahertz Transmeta Corp. processors.40 Green Destiny is a multicomputer system. Another method of implementing a multicomputer system is with grid computing which uses middleware that can run like an application on a general-purpose computer. Distributed.net pioneered academic distributed computing over the Internet by having users download an application which performed operations on their own computers. 41 Today there is also , which uses volunteers' computers to process data collected by the Search for Extraterrestrial Intelligence project. Intel, United Devices, and the American Cancer Society have created a screensaver software that performs calculations to aid in pharmaceutical research. 42 Companies such as United Devices are developing software and hardware packages to create enterprise computer grids which will allow organizations to leverage unused processor cycles on their networks to handle computer-intensive tasks. We discuss such grid computing in Section 18.6, Grid Computing. Distributed processing makes large computations feasible. However, this power comes at the cost of simplicity. Just as in uniprocessing environments, distributed processing must take into account synchronization, mutual exclusion and deadlock. Networking and the lack of shared memory make managing such issues more complex. Self Review 1. Distinguish multiprocessing and multicomputer systems. 2. What makes distributed processing difficult to implement? Ans: 1) With multiprocessing, processors share a common memory. Multicomputer systems do not share a common memory—each processing unit has access to its own resources. 2) Distributed processing must take into account synchronization, mutual exclusion and deadlock. Networking and the lack of shared memory make managing such issues more complex. 18.4 Clustering Recall that in a distributed system, several computers handle user requests simultaneously. Each computer processes user requests independently. However, the distributed system must handle communication delays introduced by its underlying networks. Multiprocessing fixes the communication delays by allocating many pro- Clustering 831 832 Distributed Systems and Web Services cessors for one computer, making the communications among processes easier and faster. Yet, multiprocessing has its own disadvantages: high cost/performance ratio, limited scalability and not all types of applications running on a system benefit from multiprocessing.43 Clustering—interconnecting nodes (single-processor computers or multiprocessor computers) within a high-speed LAN to function as a single parallel computer— is architecturally intermediate between distributed computing and multiprocessing. The set of nodes that forms the single parallel machine is called a cluster. Clustering enables multiple computers to work together to solve large and complex problems.44 Self Review 1. What is clustering? 2. Why would clustering be employed? Ans: 1) Clustering interconnects nodes within a high-speed LAN to function as a single parallel computer. 2) Clustering takes advantage of multiprocessing and distributed systems while eliminating the communication delays associated with fully distributed systems, providing a powerful computer that is capable of solving large and complex problems. 18.4.1 Clustering Types There are three major types of clusters. In a high-performance cluster, all of the nodes in the cluster are used to perform work. In a high-availability cluster, some of the nodes perform work while others serve as backups. If working nodes or their components fail, the backup nodes start running and take over the jobs that were being executed on the failed nodes immediately without interrupting service. In a load-balancing cluster, a particular node works as a load balancer to distribute the load (such as thousands of requests from the clients) to a set of nodes so that all hardware is efficiently utilized.45 High-performance clusters are used for solving large-scale computations that can be divided into smaller problems to be solved in parallel. High-availability clusters are used for mission-critical applications in which failure must be avoided. Load-balancing clusters benefit organizations that handle large volumes of user requests. For example, high-performance clusters are useful for analyzing images from Magnetic Resource Imaging (MRI) scanners, while high-availability clusters are crucial in aircraft control systems and load-balancing clusters are usually used in e-business applications with high user volumes. Self Review 1. Does a high-availability cluster require that its nodes be working all the time? 2. Argue why a cluster for a particular type of application should have the attributes of all three types of clusters. Consider a search engine being used for an application where lives could be at stake, such as locating organs for transplant. Ans: 1) No. High-availability clusters require that only some of their nodes be working while others act as backups. 2) The cluster should be high performance, because billions of 18.4 Clustering 833 Web pages would have to be searched in a fraction of a second. It should be high availability to ensure that it is always ready to operate when needed, especially in an emergency. It should be load balancing to spread the load in peak periods, ensuring that individual requests are not delayed. 18.4.2 Clustering Benefits There are several benefits associated with clustering. It economically interconnects relatively inexpensive components. This reduces the cost for building a clustered system compared to a single parallel computer with the same capability.46 High performance is another advantage of clustering. Each node in the clustering system shares the workload. Communications among nodes in the cluster are faster than those in unclustered distributed computing systems due to the high-speed LAN between nodes.47 Clustering can also provide replication of resources across the nodes, so that the failure of any one computer will not affect the availability of the system's resources. If a service on one node is down, other nodes that provide the same service in the cluster can still function and take over the tasks performed by the failed component.48 Clustering inherits the scalability of distributed systems. A cluster is able to add or remove nodes (or the components of nodes) to adjust its capabilities without affecting the existing nodes in the cluster. Clustering provides better scalability than multiprocessing. For example, the HP 9000 Superdome can have up to 64 processors, while the HP XC clusters can scale up to 512 processors.49'50 Clustering achieves reliability and fault tolerance by providing backups for the services and resources. When failure occurs in one running service, the cluster immediately switches the process to the backup service without affecting the performance and capacity of the system, providing uninterrupted service.51 Self Review 1. Why is communication among nodes in a cluster faster than among nodes in an unclustered distributed computing system? 2. How does clustering implement fault tolerance? Ans: 1) The reason is that a high-speed local area network is generally faster than the wide area networks used in distributed systems. 2) Clustering provides backups for services and resources. 18.4.3 Clustering Examples In this section, we consider how clustering is implemented in Linux and Windows. Many Linux clustering solutions are available, the best known being Beowulf, a nigh-performance cluster.52 In 1994, the NASA Earth and Space Sciences (ESS) project built the first Beowulf cluster to provide parallel computing to address large-scale problems involved in ESS applications. Figure 18.7 shows a typical Beowulf cluster. A Beowulf cluster may contain up to several hundred nodes. Theoretically, all nodes have Linux installed as their operating system and they are interconnected with high-speed Ethernet (often with a bandwidth of 100 Mbps). Each 834 Distributed Systems and Web Services Figure 18.7 | Typical Beowulf cluster. Beowulf cluster has a head node (also called a master node) that acts as a server to distribute the work load, control access to the cluster and handle the shared resources. All other nodes in the cluster are often called slave nodes. The head node usually has a monitor and keyboard, while slave nodes do not. All slave nodes should be configured with the same processor, memory and disk space so that all nodes can complete the same job at approximately the same time. Usually, all the nodes in the cluster are connected within a single room to form a supercomputer. 53 Windows Server 2003 can be used to build both high-availability clusters and load-balancing clusters. A high-availability cluster built with Windows Server 2003 can have at most eight nodes. All its nodes may share a storage device (Fig. 18.8) or each node may have a local storage, in which case the backup node must keep the exact same data (Fig. 18.9). Unlike the Beowulf cluster, the nodes in the high-availability Windows cluster can be connected either in a LAN or a WAN. In the case of shared storage, only one node is running at a time and that node has the control of the shared storage. When the running node fails, a backup node takes over control of the shared storage. However, if the shared storage breaks down, the entire clus- Figure 18.8 | High-availability cluster with shared storage. 18.4 Figure 18.9 | High-availability cluster with local storage. ter cannot keep running. When each node has its own storage, more than half of the cluster nodes must be active at all times so that when one or more nodes fail, other nodes can continue work effectively. For example, if the cluster has eight nodes, at least five of them must be active and ready to function.54 A load-balancing cluster built with Windows Server 2003 can have at most 32 nodes (Fig. 18.10). The cluster does not require sharing storage, because each node can do its job independently. If one node fails, the remaining nodes in the cluster are still capable of handling user requests. Scalability is achieved with the load-balancing cluster, because it is easy to add or remove a node from the cluster. Usually the nodes in the load-balancing cluster are interconnected with high-speed Ethernet.55 Self Review 1. List three characteristics that make Beowulf clusters high-performance. 2. What problem does the high-availability cluster with shared storage have? Figure 18.10 | Load-balancing cluster with 32 nodes. Clustering 835 836 Distributed Systems and Web Services Ans: 1) All nodes in the Beowulf cluster are interconnected with high-speed Ethernet (often with a bandwidth of 100 Mbps). All nodes can complete the same job approximately at the same time. All nodes work together in parallel to improve performance. 2) All nodes depend on the shared storage. If the shared storage fails, the entire cluster fails. 18.5 Peer-to-Peer Distributed Computing In a peer-to-peer (P2P) application, each peer—a single computer in the P2P system—performs both client and server functions. Such applications distribute processing responsibilities and information to many computers, thus reclaiming otherwise wasted computing power and storage space and eliminating many central points of failure. In the next several subsections we introduce the fundamental concepts of peer-to-peer applications. 18.5.1 Client/Server and Peer-to-Peer Applications Many network applications operate on the principle that computers should be segregated by function. Some computers, such as servers, offer common stores of programs and data. Other computers, such as clients, access the data provided by the servers. Using a search engine from a Web browser is an example of a client/server application. Clients send queries to servers, which access various databases and respond with the requested information. P2P applications are different from client/server applications because all computers in the network act as both clients and servers. Each peer has the ability to discover and communicate with other peers and may share resources (such as large multimedia files) with others. P2P applications are similar to the telephone system—a single user can both send and receive information.56 Self Review 1. How do client/server applications work? 2. List the advantages of peer-to-peer applications over client/server applications. Ans: 1) In a client/server application, the client sends requests to the server, and the server processes the client request, often accessing one or more databases, and sends the response back to the client. 2) It is easier to set up the network for a peer-to-peer application, and no network administrator is required. In a peer-to-peer application, if one peer fails, others can still function well. In a client/sever application, if the server fails, the entire application fails. 18.5.2 Centralized vs. Decentralized P2P Applications P2P applications can be implemented in two forms: centralized and decentralized.57 A centralized P2P application uses a central server system that connects to each peer. Centralized P2P applications are similar to the client/server relationship. In a 18.5 Peer-to-Peer Distributed Computing 837 Figure 18.11 | Centralized P2P instant-messenging application. centralized instant messenging application (Fig. 18.11), when peer 1 wants to talk to peer 3, peer 1 must get the address of peer 3 from the server first. Then peer 1 communicates with peer 3 directly. One major weakness of this centralized system is the dependency on the server. If one or more central servers fail, the entire network could fail. The server's capabilities limit the application's overall performance. For instance, Web sites can fail when malicious users overload the Web server(s) with an excessive number of requests. However, centralized architectures also have advantages, such as simplifying management tasks (e.g., monitoring user access by providing single points of network control). A pure P2P application, also called a decentralized P2P application, does not have a server and therefore does not suffer from the same deficiencies as applications that depend on servers. In a pure P2P instant messenger application (Fig. 18.12), when peer 1 wants to send a message to peer 3, peer 1 no longer needs to communicate with the server. Instead, peer1 discovers peer3 via distributed search mechanisms and sends messages to peer3 directly. If a peer in a welldesigned, pure P2P system fails, all other peers continue to function, so such systems are particularly fault tolerant. Peer-to-peer applications have disadvantages as well. Anyone with the appropriate software can join the network of peers and often remain anonymous, so determining who is on the network at any moment is difficult. Also, the lack of a server hinders the protection of copyright and intellectual-property rights. Realtime searches can be slow and increase network traffic, as queries propagate throughout the network. 58 Figure 18.13 lists some common peer-to-peer applications. 838 Distributed Systems and Web Services Figure 18.12 | Pure P2P instant-messenging application. Distributed Application Description Gnutella A P2P technology used to share documents on the Internet. Gnutella does not use any servers. There is no authentication, and peers search for files via a distributed search mechanism.59 (Section 18.5.3, Peer Discovery and Searching, overviews this mechanism.) KaZaA A file sharing application that is a hybrid between Gnutella and centralized applications. A server authenticates all users. Certain peers serve as search hubs, which catalog the files of peers connected to them. Searches are distributed to each search hub, which then responds with results that allow direct connections for file transfers.60 Groove A P2P system that allows users to communicate, collaborate and share documents on the Internet and intranet. Groove provides secure communication because users are authenticated and private data is not shared with third parties.61 Freenet Decentralized P2P technology that allows users to share documents on the Internet without fear of censorship. Freenet does not have a central server to govern access. Instead, access to Freenet is anonymous. Documents stored in Freenet are encrypted to improve protection.62 Instant messenging P2P application that enables users to send short text messages and files to one another. Most instant messengers use servers that authenticate all users and route messages between peers. Figure 18.13 | Common P2P applications. Pure peer-to-peer applications are completely decentralized. Some file sharing applications are not pure peer-to-peer, because they use servers to authenticate users and index each peer's shared files. However, peers connect directly to one 18.5 Peer-to-Peer Distributed Computing 839 another to transfer files. In such a system, centralization improves search performance but makes the network dependent on a server. Performing file transfers between peers decreases the load on the server. Self Review 1. What are the advantages of centralized P2P applications? 2. What is a benefit of a decentralized P2P application over a centralized one? Ans: 1) Centralized P2P applications simplify management tasks, such as monitoring user access by providing single points of network control. 2) Centralized P2P applications rely on a server for certain operations. If the server fails, those operations will be unavailable. 18.5.3 Peer Discovery and Searching Peer discovery is the act of finding peers in a P2P application. Pure P2P applications often suffer slow peer discovery and information searching due to the lack of a server. There is no general solution to peer discovery. Gnutella presents one approach for circumventing these problems. Gnutella is a pure peer-to-peer technology that enables distributed information storage and retrieval. Users can search for and download files from any peer on the network. Users first join a Gnutella network by specifying the network address of a known Gnutella peer. Without knowing at least one peer on the network, a user cannot join the network. Each user's Gnutella software functions as a server and uses the HTTP protocol to search for and transfer files.63 To perform a search, a peer sends search criteria to the peers connected to it. Those peers then propagate the search throughout the network of peers. If a particular peer can satisfy the search, that peer passes this information back to the originator. The originator then connects directly to the target peer and downloads the information. The peer that made the original query can identify itself only when it connects directly to the peer with the requested file to begin file transfer. In the Gnutella network, a peer forwards a search request to all of its directly connected peers in parallel. For example, if P1 receives a request from the user, P1 forwards the search request to all its directly connected peers first. These peers then work in parallel; each one tries to fulfill the search request and also forwards the request to its directly connected peers. In the Freenet P2P application, each peer forwards a search request to a single directly-connected peer at a time. For example, if P1 receives a request from the user, P1 forwards the search request to P2 which is one of its directly connected peers. If P 2 cannot answer the request, it forwards the request to one of its directly connected peers. If all the nodes that P2 can reach cannot answer the request, P1 then forwards the search request to another one of its directly connected peers.64 Searches conducted in both Gnutella and Freenet are called distributed searches. Distributed searches make networks more robust by removing single points of failure, such as servers. Not only can peers find information in this way, but peers can search for other peers via distributed searches as well. 840 Distributed Systems and Web Services Self Review 1. What are some of the problems with distributed searches? 2. Compare Gnutella's distributed search mechanism to Freenets. Ans: 1) Distributed searches are costly, possibly generating enormous amounts of network traffic. If poorly implemented P2P systems proliferate, this could have serious ramifications on the performance of the entire Internet. Distributed searches do not guarantee to find the requested files. 2) Searches in Gnutella will reach close peers before reaching any more distant peers. In Freenet, a search can reach a distant peer before all close peers have been searched. 18.5.4 JXTA Sun Microsystems, Inc., created Project JXTA (short for Juxtapose) in response to the growing popularity of peer-to-peer applications. Project JXTA strives to create a standard, low-level, platform and language-independent protocol that promotes interoperability among peer-to-peer applications. The current JXTA implementation is written in Java, but developers can implement JXTA in any programming language. JXTA provides a foundation on which developers can build any type of P2P application. JXTA attempts to solve the following problems of peer-to-peer applications:65 1. Security/authentication —Large peer-to-peer network applications, such as AOL Instant Messenger (AIM) and MSN Instant Messenger, use servers to bootstrap users onto the network. 66 This bootstrapping ensures that the same person uses a particular online identity. 2. Peer discovery—Without a server, it is complex to locate other peers on the network. 3. Network incompatibility—Currently, each popular peer-to-peer application has a set of proprietary protocols that are incompatible with other peer-to-peer networks due to the lack of standards. For example, users on the AIM platform cannot communicate with MSN Instant Messenger users. 4. Platform incompatibility—Software developers must rewrite the low-level core aspects of their peer-to-peer applications for each platform they wish to support. Wireless phones and other mobile devices usually have a limited selection of P2P applications, if any. JXTA attempts to solve these problems by standardizing the low-level protocols that govern peer-to-peer applications. All JXTA-based P2P applications use identical low-level protocols, hence they are compatible with one another. Networks built with the JXTA protocols consist of three basic types of entities—peers or peer groups, advertisements and pipes. A peer is any entity that uses JXTA protocols (Fig. 18.14) to communicate with other peers. Each peer need support only some of the protocols, so devices with low processing power and memory can still participate in JXTA networks (albeit with limited functionality). Peer 18.5 Peer-to-Peer Distributed Computing 841 Protocol Function Peer discovery Peers use this protocol to find other entities in the JXTA network by searching for advertisements. Peer resolver Peers that help a search process (e.g., send and process requests) implement this protocol. Peer information Peers obtain information about other peers via this protocol. Peer membership Peers use this protocol to learn about the requirements of groups, how to apply for membership, how to modify their membership and how to quit a group. Authentication and security are implemented through this protocol. Pipe binding Peers can connect pipes to one another, via advertisements, through this protocol. Endpoint routing Peer routers implement this protocol to provide other routing services to other peers (e.g., tunneling through a firewall). Figure 18.14 | JXTA low-level protocols.67 groups are logical constructs that represent sets of peers. JXTA specifies only two rules regarding peer groups: 1. peers can join or leave groups, and 2. the group administrator, if the group has one, controls access to the group. Advertisements are XML documents formatted according to JXTA specifications that are used by a peer to advertise itself and notify others of its existence. Pipes are virtual communication channels that connect two or more peers for sending and receiving messages among peers. At the simplest level, pipes are one-way communication channels. Two peers communicate by configuring two pipes that "flow" in opposite directions. When a source peer needs to send a message to a destination peer, a pipe is dynamically bound to the two peers. Once the pipe is set up, the source peer can send messages to the destination peer via the pipe. Examples of P2P applications that uses JXTA are: VistaPortal (www.vistaportal.com/), Momentum by InView Software (www.inviewsoftware.com/news/20030305_momentuml.htm) and ifreestyle by Digital Dream (www.digitaldream.jp/en/). 6 8 Self Review 1. How do AOL Instant Messenger and MSN Instant Messenger bootstrap users onto the network? 2. How do peers communicate in networks built with the JXTA protocol? Ans: 1) Both AOL Instant Messenger and MSN Instant Messenger use central servers to bootstrap users onto the network. 2) Peers use one-way communication channels called pipes to exchange messages. 842 Distributed Systems and Web Services 18.6 Grid Computing A few decades ago, the number of computers was relatively small, their computing power was modest and they were expensive and heavily utilized. With the advent of the microprocessor, there are now hundreds of millions of inexpensive, relatively powerful computers in homes and businesses. However, a vast amount of computing power is wasted (e.g., personal computers at home sit idle while people are at work, and work computers sit idle while people are at home). Grid computing links potentially vast numbers of computational resources (such as computers, storage devices and scientific instruments) that are distributed over a network to solve complex, massive, computation-intensive problems. As in other distributed systems, the resources in grid computing are distributed, and users can access them transparently. While clustering typically is localized or centralized, grid computing emphasizes public collaboration. Individuals and research institutes participating in grid computing coordinate resources that are not subject to centralized control. High performance, the goal of grid computing. is achieved by using spare computer power at little or no cost. Grid computing offers better scalability than clusters. For example, the project, which is discussed later, runs on half-a-million PCs while an HP XC cluster runs on up to 512 nodes. Grid computing requires sophisticated software to manage distributed computing tasks on such a massive scale.69-70-71 To enable interoperability of heterogeneous distributed resources, a grid-computing system is often divided into five layers. The application layer, which is the highest level, contains applications that use the lower layers to access distributed resources. The collective layer is responsible for coordinating distributed resources, such as scheduling a task to analyze data received from a scientific device. The resources layer enables applications to request and share a resource. The connectivity layer carries out reliable and secure network communications between resources. The fabric layer accesses physical resources, such as disks.72 The (Search for Extraterrestrial Intelligence at home) project (setiathome.ssl.berkeley.edu/) is a popular grid-computing implementation enables individuals to participate in a scientific effort seeking intelligent life elsewhere in the universe. Computers belonging to participants, when not in use, download data representing signals from outer space from the server, analyze the data and return the results to the server.73, 74 The Globus Alliance (www.globus.org/), based at Argonne National Laboratory, the University of Southern California's Information Sciences Institute, the University of Chicago, the University of Edinburgh and the Swedish Center for Parallel Computers, researches grid-computing resource management and develops software (called the Globus Toolkit) to implement grid computing.75 United Devices (www.ud.com) provides grid-computing solutions for businesses.76 Self Review 1. What makes grid computing a promising trend? 18.7 Java Distributed Computing 843 2. Why is the grid-computing system divided into several layers? Ans: 1) Grid computing helps coordinate distributed resources and allows access to potentially massive amounts of computing resources, enabling complex problems to be solved that usually cannot be approached without enormous computing powers. 2) Distributed resources are typically heterogeneous, so the layered architecture enables interoperability and modularity. 18.7 Java Distributed Computing Java is used widely to implement distributed systems. We discussed developing distributed systems using Java's RMI in Section 17.3.3, and CORBA in Section 17.3.4. Sun provides an article about CORBA and Java at java.sun.com/j2ee/corba/. The following subsections introduce several additional Java technologies that enable developers to build distributed systems and to share and manage distributed resources. 18.7.1 Java Servlets and JavaServer Pages (JSP) Java provides a number of built-in networking capabilities that make it easy to develop Internet-based and Web-based applications. This section focuses on both sides of a client-server relationship which is the foundation for the highest-level views of networking in Java, namely servlets and JavaServer Pages (JSP). A servlet extends the functionality of a server, most frequently a Web server. Using special Syntax, JSP allows Web-page programmers to create pages that use encapsulated Java functionality and even to write scriptlets (Java code embedded in a JSP) of actual Java code directly in the page. 77 A common implementation of the request-response model is between Web browsers and Web servers that interact via the HTTP protocol. When a user accesses a Web site by means of a browser (the client application), a request is sent to the appropriate Web server (the server application). The server normally responds to the client by sending the appropriate HTML Web page. Servlets enhance the functionality of Web servers to provide capabilities such as secure access to Web sites, interacting with databases on behalf of a client, dynamically generating custom documents to be displayed by browsers and maintaining unique session information for each client. In some ways, JSPs look like standard HTML or XML documents. In fact, JSPs normally include HTML or XML markup. Such markup is known as fixedtemplate data or fixed-template text. JSPs are generally used when most of the content sent to the client is static text and markup, and only a small portion of the content is generated dynamically with Java code. Servlets are commonly used when a small portion of the content sent to the client is static text or markup. In fact, some servlets do not produce content. Rather, they perform a task on behalf of the client (such as a database query), then invoke other servlets or JSPs to provide a response. Note that in most cases servlet and JSP technologies are interchangeable. The server that executes a servlet is referred to as the servlet container or servlet 844 Distributed Systems and Web Services engine. JSPs are translated into servlets by the servlet container, which then compiles and executes them. 78 Many developers believe that servlets are appropriate for database-intensive applications that communicate with so-called thin clients—applications that require minimal client-side support. With these applications, the server is responsible for database access and clients connect to the server using HTTP. Thus, the presentation-logic code for generating dynamic content can be written once and reside on the server for access by clients, to allow programmers to create efficient thin clients. Sun Microsystems, through the Java Community Process, is responsible for the development of the servlet and JavaServer Pages specifications. The reference implementations of these standards are under development by the Apache Software Foundation (www.apache.org) as part of the Jakarta Project (jakarta.apache.org). 7 9 As stated on the Jakarta Project's home page, "The goal of the Jakarta Project is to provide commercial-quality server solutions based on the Java platform that arc developed in an open and cooperative fashion." The servlet and JSP part of the Jakarta Project is called Tomcat—this is the official reference implementation of the JSP and servlet standards. The most recent version can be downloaded from the Apache Software Foundation's Web site. Servlets and JavaServer Pages have become so popular that they are now supported by most major Web servers and application servers, including the Sun ONE Application Server, Microsoft's Internet Information Services (IIS), the Apache HTTP Server, BEA's WebLogic application server, IBM's WebSphere application server, the World Wide Web Consortium's Jigsaw Web server and many more. Servlet Life Cycle Architecturally, all servlets implement a common interface, which enables the servlet container to interact with any servlet, regardless of its functionality. A servlet's life cycle begins when the servlet container loads the servlet into memory—usually in response to the first request that the servlet receives. Before the servlet can handle that request, the servlet container initializes the servlet. Then, the servlet can respond to its first request. Typically, each new request results in a new thread of execution (created by the servlet container). The servlet container can also terminate the servlet to release its resources.80 JavaServer Pages (JSP) JavaServer Pages is an extension of servlet technology. JSPs simplify the delivery of dynamic Web content. Web-application programmers can create dynamic content by reusing predefined components and by interacting with components using server-side Java code. JSP programmers can reuse Java objects and create custom tag libraries that encapsulate complex, dynamic functionality. These libraries enable Web-page designers who are not familiar with Java to enhance Web pages with powerful dynamic content and processing capabilities. There are four key components to JSPs: directives, actions, scriptlets and tag libraries. Directives are messages to the JSP container that enable the programmer 18.7 Java Distributed Computing 845 to specify page settings, to include content from other resources and to specify custom tag libraries for use in a JSP. Actions encapsulate functionality in predefined tags that programmers can embed in a JSP. Actions often are performed based on the information sent to the server as part of a particular client request. They also can create Java objects for use in JSP scriptlets. Scriptlets, or scripting elements, enable programmers to insert Java code that interacts with components in a JSP (and possibly other Web-application components) to process requests. The tagextension mechanism enables programmers to create custom tags. Such tags enable programmers to manipulate JSP content.81 When a JSP-enabled server receives the first request for a JSP, the JSP container translates that JSP into a Java servlet that handles the current request and future requests to the JSP. If the new servlet compiles properly, the JSP container invokes the servlet to process the request. The JSP may respond directly to the request or may invoke other Web-application components to assist in processing the request. Overall, a JSP's request/response mechanism and life cycle are identical to that of a servlet. Self Review 1. Suppose a request for a Web page requires the server to generate its content from a database. Which technology discussed in the previous section is most appropriate to accomplish this? 2. Which JSP component enables programmers to insert Java code into JSP? Ans: 1) Servlets should be used, because JSP is used when a small portion of the content is generated dynamically with Java code. 2) Scriptlets. 18.7.2 Jini Many network devices, such as printers and Web servers, provide services to network clients. Jini—a framework for building reliable and fault-tolerant distributed systems with existing Java technologies—extends this idea of providing services beyond industry computer-based networks and into home-based networks. For example, when you arrive home, your car could use a wireless network to notify your home's lighting service to turn on the lights over the walkway. Each service mentioned here has a well-defined interface. The network-printing service provides an interface that enables applications to print documents. A Web server provides an HTTP interface that enables Web browsers to download documents. Your home's lighting service provides an interface that enables other devices on the network to turn the lights on and off. To use a Jini service, a Jini client must be able to discover that a service exists and must know the interface for interacting with the service. For example, your car must be able to discover that your home provides a lighting service and must know the service's interface to interact with it. However, the car need not know the implementation of the underlying lighting service. 846 Distributed Systems and Web Services The Jini lookup service is the heart of the Jini architecture (Fig. 18.15). A lookup service maintains information about available Jini services and enables clients to discover and use them. The process of finding the lookup services and obtaining references to them is called discovery. A service provider discovers and registers the service interface with one or more lookup services to make itself available to clients. Clients discover the lookup services and request a service they require. The lookup services then send a copy of the service interface to the client. Upon obtaining the service interface, the client communicates with the service provider via RMI, as discussed in Section 17.3.3, using the service interface.82 Recall that Jini services register with lookup services to make the Jini service's functionality available to other members in the network. If all goes well, other members use the service, and the service stays up and running perpetually. In reality, however, services fail for many reasons—network outages can make a service unreachable, a physical device associated with a service (e.g., a printer) might need repairs or the service itself could encounter an unrecoverable error. In these and many other situations, a service could become unavailable, and that service might not be able to unregister itself from lookup services to prevent other clients from attempting to use it. One goal of Jini technology is to make Jini communities "self-healing" and able to recover from these common problems. Therefore, when a Jini service registers with a lookup service, the registration is not permanent. The registration is leased for a specific amount of time, after which the lookup service revokes the registration (a service provider can also renew the lease anytime before it expires).83 This prevents problematic services from disrupting the entire Jini network. If a Jini Figure 1 8 . 1 5 | Jini architecture. 18.7 Java Distributed Computing 847 service fails, the Jini service's lease eventually expires and lookup services will no longer provide the failed Jini service to clients. The leasing strategy that Jini employs is strict—if a Jini service provider does not renew the lease, the lookup service terminates the registration when the lease expires and the service becomes unavailable to clients. Self Review 1. What information must a Jini client know in order to use a Jini service? 2. What is the function of the Jini lookup service? Ans: 1) The Jini client must know the service interface to use a Jini service. 2) It acts as a registry for Jini services so that a Jini client can find and use the Jini services that are available. 18.7.3 JavaSpaces JavaSpaces is a Jini service that implements a simple, high-level architecture for building distributed systems. Objects that take part in distributed systems must be able to communicate with one another and share information. The JavaSpaces service provides distributed, shared storage (and shared memory) for Java objects and enables Java objects to communicate, share objects and coordinate tasks using the storage. Any Java-compatible client can put shared objects into the storage. A JavaSpaces service provides three fundamental operations—write, take and read, which allow the communications between objects stored in JavaSpaces. The write operation adds an object into the JavaSpaces service. When clients want to remove an object, they perform a take operation. The take operation removes from the JavaSpaces service an object that satisfies the client's search criteria. The read operation is similar to the take operation, but it does not remove the matching object from the JavaSpaces service so that other clients can still find it. Clients can perform read operations concurrently, but only one client can perform the take operation to remove the object from JavaSpaces.84 JavaSpaces eases the design and development of distributed systems that share Java objects between the service providers and the clients. A JavaSpaces service has six major properties: 85 1. A JavaSpaces service is a

Jini service. 2. Multiple processes can access a JavaSpaces service concurrently, which helps synchronize actions between distributed applications. 3. An object stored in a JavaSpaces service will remain in the service until its lease expires or until a program removes the object from the JavaSpaces service. 4. A JavaSpaces service locates objects by comparing them to a template. The template specifies the search criteria against which the JavaSpaces service compares each object in the JavaSpaces service. When one or more entries match the template, the JavaSpaces service returns a single matching object. 848 Distributed Systems and Web Services 5. JavaSpaces services ensure that operations execute atomically. 6. Objects in a JavaSpaces service are shared. Both service providers and clients can read and take objects from the JavaSpaces service. Now we demonstrate how to build a distributed image-processing system with the JavaSpaces service (Fig. 18.16). Image processing can be a time-consuming task, especially for large images. However, we can improve performance by using JavaSpaces services to build a distributed image-processing system for applying filters to images (e.g., blur, sharpen, etc.). The client first partitions a large image into smaller pieces and writes them into a JavaSpaces service. Multiple image processors (distributed among the system) run in parallel to take the smaller images from the storage and process them by applying appropriate filters, then write the processed images back into the JavaSpaces service. The client takes the processed subimages from the JavaSpaces service and builds the complete, processed image. Self Review 1. (T/F) After a service provider adds an object to a JavaSpaces service, the object will always be there if no one removes it. 2. How might JavaSpaces be used to implement a chat system? Al*9-- 1) False. Even though the client does not execute a take operation, the object will not remain in the JavaSpaces service permanently. Each object stored in a JavaSpaces service has Figure 18.16 | Distributed image-processing application using JavaSpaces. 18.7 Java Distributed Computing 849 a lease; when its lease expires, the object is removed. 2) JavaSpaces can be used to build a chat system in which the message sender writes messages to JavaSpaces and the message receiver reads or takes the message from JavaSpaces. The sender can send messages to a particular recipient or to a group of recipients. 18.7.4 Java Management Extensions (JMX) A crucial function in any distributed system is network management. Businesses increasingly need networks that provide services which are customized to customer demands, consistently available and easily updated. As more organizations add networks and expand existing networks to increase productivity, network management becomes more difficult. The proper functionality of printers, network routers and other devices—many of which play an integral role in the productivity of a firm—is important. As more devices are networked and as networks grow larger, more problems can occur. The demand for managing distributed applications increases when business applications move into distributed environments. 86 Existing tools for managing networked devices often use both standard and proprietary protocols and management tools. The diversity of such proprietary protocols and tools makes managing a diverse network difficult. Many network management schemes are inflexible and nonautomated. This usually means that significant amounts of time and resources are spent trying to fix and operate large networks. New technologies are required that will help shift the burden of routine tasks to the network management software itself and leave special issues to the network manager. Recent technological advances have provided network management developers with the tools necessary to develop smart agents that can assume various responsibilities. These tools allow agents to incorporate themselves into frameworks in which numerous agents interact with each other and provide a dynamic and extensible network management solution. Java Management Extensions (JMX), developed by Sun and other companies in the network management industry, defines a component framework that enables developers to build automated, intelligent and dynamic network management solutions. JMX defines a three-level management architecture —the instrumentation level, the agent level and the manager level (Fig. 18.17). The instrumentation level makes any Java-based object manageable so that the management application can access and operate these objects. The agent level provides services for exposing the managed resources. The manager level gives management applications access to resources created in the instrumentation level and operates these resources via the JMX agents.87 Self Review 1. What are the problems with many existing network-management schemes? 2. What are the three levels in JMX architecture? Briefly describe the function of each. 850 Distributed Systems and Web Services Manager level Agent level Instrumentation level Figure 18.17 | JMX's three-level management architecture. Ans: 1) Many existing network-management schemes are inflexible and nonautomated, because various networked devices may be managed using diverse protocols and tools and require large amounts of time to fix and operate in a large network. 2) The three levels are: the instrumentation level, the agent level and the manager level. The instrumentation level makes any Java-based object manageable. The agent level provides services for exposing the managed resources. The manager level gives management applications access to resources and operates these resources via the JMX agents. 18.8 Web Services Over the past several decades, computing has evolved at an unprecedented pace. This progress impacts organizations in significant ways, forcing information-technology (IT) managers and developers to accept new computing paradigms. Innovations in programming and hardware have led to more powerful and useful technologies, including object-oriented programming, distributed computing, Internet protocols and XML (Extensible Markup Language). At the same time, organizations have learned to leverage the power of their networks and the Internet to gain competitive advantages. Web services technology, which represents the next stage in distributed computing, will profoundly affect organizations in the future. Web services encompass a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. Many factors—including software vendors' widespread support for underlying standards —indicate that Web services will radically change IT architectures and partner relationships. Companies are already implementing Web services to facilitate a wide variety of business processes, such as application integration and business-to-business transactions.88 18.8 Web Services 851 For a distributed system to function correctly, application components executing on different computers throughout a network must be able to communicate. In Section 17.3, we discussed technologies such as DCOM and CORBA that enable communication among distributed components. Unfortunately, DCOM and CORBA cannot intercommunicate easily. Their components often communicate via a COM/CORBA bridge. If DCOM's and CORBA's underlying protocols change, programmers must modify the bridge to reflect the changes. Such problems have impeded distributed computing's ability to facilitate business-process integration and automation. Web services improve distributed computing capabilities by addressing the issue of limited interoperability.89 Web services operate using open (i.e., nonproprietary) standards, which means that Web services can, theoretically, enable any two software components to communicate—regardless of the technologies used to create the components or the platforms on which the components reside. Also, Webservices-based applications are often easier to debug, because Web services use a text-based communications protocol, rather than the (albeit faster) binary communications protocols employed by DCOM and CORBA. Organizations are implementing Web services to improve communication between DCOM and CORBA components and to create standards-based distributed computing systems. Thus, Web services will help organizations achieve the goals of distributed computing and do so more economically.90 The industry's experience with interoperability problems led to the development of open standards for Web services technologies, in an effort to enable crossplatform communication. The primary standard used in Web services is Extensible Markup Language (XML), a language for marking up data so that information can be exchanged between applications and platforms. Microsoft and DevelopMentor developed the Simple Object Access Protocol (SOAP) as a messaging protocol for transporting information and instructions between Web services, using XML as a foundation for the protocol. Two other Web services specifications—Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) —are also based on XML. WSDL provides a standard method of describing Web services and their specific capabilities; UDDI defines XML-based rules for building directories in which companies advertise themselves and their Web services.91 Open standards enable businesses (running different platforms) to communicate and transfer data without designing costly platform-specific software. Web services improve collaborative software development by allowing developers to create applications by combining code written in any language on any platform. Also, Web services promote modular programming. Each specific function in an application can be exposed as a separate Web service. Individuals or businesses can create their own unique applications by mixing and matching Web services that provide the functionality they need. Modularization is less error prone and promotes software reuse. 852 Distributed Systems and Web Services A Web service can be as trivial as multiplying two numbers together or as complex as the functions carried out by an entire customer relationship management (CRM) software system. For example, tracking the location and status of a FedEx package is a Web service (f e d e x . c o m / u s / t r a c k i n g) . Some e-commerce Web sites allow independent developers to harness the power of their sites' technologies by exposing certain functions as Web services. For example, online retailer Amazon. com allows developers to build online stores that search its product databases and display detailed product information via Amazon.com Web Services (www.amazon.com/gp/aws/landing.html). The Google search engine also can be integrated with other functionality through the Google Web APIs (www.google.com/apis), which connect to Google's indices of Web sites using Web services. Amazon. com and Google provide access to their sites' features through SOAP and other standard protocols in exchange for increased exposure. Self Review 1. How are Web services more portable than DCOM? 2. Can a client application written in Java access a Web service written in C#? Why? Ans: 1) Web services provide interoperability, platform independence and language independence. 2) Yes, a client application written in Java can access a Web service written in C#, because Web services technologies use open standards such as XML, SOAP, WSDL and UDDI. 18.8.1 Microsoft's .NET Platform Microsoft introduced the term Web services during the June 2000 launch of its .NET initiative. Today, Microsoft is one of the dominant companies in the Web services market. The .NET initiative includes the Visual Studio .NET integrated development environment which enables programmers to develop Web services in a variety of languages, including C++, C# and Visual Basic .NET However, .NET technologies are available only for Windows 2000 and XP. 92 .NET Web services, which are central to the .NET initiative, extend the concept of software reuse to the Internet by allowing developers to reuse software components that reside on other machines or platforms. Employing Web services as reusable building blocks, programmers can concentrate on their specialties without having to implement every component of an application. For example, a company developing an e-commerce application can subscribe to Web services that process payments and authenticate users. This enables programmers to focus on other, more unique aspects of the e-commerce application. In .NET, a Web service is an application stored on one machine that can be accessed by another machine over a network. In its simplest form, a Web service created in .NET is a class, or a logical grouping of methods and data that simplifies program organization. Methods are defined within a class to perform tasks and return information when their tasks are complete. .NET Web service classes contain 18.8 Web Services 853 certain methods (called Web service methods) that are specified as part of the Web service. These methods can be invoked remotely using RPC. Self Review 1. Can .NET technologies be used on Linux? 2. How have .NET Web services extended the concept of software reuse to the Internet? Ans: 1) No. .NET technologies are available only for Windows 2000 and XP. 2) .NET Web services allow developers to reuse software components that reside on other machines or platforms. 18.8.2 Sun Microsystems and the Sun ONE Platform Sun Microsystems' Web services strategy is based on the Sun Open Net Environment (Sun ONE), which consists of three components—a vision, an architecture and a conceptual model for developing standards-based software.93 The Sun ONE vision incorporates a model for software development, in which critical business information and applications are available at any time to any type of device, including cell phones and PDAs. Sun ONE's goal is to help developers create networks of distributed applications or Web services that are highly reliable and promote the reuse of components and services.94 The Sun ONE architecture is designed to be scalable to ensure reliable access to services. Scalability is crucial; as new technologies and new components are added to systems, more demands are placed on system resources, potentially degrading service.95 The Sun ONE platform is comprised of three products: the Solaris™ Operating Environment, the Infrastructure Software and the Sun ONE Studio. The Infrastructure Software includes the Sun ONE Directory Server and the Sun ONE Portal Server, which offer user authentication and personalization. Other Infrastructure Software capabilities include scheduling management, billing and communication. Sun ONE allows programmers to deploy Web services using third-party products. By integrating disparate products, programmers can develop Web services infrastructures that best suit their companies' requirements. Sun ONE incorporates support for open standards, including XML, SOAP, UDDI and WSDL, to help ensure high levels of interoperability and system integration. 96 Sun ONE promotes the notion that a company's Data, Applications, Reports and Transactions, which compose the conceptual DART model, can be published as services online.97 Using the DART model, companies can organize business applications and processes that involve data, applications, reports and transactions, so that programmers can map business elements to corresponding services. Self Review 1. How is Sun ONE more portable than .NET? 2. What does infrastructure software do in Sun ONE? 854 Distributed Systems and Web Services Ans: 1) .NET is available only on Windows, while Sun ONE is available for many platforms. 2) Infrastructure software offers user authentication and personalization. Infrastructure software is also capable of scheduling management, billing and communication. www.faqs.org/ftp/rfc/pdf/rfcl094.txt.pdf PDF version of the Network File System version 2 protocol specification. www.connectathon.org/nfsv3.pdf PDF version of the Network File System version 3 protocol specification. www.ietf.org/rfc/rfc2054.txt Text version of the WebNFS client specification. www.ietf.org/rfc/rfc3010.txt Text version of the Network File System version 4 protocol specification. www.ietf.org/rfc/rfc2203.txt Text version of the RPCSEC_GSS protocol specification. www.cs.berkeley.edu/projects/sprite/ retrospective.html Brief retrospective on the Sprite Network Operating System. www.es.jamstec.go.jp/esc/eng/index.html Earth simulator home page. www.distributed.net/pressroom/presskit.php Provides news and articles for distributed system projects. library.thinkquest.org/C007645/english/1parallel.htm Compares distributed computing and parallel computing and introduces clustering. www.linux-mag.com/2000-10/clustering_01.html Discusses Linux clustering in detail. www.jini.org The Jini technology site; provides links for downloading the Jini technology implementation. wwws.sun.com/software/jini/specs/js2_0.pdf Provides the JavaSpaces service specification. java.sun.com/products/HavaManagement/wp/ Contains Java Management Extensions (JMX) White Paper. www.jxta.org Official Web site for Project JXTA; contains the newest downloads of the source code and opportunities to participate in developing JXTA. www.openp2p.com This O'Reilly Network online resource provides articles and links related to peer-to-peer technologies. java.sun.com/j2ee/corba/ Discusses CORBA technology and the Java platform. java.sun.com/j2ee Provides reference material for all Java 2 Enterprise Edition technologies, including servlets and JavaServer Pages. jakarta.apache.org The Apache Jakarta Project site provides resources for servlets and JavaServer Pages and the reference implementation of these technologies. www.webservices.org The Web Services Community Portal contains the latest news regarding Web services and Web services vendors. There is also a collection of articles and papers relating to Web services technologies. www.microsoft.com/net Microsoft's .NET site provides .NET resources, including product information, news, events and downloads. www.sun.com/sunone Sun Microsystems' Sun ONE initiative site explains the Open Net Environment and provides news about the products and its users. www.xmethods.com Lists free, publicly available Web services and provides links for tutorials and implementations of Web services. Summary Networked file systems allow clients to access files stored on remote computers. Distributed file systems, which are the focus of this chapter, are special examples of networked file systems that allow transparent access to remote files. A distributed file server can be either stateful or stateless. In a stateful system, the server keeps state infor- mation of the client requests so that subsequent access to the file is easier. In a stateless system, the client must specify which file to access in each request. Distributed file systems provide the illusion of transparency. Complete file location transparency means that the user is unaware of the physical location of a file within a Summary distributed file system; the user just sees a global file system. Many distributed systems implement client caching to avoid the overhead of multiple RPCs. Clients keep a local copy of a file and flush it to the server from time to time. Because there are multiple copies of the same file, files can become inconsistent. Distributed file systems are designed to share information among large groups of computers. To reach this goal new computers should be able to be added to the system easily. Distributed file systems should be scalable. There are two primary security concerns in distributed file systems: ensuring secure communications and guaranteeing correct file access permissions. NFS (Network File System) versions 2 and version 3 assume a stateless server implementation, which makes fault tolerance easier to implement than with a stateful server. With stateless servers, if the server crashes, the client can simply retry until the server responds. NFS-4 is stateful. The stateful server allows faster access to files. However, if the server crashes, all the state information of the client is lost, so the client needs to rebuild its state on the server before retrying. Client-side caching is essential to efficient distributed file system operation. NFS-4 extends the client-caching scheme through delegation, which allows the server to temporarily transfer the control of a file to a client. When the server grants a read delegation of a particular file to the client, then no other client can write to that file. When the server grants a write delegation of a particular file to the client, then no other client can read or write to that file. If another client requests a file that has been delegated, the server will revoke the delegation and request that the original client flush the file to disk. AFS (Andrew File System) is a global file system that appears as a branch of a traditional UNIX file system at each workstation. AFS is completely location transparent and provides a high degree of availability, which means the files are always available regardless of the user's location. AFS is based on the client-server model and relies on RPCs for communication. The Sprite file system has the perceived form of a UNIX file system, and every client has the exact same view of the hierarchy. It goes a step further in UNIX file system emulation by allowing transparent remote access to I/O devices (which are represented as files in UNIX). In Sprite, both the client and the server keep a cache. To open a file, the client first checks its cache, then makes a request to its backing store (which is generally a server). If the server is unable to satisfy the request from its cache, it reads the data from disk. Both caches retain a copy of the data when 855 it is returned to the client. When a client writes a block, it writes into its cache. Sprite flushes updated pages to the server every 30 seconds. Clustering involves interconnecting nodes (singleprocessor computers or multiprocessor computers) within a high-speed LAN to function as a single parallel computer. The set of nodes that forms the single parallel machine is called a cluster. Clustering enables multiple computers to work together to solve large and complex problems such as weather prediction. There are three major types of clusters: high-performance clusters, high-availability clusters and load-balancing clusters. In a high-performance cluster, all the nodes in the cluster work to improve performance. In a high-availability cluster, only some of the nodes in the cluster are working while others act as backups. If working nodes or their components fail, the backup nodes immediately start running and take over the jobs that were being executed on the failed nodes, without interrupting service. In a load-balancing cluster, a particular node works as a load balancer to distribute the load to a set of nodes so that all hardware is efficiently utilized. P2P (peer-to-peer) applications are different from client/server applications. Instead of segregating computers by function, all computers act as both clients and servers. Each peer has the ability to discover and communicate with other peers. Peers may share resources (such as large multimedia files) with others. P2P applications can be implemented in two forms: centralized and decentralized. A centralized P2P application uses a server that connects to each peer. Centralized P2P applications exemplify the client/server relationship. A pure P2P application, also called a decentralized P2P application, does not have a server and therefore does not suffer from the same deficiencies as applications that depend on servers. Distributed searches make networks more robust by removing single points of failure, such as servers. Not only can peers find information in this way, but they can search for other peers via distributed searches. Grid computing links computational resources that are distributed over the wide area network (such as computers, data storages and scientific devices) to solve complex problems. Resources in grid computing are distributed, and users can access these resources transparently without knowing where they are located. Grid computing emphasizes public collaboration—individuals and research institutes coordinate resources that are not subject to a centralized control to deliver qualities of service. Grid computing has the same advantage as clustering—high per- 856 Distributed Systems and Web Services formance, achieved by cost-effectively using spare computer power and collaborating resources. However, grid computing requires advanced software to manage distributed computing tasks efficiently and reliably. A Java servlet extends the functionality of a server, most frequently a Web server. Using special syntax, JSP allows Web-page programmers to create pages that use encapsulated Java functionality and even to write scriptlets (Java code embedded in a JSP) of actual Java code directly in the page. JSPs are generally used when most of the content sent to the client is static text and markup and only a small portion of the content is generated dynamically with Java code. Servlets are commonly used when a small portion of the content sent to the client is static text or markup. In fact, some servlets do not produce content. Rather, they perform a task on behalf of the client (such as a database query), then invoke other servlets or JSPs to provide a response. Project JXTA strives to create a standard, low-level platform and language-independent protocol that promotes interoperability among peer-to-peer applications. JXTA provides a foundation on which developers can build any type of P2P application. Jini—a framework for building reliable and fault-tolerant distributed systems with existing Java technologies— extends this idea of providing services beyond computer- based networks and into home-based networks. The JavaSpaces service provides distributed, shared storage (and shared memory) for Java objects and enables Java objects to communicate, share objects and coordinate tasks using the storage. Any Java-compatible client can put shared objects into the storage. Java Management Extensions (JMX) defines a component framework that enables developers to build automated, intelligent and dynamic network-management solutions. Web services is an emerging area of distributed computing. Web services encompass a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. Web services can enable any two software components to communicate — regardless of the technologies used to create the components or the platforms on which the components reside. Web services improve collaborative software development by allowing developers to create applications by combining code written in any language on any platform. Also, Web services promote modular programming. Each specific function in an application can be exposed as a separate Web service. With separate Web service components, individuals or businesses can create their own unique applications by mixing and matching Web services that provide the functionality they need. Such modularization is less error prone and promotes software reuse Key Terms advertisement (in JXTA)—XML document formatted according to JXTA specifications that is used by a peer to advertise itself and notify others of its existence. agent level (in JMX)—JMX level that provides services for exposing the managed resources. Andrew File System (AFS)—Scalable distributed file system that would grow to support a large community while being secure. AFS is a global file system that appears as a branch of a traditional UNIX file system at each workstation. AFS is completely location transparent and provides a high degree of availability. Apache Software Foundation—Provides open-source software, such as Tomcat, the official reference implementation of JSP and servlet specifications. application layer (in grid computing) — Contains applications that use lower-level layers to access the distributed resources. available volume storage group (AVSG) (in Coda)—Members of the VSG with which the client can communicate. Beowulf cluster—Linux clustering solution, which is a highperformance cluster. A Beowulf cluster may contain several nodes or several hundred. Theoretically, all nodes have Linux installed as their operating system and are interconnected with high-speed Ethernet. Usually, all the nodes in the cluster are connected within a single room to form a supercomputer. callback (in AFS)—Sent by the server to notify the client that the cached file is modified. cell (in AFS-3)—Unit in AFS-3, which preserves namespace continuity while allowing different systems administrators to oversee each cell. centralized P2P application—Uses a server that connects to each peer. client caching—Clients keep local copies of files and flush them to the server after having modified the files. client modification log (CML) (in Coda)—Log which is updated to reflect file changes on disk. Key cluster—Set of nodes that forms what appears to be a single parallel machine. clustering—Interconnection of nodes within a high-speed LAN so that they function as a single parallel computer. Coda Optimistic Protocol—Protocol used by Coda clients to write a copy of the file to each of the members of the AVSG, which provides a consistent view of a file within an AVSG. collective layer (in grid computing)—Layer responsible for coordinating distributed resources, such as scheduling a task to analyze data received from a scientific device. concurrent write-sharing—Occurs when two clients modify cached copies of the same file. connectivity layer (in grid computing)—Layer that carries out reliable and secure transactions between distributed resources. decentralized peer-to-peer application—Also called a pure peer-to-peer application. It does not have a server and therefore does not suffer from the same deficiencies as applications that depend on servers. delegation (in NFS)—Allows the server to temporarily transfer the control of a file to a client. When the server grants a read delegation of a particular file, then no other clients can write to that file, but they can read it. When the server grants a write delegation of a particular file to the client, then no other clients can read or write that file. discovery (in Jini)—Process of finding the lookup services and obtaining references to them. distributed file system—Special examples of networked file systems that allow transparent access to remote files. distributed search—Searching technology used in peer-to-peer applications to make networks more robust by removing single points of failure, such as servers. In a distributed search, if a peer cannot answer the client's request, the peer forwards the request to its directly connected peers and the search is distributed to the entire peer-to-peer network. domain (in Sprite file system)—Unit that represents a portion of the global file hierarchy and is stored at one server. Terms 857 Extensible Markup Language (XML)—Language for marking up data so that information can be exchanged between applications and platforms. fabric layer (in grid computing)—Layer that accesses physical resources, such as disks. file handle—Identifies a file on the file server with file type, location and access permissions. file identifiers (fids) (in AFS)—Entity that specifies a volume, an index within a volume and an identifier to guarantee object uniqueness within a volume. grid computing—Links computational resources that are distributed over the wide area network (such as computers, data storages and scientific devices) to solve complex problems. head node (in a Beowulf cluster)—Node, also called master node, that acts as a server to distribute the workload, control access to the cluster and handle the shared resources. high-availability cluster—Cluster in which only some of the nodes are working while other nodes act as backups. The goal of a high-availability cluster is to stay up all the time. high-performance cluster—Cluster in which all the nodes work to achieve maximum performance. hoarding stage (in Coda) —Stage that clients enter when they are connected to Coda. In this stage, clients prepare for a possible disconnection from the system by caching any requested file. idempotent request—A requested operation that if performed several times will return the same result, so it is acceptable to perform the same operation twice. instrumentation level (in JMX)—Makes any Java-based object manageable so that the management application can access and operate these objects. invalidate—To invalidate a file, the client checks the time stamp of the copy of the file on the server. If this copy has been updated since the client last copied the file, then the client must download the latest version. If the server copy has not been updated since the client copied it, then the client can work on its cached copy. emulation stage—When a Coda client becomes disconnected, it is said to enter the emulation stage. During this stage all file read requests are satisfied from cache. Write requests during the emulation stage occur in two steps. First, the file is updated on disk. Second, a log called the client modification log (CML) is updated to reflect file changes. Jakarta project—Provides commercial-quality server solutions based on the Java platform that are developed in an open and cooperative fashion. export local file—Performed by an NFS server to make the local directory of files available to the remote client via the mount protocol. Java Management Extensions (JMX)—Developed by Sun and network management industry leaders, which defines a component framework that enables developers to build Java Community Process—Open organization that focus on developing Java technology specifications, including servlet and JavaServer Pages. 858 Distributed Systems and Web Services automated, intelligent and dynamic network management solutions. JavaServer Pages (JSP)—Allows Web-page programmers to create pages that use encapsulated Java functionality and even to write scriptlets of actual Java code directly in the page. JavaSpaces—Jini service that implements a simple, high-level architecture for building distributed systems. The JavaSpaces service provides distributed, shared storage (and shared memory) for Java objects and enables Java objects to communicate, share objects and coordinate tasks using the storage. Jini—Framework for building reliable and fault-tolerant distributed systems with existing Java technologies. Jini extends the idea of providing services beyond industrybased computer networks and into home-based networks. JXTA—Project created at Sun Microsystems, Inc., which creates a standard, low-level, platform and language-independent protocol that promotes interoperability among peer-to-peer applications. lease—Agreement between the client and server for controlling file locks. load-balancing cluster—Cluster in which one particular node works as a load balancer to distribute the load to a set of nodes, so that all hardware is efficiently utilized. load balancer—Node in a load-balancing cluster that distributes the workload (such as thousands of requests from the clients) to a set of nodes so that all hardware is efficiently utilized. lookup service—Heart of the Jini architecture, which maintains information about available Jini services and enables clients to discover and use them. manager level (in JMX)—Level that gives a management application access to managed resources (created in the instrumentation level) and operates these resources via the JMX agents. master node (in a Beowulf cluster)—Node also known as the head node, that acts as a server to distribute the workload, control the access to the cluster and handle the shared resources. multicomputer system—System in which processors do not share a common memory or bus. Each processor has access to its own resources. These independent processors are connected in a network to operate cooperatively to form a multicomputer system. Network File System (NFS) — Current de facto standard for network file sharing, natively supported in most varieties of UNIX (and many other operating systems) with client and server software available for other common platforms. network file system—System that allows clients to access files on remote computers. Network file systems do not provide location transparency as distributed file systems do. nonvolatile storage—The contents of nonvolatile storage are not lost when the machine loses power or is powered off. peer—Single computer in a peer-to-peer system. peer discovery—Finding peers in a peer-to-peer application. peer group—Logical construct that represents a set of peers. A peer group is one of the basic types of entities in a network built with the JXTA protocols. peer-to-peer (P2P) application—Distributes processing responsibilities and information to many computers, thus reclaiming otherwise wasted computing power and storage space and eliminating central points of failure. In a peer-to-peer system, each peer performs both client and server functions. pipe (in JXTA)—Virtual communication channel that connects two or more peers for sending and receiving messages among peers. preferred server (PS) (in Coda)—Member of the AVSG that provides copies of files for Venus. prefix table (in Sprite)—Table that stores domain information to aid in file lookups. Each entry in the table represents a separate domain and consists of the absolute path to the root directory within the domain, the server which houses the domain and a token that identifies the domain. pure peer-to-peer application—Also called a decentralized peer-to-peer application. It does not have a server and therefore does not suffer from the same deficiencies as applications that depend on servers. read operation (in JavaSpaces)—Operation that is similar to the take operation, but does not remove the object from the JavaSpaces service so that other clients can still find it. reintegration stage (in Coda)—Stage right after the client reconnects to the system during which Venus asynchronously updates the server using the CML. resources layer (in grid computing)—Layer that enables applications to query and share a resource. safe asynchronous write (in NFS-3)—Allows a server to return before a write has been completed. scriptlet—Java code embedded in a JSP. sequential write-sharing—Occurs when one client modifies a file, then another client tries to read or write its own cached copy of the file. Sequential write-sharing introduces cache inconsistency. Exercises servlet—Enhances the functionality ofWeb servers to provide capabilities such as secure access to Web sites, interacting with databases on behalf of a client, dynamically generating custom documents to be displayed by browsers and maintaining unique session information for each client. servlet container—Server that executes a servlet. Also known as servlet engine. Simple Object Access Protocol (SOAP)—Messaging protocol for transporting information and instructions between Web services, using XML as a foundation for the protocol. slave node (in a Beowulf cluster)—Beowulf cluster node that is not a head node. Sprite—Distributed operating system whose goal is transparency and complete consistency. stateful server—Keeps state information of the client requests—such as the file name, a pointer to the file and the current position in the file—so that the subsequent access to the file is easier and faster. stateless server—The server does not keep state information of the client requests, so the client must specify which file to access in each request. Sun Open Net Environment (Sun ONE)—Consists of three components—a vision, an architecture and a conceptual model for developing standards-based software. take operation (in JavaSpaces)—Removes from the JavaSpaces service an object that matches the given criteria. Take operations, together with write and read operations, allow distributed applications to dynamically exchange objects within JavaSpaces services. thin client—Application that requires minimal client-side support. Tomcat—Official reference implementation of the JSP and servlet standards. Universal Description, Discovery and Integration (UDDI)— Defines XML-based rules for building directories in 859 which companies advertise themselves and their Web services. update set (in Coda) — Specifies, to each member of the AVSG, the members of the AVSG that have successfully performed the write. Venus (in AFS) —User-level process that interacts with the Vice processes run on distributed file servers to govern distributed file access. virtual file system (VFS)—Provides the abstraction of a common file system at each client and is responsible for all distributed file operations. Vice (in AFS)—Entity that governs distributed file access in AFS. volume (in AFS-2) —Introduced in AFS-2 to manage subtrees. Volumes are primarily of administrative value, allowing replication and isolation of certain subtrees, and are therefore transparent to users. volume storage group (VSG) (in Coda)—Volumes are logical pieces of the file system and are replicated physically across multiple file servers. Servers that hold the same volume are known as a volume storage group (VSG). WebNFS-Allows NFS clients to access WebNFS-enabled servers with a minimum of protocol overhead. Marketed as the file system for the Web, WebNFS is designed to improve NFS functionality and performance over wide area Internet and intranets. Web service—Technology that encompasses a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. Web Services Description Language (WSDL)—Provides a standard method of describing Web services and their specific capabilities. Web service methods (in .NET)—Methods contained in a .NET Web service class. write operation (in JavaSpaces) — Operation that adds an object into the JavaSpaces service. Exercises 18.1 Explain the benefits and drawbacks of client caching in distributed file systems. 18.2 What are the primary concerns of implementing distributed file systems? 18.3 A file server can be either stateful or stateless. What are the advantages and disadvantages of each implementation? 18.4 How can distributed file systems ensure cache consistency? 18.5 Is NFS-4 more secure than NFS-2 and NFS-3? How? 18.6 In AFS-1, each time the VFS tried to open a cached file it would have Venus contact the server to make sure that the file was still valid. What is the problem with this approach? How do AFS-2 and AFS-3 solve this problem? 18.7 Explain how Coda provides fault tolerance. 18.8 Describe the three stages Coda clients go through when reading and writing to a file on the Coda server. 860 Distributed Systems and Web Services 18.9 How does Sprite handle server crashes? 18.10 Describe the differences between high-performance clusters, high-availability clusters and load-balance clusters. Give examples of each of these cluster types in use. 18.11 List the essential elements for building a Beowulf cluster. 18.12 Compare centralized and pure peer-to-peer applications. What are the advantages and disadvantages of each approach? 18.13 Given the peer network of Fig. 18.18, apply the distributed search mechanism used in Gnutella to find the requested resource which is on P 7 . Assume the search starts at P1. Suggested Projects 18.19 Research transparency in distributed file systems. What research is currently being done in that field? 18.20 Research client caching in distributed file systems. What research is currently being done in that field? Distributed file systems play an important role in distributed systems.98, 99 Examples of distributed file systems are NFS,136, 101, 102, 103, 104, 105 AFS,106, 107 Coda 108 and Sprite.109, 110, 111, 112 Clustering enables multiple computers to work together to solve large and complex problems.113, 114 The most popular clustering technologies are Beowulf and Windows Server 2003 clusters.115, 116 The peer-to-peer distributed computing model is another architecture for building distributed systems.117, 118, 119 Peer-to-peer applications use a distributed search to find a peer in the network.120, 121 Sun provides several technologies for developing distributed computing systems with servlets and JavaServer Pages (JSP),122, 123, 124, 125 including Jini,126, 127 JavaSpaces128 and Figure 18.18 | P2P system with seven peers. 18.14 Given the peer network as shown in Fig. 18.18, apply the distributed search mechanism used in Freenet to find the requested resource, which is on P7. Assume the search starts at P1. 18.15 Describe the three basic types of entities of JXTA. 18.16 Explain why the Jini lookup service is the heart of the Jini architecture. 18.17 When a service provider registers a service with a Jini lookup service, is the registration permanent? What is the benefit of this design? 18.18 What problems do DCOM and CORBA have? How can Web services solve this problem? 18.21 Prepare a research paper for Web services. How is used? JMX.129 Web services is an emerging area of distributed computing. Web services encompass a set of related standards that can enable any two computer applications to communicate and exchange data via the Internet. Web services can enable any two software components to communicate—regardless of the technologies used to create the components or the platforms on which the components reside.130, 131, 132, 133 Web services is one of the most recent distributed systems architectures.134, 135 Distributed systems exist in many forms— distributed object-based systems, distributed file systems, distributed document-based systems and distributed coordinationbased systems. Tanenbaum and Steen discuss each of these types Works Cited 861 of distributed systems.136 Both centralized and decentralized P2P applications have scalability limitations. Grid computing enables the creation of a large-scale infrastructure used to share computing power and data resources.137, 138 Works Cited 1. Tanenbaum, A., and R. Renesse, "Distributed Operating Systems," ACM Computing Surveys, Vol. 17, No. 4, December 1985, pp. 419-470. 19. Morris, J. H., et al., "Andrew: A Distributed Personal Computing Environment," Communications of the ACM, Vol. 29, No. 3, March 1986, pp. 186-187. 2. Levy, E., and A. Silberschatz, "Distributed File Systems: Concepts and Examples," ACM Computing Surveys, Vol. 22, No. 4, December 1990, pp. 321-374. 20. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, p. 10. 21. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, pp. 11-12. 22. Morris, J. H., et al., "Andrew: A Distributed Personal Computing Environment," Communications of the ACM, Vol. 29, No. 3, March 1986, p. 193. 3. Howard, J. H., et al, "Scale and Performance in a Distributed File System", ACM Transactions on Computer Systems, Vol. 6, No. 1, February 1988, pp. 51-81. 4. Andrade, H., "Distributed File Systems", . 5. Tanenbaum, A., and R. Renesse, "Distributed Operating Systems," ACM Computing Surveys, Vol. 17, No. 4, December 1985, pp. 429-430. 6. "Difference Between Fault Tolerance and High-Availability," . 7. Froese, K. W., and R. B. Bunt, "The Effect of Client Caching on File Server Workloads," Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, Wailea, HI, January 1996, pp. 15-159. 8. Tanenbaum, A., and R. Renesse, "Distributed Operating Systems," ACM Computing Surveys, Vol. 17, No. 4, December 1985, pp. 427-429. 9. Sandberg, R., "The Sun Network Filesystem: Design, Implementation and Experience", Proceedings of The USENIX Conference, 1986, pp. 300-314, . 10. Nowicki, B., "NFS: Network File System Protocol Specification," March 1989, < w w w . i e t f . o r g / r f c / r f c l 0 9 4 . t x t > . 11. Callaghan, B., "NFS Version 3 Protocol Specification," , June 1995. 12. Network File Systems: Version 2 Protocol Specification, . 13. Callaghan, B., "WebNFS Client Specification," October 1996, . 14. Shepler, S., "RFC 2624-NFS Version 4 Design Considerations," June 1999, . 15. Pawlowski, B., et al., "The NFS Version 4 Protocol," . 16. Shepler, S., et al., "NFS Version 4 Protocol," December, 2000, . 17. Eisler, M., "NFS Version 4 Security," December 1998, . 18. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, p. 9. 23. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, p. 12. 24. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, pp. 13-15. 25. Coda credits, < w w w . m i t . e d u / a f s / s i p b / p r o j e c t / c o d a / s r c / coda/CREDITS>. 26. Satyanarayanan, M., "Scalable, Secure and Highly Available Distributed File Access," IEEE Computer, May 1990, p. 15. 27. Satyanarayanan, M., "The Evolution of Coda," ACM Transactions on Computer Systems, Vol. 20, No. 2, May 2002, p. 89. 28. Satyanarayanan, M., "The Evolution of Coda," ACM Transactions on Computer Systems,Vol. 20, No. 2, May 2002, pp. 89-90. 29. Satyanarayanan, M., "The Evolution of Coda," ACM Transactions on Computer Systems, Vol. 20, No. 2, May 2002, pp. 90-91. 30. Satyanarayanan, M., "The Evolution of Coda," ACM Transactions on Computer Systems, Vol. 20,No. 2,May 2002,pp. 94-95. 31. Ousterhout, J., "A Brief Retrospective on the Sprite Network Operating System," . 32. Ousterhout, X, et al., "The Sprite Network Operating System," IEEE Computer, February 1988, p. 28. 33. Ousterhout, J., et al., "The Sprite Network Operating System," IEEE Computer, February 1988, pp. 28-30. 34. Ousterhout, J., et al, "The Sprite Network Operating System," IEEE Computer, February 1988, pp. 30-31. 35. Nelson, M., "Caching in the Sprite Network File System," ACM Transactions on Computer Surveys, Vol. 6, No. 1, February 1988, pp. 134-154. 36. Baker, M., and J. Ousterhout, "Availability in the Sprite Distributed File System," Proceedings of the 4th workshop on ACM SIGOPS European workshop, September 1990, pp. 1-4. 37. McGregor, J. D., and A. M. Riehl, "The Future of High Performance Super Computers in Science and Engineering," Communications of the ACM, Vol. 32, No. 9, September 1989, p. 1092. 38. Earth Simulator Performance, . . 63. Oram, A., "Gnutella and Freenet Represent True Technological Innovation," . 42. Press Release: "Intel and Scientific Community Announce Cancer Research Program," April 3, 2001, . 64. Oram, A., "Gnutella and Freenet Represent True Technological Innovation," < w w w .o r e 12/magazi ne/gnutel1a.html?page=l>. 43. Parallel Processing, . 65. Gong, L., "Project JXTA: A Technology Overview." . 44. Alternatives and forms of DC; < l i b r a r y . t h i n k q u e s t . o r g / C007645/english/0-alternatives.htm?tqskipl=l&tqtime= 0818>. 45. "Linux Clustering in Depth", Linux Magazine, . 46. 'Distributed Computing: Power Grid", . 47. "Unraveling the Mysteries of Clustering", . 48. The UNIX Operating System: A Robust, Standardized Foundation for Cluster Architectures, . 66. AOL Instant Messenger™, . 67. Gong, L., "Project JXTA: A Technology Overview" . 68. See, S., "Project JXTA Technology Overview," . 69. "Grid Computing," < s e a r c h c i o . t e c h t a r g e t . c o m / s D e f i n i t i o n/0,,sidl9_gci773157,00.html>. 70. Foster, I., "Internet Computing and the Emerging Grid." . 71. "Grid Computing," grid.html >. . 51. Understanding Clustering, . 74. Hipschman, R., "How ome.ssl.berkeley.edu/about_seti/ about_seti_at_home_l.html>. 52. Jan Lindheim, C, "Building a Beowulf System", . 53. "What's a Beowulf?" beowulf_book/node9.html>. File B: Printer: Figure 19.7 I Access control list derived from the access control matrix. 19.4 Access Control 891 object. Capabilities are created by carefully guarded operating system routines. A subject that possesses a capability may perform certain operations, including creating copies of the capability or passing it as a parameter. Upon creation of an object, a capability for that object is created. This original capability includes full privileges for the new object. The subject that creates the object may pass copies of the capability to other subjects. Likewise, a subject receiving a capability may use it to access the object, or the subject may create additional copies and pass them to other subjects. When one subject passes a capability to another, it may reduce the associated privileges. Thus, as a capability propagates through the system, its privilege set can either remain the same or decrease in size. Users must be prevented from creating capabilities arbitrarily. This can be accomplished by storing capabilities in segments that user processes cannot access. The identifier in a capability may be implemented as a pointer to the desired object, or it may be a unique bit sequence (i.e., a token). Pointers simplify access to the address at which the object is stored, but all such pointers in the system must be updated if the object is moved, which can degrade performance. When using tokens, capabilities do not rely on the object's location in memory. However, because a token does not specify the location of its corresponding object, tokens require that the object's address be determined when the capability is first used. A hashing mechanism implements token-based capabilities efficiently; high-speed caches often reduce the overhead of repeated references to the same object. Systems that employ capabilities can suffer from the "lost object" problem. If the last remaining capability for an object is destroyed, the associated object can no longer be accessed. To prevent this problem, many operating systems ensure that the system always maintains at least one capability for each object. Controlling the propagation of capabilities is a difficult problem. Systems generally do not allow direct manipulation of capabilities by users; instead capability manipulation is performed by the operating system on behalf of the users. Keeping track of capabilities is an important task that becomes difficult in multiuser systems containing a large numbers of capabilities. Many systems employ a directory structure to manage their capabilities.39 Self Review 1. In what type of environment are access control lists more appropriate than access control matrices? 2. Discuss the advantages and disadvantages of token-based and pointer-based capabilities. Ans: 1) Access control lists are more appropriate than access control matrices when the matrices are sparse. In this case, access control lists require much less space than access control matrices. 2) Pointers simplify access to the address at which the object is stored, but all such pointers in the system must be updated if the object is moved, which can degrade performance. When using tokens, capabilities are independent of the object's location in memory. However, because a token does not specify the location of its corresponding object, tokens require that an object's address be determined when the capability is first used. 892 Security 19.5 Security Attacks Recent cyber attacks on e-businesses have made the front pages of newspapers worldwide. Denial-of-service attacks (DoS), viruses and worms have cost companies billions of dollars and caused countless hours of frustration. Many of these attacks allow the perpetrator to penetrate a network or system, which can lead to data theft, data corruption and other attacks. In this section we discuss several types of attacks against computer systems. Section 19.6, Attack Prevention and Security Solutions, discusses possible solutions to protect a computer's information and integrity. 19.5.1 Cryptanalysis A cryptanalytic attack attempts to decrypt ciphertext without possessing the decryption key. The most common form of cryptanalytic attacks are those in which the encryption algorithm is analyzed to find relations between bits of the encryption key and bits of the ciphertext. The goal of such an attack is to determine the key from the ciphertext. Weak statistical trends between ciphertext and keys can be exploited to gain knowledge about the key. Proper key management and key expiration dates can reduce susceptibility to cryptanalytic attacks. The longer an encryption key is used, the more ciphertext an attacker can use to derive the key. If a key is covertly recovered by an attacker, it can be used to decrypt every message that uses the key. Self Review 1. How do key expiration dates reduce the effectiveness of cryptanalytic attacks? Ans: 1) The longer an encryption key is used, the more ciphertext an attacker can use to derive the key. Key expiration addresses this concern by limiting hour long each key is in use. 19.5.2 Viruses and Worms A virus is executable code—often sent as an attachment to an e-mail message or hidden in files such as audio clips, video clips and games—that attaches to or overwrites other files to replicate itself. Viruses can corrupt files, control applications or even erase a hard drive. Today, viruses can be spread across a network simply by sharing "infected" files embedded in e-mail attachments, documents or programs. A worm is executable code that spreads by infecting files over a network. Worms rarely require any user action to propagate, nor do they need to be attached to another program or file to spread. Once a virus or worm is released, it can spread rapidly, often infecting millions of computers worldwide within minutes or hours. Viruses can be classified as follows: 1. boot sector virus—infects the boot sector of the computer's hard disk, allowing it to load with the operating system and potentially control the system. 19.2 Security Attacks 893 2. transient virus—attaches itself to a particular computer program. The virus is activated when the program is run and deactivated when the program is terminated. 3. resident virus—once loaded into the memory of a computer, it operates until the computer is powered down. 4. logic bomb—executes its code, or payload, when a given condition is met. An example of a logic bomb is a time bomb, which is activated when the clock on the computer matches a certain time or date. A Trojan horse is a malicious program that hides within a trusted program or simulates a legitimate program or feature, while actually causing damage to the computer or network when the program is executed. The name Trojan horse originates from the story of the Trojan War in Greek legend. [Note: In this story, Greek warriors hid inside a wooden horse, which the Trojans took within the walls of the city of Troy. When night fell and the Trojans were asleep, the Greek warriors came out of the horse and opened the gates to the city, letting the Greek army enter and destroy the city of Troy] Trojan horse programs can be particularly difficult to detect, because they appear to be legitimate and operational applications. Back-door programs are resident viruses that allow the sender complete, undetected access to the victim's computer resources. These types of viruses are especially threatening to the victim, as they can be programmed to log every keystroke (capturing all passwords, credit card numbers, etc.). Widespread Viruses Two viruses that attracted significant media attention are Melissa, which struck in March 1999, and the ILOVEYOU virus that hit in May 2000. Each caused billions of dollars of damage. The Melissa virus spread in Microsoft Word documents sent via e-mail. When the document was opened, the virus was triggered, at which point it accessed a user's Microsoft Outlook address book (i.e., a list of e-mail addresses) on that computer and sent the infected Word attachment by e-mail to a maximum of 50 people in the user's address book. Each time another user opened the attachment, the virus would send up to 50 additional messages. Once resident in a system, the virus also infected any files saved using Microsoft Word. The ILOVEYOU virus was sent as an attachment to an e-mail posing as a love letter. The message in the e-mail said "Kindly check the attached love letter corning from me." Once opened, the virus accessed the Microsoft Outlook address book and sent messages to each address listed, enabling the virus to spread rapidly worldwide. The virus corrupted many types of files, including operating system files. Many networks were disabled for days due to the massive number of e-mails generated. This virus exposed e-mail security inadequacies, such as a lack of software to scan file attachments for security threats before they are opened. It also taught users to be more aware of suspicious e-mails, even those from someone with whom they are familiar. 894 Security Sapphire/Slammer Worm: Analysis and Implications Worms spread by exploiting weaknesses in communications channels established by software, either by applications or by the operating system. Once a weakness is discovered, a worm can produce network traffic sufficient to disable a single computer or a network of computers. Further, a worm can be designed to execute code on the computer it infects, potentially allowing the worm's creator to obtain or destroy sensitive information. Worm attacks that have received media attention include Nimda, Code Red and Sapphire, also called Slammer. Slammer, which infected most vulnerable computers within 10 minutes of its release on January 25, 2003, doubled the number of computers it infected every 8.5 seconds. [Note: Vulnerable computers were those running Microsoft SQLServer 2000 to which a security patch, released by Microsoft in July 2002, was not applied.] The infection rate was two orders of magnitude faster than its notable predecessor, the Code Red worm. The latter, a 4KB worm, instantiated multiple threads to create TCP connections to infect new hosts. On the contrary, Slammer operated over UDP and its payload was contained in a single 404byte UDP packet. The connectionless UDP protocol, coupled with a random-scanning algorithm to generate target IP addresses, made the Slammer worm particularly virulent. A random-scanning algorithm uses pseudorandom numbers to generate a broad distribution of IP addresses as targets to infect. The Slammer worm caused system and network outages due to network saturation by its UDP packets. Interestingly, the worm included no malicious payload and attacked a vulnerability in an application of relatively limited worldwide use. The Slammer worm also contained what appeared to be a logic error in its randomscanning algorithm that significantly limited the number of IP addresses the worm could reach. 40 Antivirus software can protect against viruses and some worms. Most antivirus software is reactive, meaning that it can attack known viruses, rather than protecting against unknown or future viruses. We discuss antivirus software in Section 19.6.3. Self Review 1. How do worms differ from other viruses? 2. What weaknesses in computer systems did the Melissa and ILOVEYOU viruses expose? Ans: 1) Worms typically spread via network connections and do not require user interaction to spread. 2) The viruses exposed insufficient file-scanning antivirus software for e-mail applications and demonstrated that users did not treat suspicious e-mails with appropriate care. 19.5.3 Denial.of.Service (DoS) Attacks A denial-of-service (Dos) attack prevents a system from servicing legitimate requests. In many DoS attacks, unauthorized traffic saturates a network's resources, 19.5 restricting access for legitimate users. Typically, the attack is performed by flooding servers with data packets. Denial-of-service attacks usually require a network of computers to work simultaneously, although some skillful attacks can be achieved with a single machine. Denial-of-service attacks can cause networked computers to crash or disconnect, disrupting service on a Web site or even disabling critical systems such as telecommunications or flight-control centers. Another type of denial-of-service attack targets a network's routing tables. Recall that routing tables provide a view of the network topology and are used by a router to determine where to send data. This type of attack is accomplished by modifying the routing tables, thus maliciously redirecting network activity. For example, the routing tables can be changed to send all incoming data to one address in the network. A similar attack, called a DNS (domain name system) attack, can modify the address to which network traffic for a particular Web site is sent. Such attacks can be used to redirect users of a particular Web site to another, potentially malicious, Web site. These attacks are particularly dangerous if the illegitimate Web site poses as the real one, leading users to disclose sensitive information to the attacker. In a distributed denial-of-service attack, packet flooding comes from multiple computers at once. Such attacks are typically initiated by an individual who has infected several computers with a virus to gain unauthorized access to them to carry out the attack. Distributed denial-of-service attacks can be difficult to stop, because it is not easy to determine which requests on a network are from legitimate users and which are part of the attack. It is also particularly difficult to identify the perpetrator of such attacks, because they are not carried out directly from the attacker's computer. In February 2000, distributed denial-of-service attacks shut down a number of high-traffic Web sites, including Yahoo!, eBay, CNN Interactive and Amazon. In this case, a single user utilized a network of computers to flood the Web sites with traffic that overwhelmed the sites' computers. Although denial-of-service attacks merely shut off access to a Web site and do not affect the victim's data, they can be extremely costly. For example, when eBay's Web site went down for a 24-hour period on August 6,1999, its stock value declined dramatically.41 Who is responsible for viruses and denial-of-service attacks? Most often the responsible parties are referred to as hackers, but really that is a misnomer. In the computer field, hacker refers to an experienced programmer, often one who programs as much for personal enjoyment as for the functionality of the application. The true term for such people is cracker, which is someone who uses a computer maliciously (and often illegally) to break into another system or cause another system to fail. Self Review 1. How are DNS attacks harmful? 2.. Why is it difficult to detect and stop distributed denial-of-service attacks? Ans: 1) Users unaware of malicious activity could submit sensitive information to a cracker. Legitimate Web sites may lose revenue because users are redirected to alternate sites. 2) It may be difficult to distinguish between legitimate and malicious users. Security Attacks 895 896 Security 19.5.4 Software Exploitation Another problem plaguing e-businesses is software exploitation by crackers. Every program on a networked machine should be checked for vulnerabilities. However, with millions of software products available and vulnerabilities discovered daily, this becomes an overwhelming task. One common vulnerability exploitation method is a buffer overflow, in which a program receives input that is larger than its allocated space. A buffer overflow occurs when an application sends more data to a buffer than it can hold. A buffer overflow attack can push the additional data into adjacent buffers, corrupting or overwriting existing data. A well-designed buffer overflow attack can replace executable code in an application's stack to alter its behavior. Buffer overflow attacks may contain malicious code that will then be able to execute with the same access rights as the application it attacked. Depending on the user and application, the attacker may gain access to the entire system. BugTraq (www.securityfocus.com) was created in 1993 to list vulnerabilities, how to exploit them and how to repair them. Self Review 1. Why are buffer overflow attacks dangerous? 2. How can buffer overflow attacks be prevented? Ans: 1) Buffer overflow attacks may modify an application's stack so that an attacker can execute code to gain access to the entire system. 2) An application can rigidly enforce buffer limits when accepting input. Also, the system can implement stack segments that are not executable. 19.5.5 System Penetration The result of many security attacks is system or network penetration. According to a study by the Computer Security Institute (www.gocsi.com), 40 percent of the respondents reported that an outsider had successfully penetrated their systems.42 After an attacker exploits an operating system or the software running on the computer, the system is vulnerable to a number of attacks—ranging from data theft and manipulation to a system crash. A system penetration is a successful breach of computer security by an unauthorized external user.43 Every system penetration is potentially dangerous, although a quick response can usually thwart an intruder's attack before any significant damage is done. Many attacks, such as data theft and Web defacing, rely on a successful system penetration as a foundation. Web defacing is a popular form of attack wherein the crackers illegally obtain access to modify an organization's Web site and change the contents. Web defacing has attracted significant media attention. A notable case occurred in 1996, when Swedish crackers changed the Central Intelligence Agency Web site to read "Central Stupidity Agency." The vandals placed obscenities, political messages, notes to system administrators and links to adult-content sites on the page. Many other popular and large Web sites have been defaced. Defacing Web sites has become over- 19.6 Attack Prevention and Security Solutions 897 whelmingly popular among crackers today, causing archives of attacked sites (with records of more than 15,000 vandalized sites) to close because of the volume of sites that were vandalized daily.44 System penetration often occurs as a result of a Trojan horse, back-door program or an exploited bug in software or the operating system. Allowing external users to access applications via the Web provides another channel for an attacker to penetrate a system.45 Vulnerabilities in common Web server applications, such as Microsoft Internet Information Services (IIS) and Apache HTTP Server, give attackers a well-known route to penetrating a system if administrators fail to apply necessary patches. System penetration can also occur on personal computers through software connected to the Internet, such as Web browsers. The CERT®/CC (Computer Emergency Response Team Coordination Center; www.cert.org) at Carnegie Mellon University's Software Engineering Institute is a federally funded research and development center. CERT/CC responds to reports of viruses and denial-of-service attacks and provides information on network security, including how to determine if a system has been compromised. The site provides detailed incident reports of viruses and denial of service attacks, including descriptions of the incidents, their impact and solutions. The site also reports the vulnerabilities in popular operating systems and software packages. The CERT Security Improvement Modules are excellent tutorials on network security. These modules describe the issues and technologies used to solve network security problems. Self Review 1. List several techniques crackers use to penetrate systems. 2. Compare Web defacing to a DNS attack. Ans: 1) Crackers can issue a Trojan horse back-door program, place an exploited bug in software or the operating system or access applications via an organization's Web site. 2) Web defacing occurs when crackers illegally obtain access to modify an organization's Web site and change its contents. DNS attacks modify the address to which network traffic for a particular Web site is sent, which can redirect users of a particular Web site to another. 19.6 Attack Prevention and Security Solutions The previous section detailed several common attacks against computer security, Though the number of threats to computer security may seem overwhelming, in practice, common sense and diligence can prevent a large number of attacks. To further bolster security, additional hardware and software that specializes in thwarting a variety of attacks can be installed on computers and in networks. This section describes several common security solutions. 19.6.1 Firewall A firewall protects a local area network (LAN) from intruders outside the network. A firewall polices inbound and outbound traffic for the LAN. Firewalls can prohibit all data transmission that is not expressly allowed, or allow all data transmission that 898 Security is not expressly prohibited. The choice between these two models can be determined by the network security administrator; the former provides a high level of security, but might prevent legitimate data transfer. The latter leaves the system more susceptible to attack, but generally does not restrict legitimate network transfer. Each LAN can be connected to the Internet through a gateway, which typically includes a firewall. For years, the most significant security threats originated from employees inside the firewall. Now that businesses rely heavily on access to the Internet, an increasing number of security threats originate outside the firewall—from the hundreds of millions of people connected to the company network via the Internet. 46 There are two primary types of firewalls. A packet-filtering firewall examines

all data sent from outside the LAN and rejects data packets based on predefined rules, such as reject packets that have local network addresses or reject packets from certain addresses or ports. For example, suppose that a hacker from outside the network obtains the address of a computer inside the network and attempts to pass a harmful data packet through the firewall by sending a packet indicating that it was sent from the computer inside the network. In this case, a packet-filtering firewall will reject the data packet because the return address of the inbound packet has clearly been modified. A limitation of packet-filtering firewalls is that they consider only the source of data packets; they do not examine the attached data. As a result, malicious viruses can be installed on an authorized user's computer, allowing an attacker access to the network without that user's knowledge. The goal of an application-level gateway is to protect the network against the data contained in packets. If the message contains a virus, the gateway can block it from being sent to the intended receiver. Installing a firewall is one of the most effective and easiest ways to add security to a small network.47 Often, small companies or home users who are connected to the Internet through permanent connections, such as cable modems, do not employ strong security measures. As a result, their computers are prime targets for crackers to exploit for denial-of-service attacks or information theft. However, it is important for all computers connected to the Internet to contain some degree of security for their systems. Many popular network products, such as routers, provide firewall capabilities and certain operating systems, such as Windows XP, provide software firewalls. In fact, the Windows XP Internet Connection Firewall (ICF) is enabled by default. Section 21.13.3, Internet Connection Firewall, discusses how Windows XP implements its firewall. Numerous other firewall software products are available; several are listed in the Web Resources section at the end of the chapter. Air gap technology is a network security solution that complements the firewall. It secures private data from external traffic accessing the internal network. The air gap separates the internal network from the external network, and the organization decides which information will be made available to external users. Whale Communications created the e-Gap System, which is composed of two computer servers and a memory bank. The memory bank does not run an operating system: therefore hackers cannot take advantage of common operating system weaknesses to access network information. 19.6 Attack Prevention and Security Solutions 899 Air gap technology does not allow outside users to view the network's structure, which prevents hackers from searching the network for weaknesses. The eGap Web Shuttle feature allows safe external access by restricting the system's "back office," where an organization's most sensitive information and IT-based business processes are controlled. Users who wish to access a network hide behind the air gap, where the authentication server is located. Authorized users gain access through a single sign-on capability, allowing them to use one login password to access authorized areas of the network. The e-Gap Secure File Shuttle feature moves files in and out of the network. Each file is inspected behind the air gap. If the file is deemed safe, it is carried by the File Shuttle into the network. 48 Air gap technology is used by e-commerce organizations to allow their clients and partners to access information with transparent security, thus reducing the cost of inventory management. Military, aerospace and government industries, which store highly sensitive information, employ air gap technology. Self Review 1. Is a home user more likely to use a firewall to prohibit all data flow not expressly allowed or to allow all data that is not expressly prohibited? 2. Discuss the difference between packet-filtering firewalls and application gateways. Ans: 1) A home user would be more likely to allow all data that is not expressly prohibited, because the user will most likely want to access a large (and evolving) set of Web sites, located at different IP addresses. 2) A packet-filtering firewall examines all data sent from outside the LAN and rejects data packets based on predefined rules, such as reject packets that have local network addresses or reject packets from certain addresses or ports. An application-level gateway protects the network against the data contained in packets. If the message contains a virus, the gateway can block it from being sent to the intended receiver. 19.6.2 Intrusion Detection Systems (IDSs) Intrusion detection systems (IDSs) monitor networks and application log files, which record information about system behavior, such as the time at which operating system services are requested and the name of the process that requests them. IDSs examine log files to alert system administrators of suspicious application and/ or system behavior, ff an application exhibits erratic or malicious behavior, an IDS can halt the execution of that process.49 Host-based intrusion detection systems monitor system and application log files, which is especially useful for detecting Trojan horses. Network-based intrusion detection software monitors traffic on a network for any unusual patterns that might indicate DoS attacks or access to a network by an unauthorized user. System administrators can then check their log files to determine if there was an intrusion and, if so, track the offender. Intrusion detection products are commercially available from companies such as Cisco (www.cisco.com/warp/public/cc/pd/sqsw/ sqidsz), Hewlett-Packard (www.hp.com/security/home.html) and Symantec 900 Security (www.symantec.com). There is also an open-source network intrusion detection system from Snort (www.snort.org). Intrusion detection via static analysis attempts to detect when applications have been corrupted by a hacker. The static analysis technique assumes that hackers attempt to attack a system using system calls. Under this assumption, the first step in detecting intrusions is to build a model of an application's expected behavior (i.e., a pattern of system calls the application typically generates). The application's pattern of system calls is then monitored as it runs; an attack can be detected if this pattern differs from the static model. 50 The OCTAVESM (Operationally Critical Threat, Asset and Vulnerability Evaluation) method, developed at the Software Engineering Institute at Carnegie Mellon University, evaluates a system's security threats. There are three phases in OCTAVE: building threat profiles, identifying vulnerabilities, and developing security solutions and plans. In the first stage, the organization identifies its important information and assets, then evaluates the levels of security required to protect them. In the second phase, the system is examined for weaknesses that could compromise the valuable data. The third phase is to develop a security strategy as advised by an analysis team of three to five security experts assigned by OCTAVE. This approach is one of the first of its kind, in which the owners of computer systems not only obtain professional analysis, but also participate in prioritizing the protection of crucial information.51 Self Review 1. Name a major drawback to IDSs. 2. Explain the difference between host-based and network-based IDSs. Ans: 1) IDSs cannot detect intrusion with perfect accuracy. Thus, an IDS may prevent an authorized user from performing a legitimate operation and may enable an intrusion to go undetected. 2) Host-based IDSs monitor the current system's log files. Network-based IDSs monitor packets that travel over a network. 19.6.3 Antivirus Software As discussed in Section 19.5.2, viruses and worms have become a menace to businesses and home users alike and have cost businesses billions of dollars.52 The number of reported viruses has increased steadily since the mid-1990s.53 In response, antivirus software has been developed and modified to meet the increasing number and variety of virus attacks on computer systems. Antivirus software attempts to protect a computer from a virus and/or identify and remove viruses on that computer. There are a variety of techniques antivirus software may use to detect and remove viruses from a system; however, none can offer complete protection. Signature-scanning virus detection relies on knowledge about the structure of the computer virus's code. For example, many antivirus programs maintain a list of known viruses and their code. All viruses contain a region called the virus signature that does not change as the virus spreads. In practice, most known virus lists main- 19.6 Attack Prevention and Security Solutions 901 tain a list of virus signatures. In this case, virus detection software scans the computer, comparing file data to virus code. A weakness of known virus lists is that they could become prohibitively large as viruses proliferate. The virus list must be updated regularly to successfully identify emerging viruses. Perhaps the most serious weakness of known virus lists is that they can recognize only viruses that have been previously identified by the list provider. Thus, known virus lists generally do not protect against new and unidentified viruses. A known virus list can be particularly ineffective against variants and polymorphic viruses. A variant is a virus whose code has been modified from its original form, yet still retains its malicious payload. A polymorphic virus changes its code (e.g., via encryption, substitution, insertion, and the like) as it spreads, to evade known virus lists (Fig. 19.8). Although virus signatures improve a virus scanner's ability to detect viruses and their variants, they introduce the possibility of false positive and false negative virus detection. False positive virus alerts incorrectly indicate that a virus is resident in a file, whereas false negatives incorrectly determine that an infected file is clean. These incorrect results become more frequent as the size of a virus signature Figure 19.8 | Polymorphic virus. 902 Security becomes smaller. The problem of preventing false positive and false negative readings when scanning for viruses is a challenging one, as it has been demonstrated that virus signatures as small as two bytes can be created. 54, 55 An alternative to signature scans is heuristic scanning. Viruses are characterized by replication, residence in memory and/or destructive code. Heuristic scans can prevent the spread of viruses by detecting and suspending any program exhibiting this behavior. The primary strength of heuristic scanning is that it can detect viruses that have not yet been identified. Similar to signature scanning, however, heuristic scanning is also susceptible to false reporting. Most antivirus software employs a combination of signature and heuristic scanning. All viruses (with the exception of worms) must modify a file to infect a computer. Consequently, another antivirus technique is to monitor changes to files by checking file consistency. Most consistency checks are implemented as checksums of protected files. In a system that experiences a high volume of file I/O, maintaining a record of file consistency is not a viable option if users expect fast response times. Consequently, many antivirus programs ensure file consistency for a limited set of files (typically the operating system files). However, file consistency scans cannot protect against viruses resident on the computer before antivirus software installation. Some operating systems, such as Windows XP, perform consistency checks on vital system files and replace them if they are altered, to protect system integrity.56 In addition to scanning techniques, antivirus software can be characterized by its program behavior. For example, real-time scanners are resident in memory and actively prevent viruses, whereas other antivirus software must be loaded manually and serves only to identify viruses. Some antivirus software programs prompt the user for action when a virus is detected, whereas others remove viruses without user interaction. 57 Self Review 1. Does signature scanning or heuristic scanning provide better protection against new .. unidentified viruses? 2. Describe several weaknesses of known virus lists. Ans: 1) Heuristic scanning provides better protection against new and unidentified viruses because it can detect new virus behavior even when its signature is not available. 2) A weakness of known virus lists is that they could become prohibitively large as viruses proliferate. The virus list must be updated regularly to successfully identify emerging viruses. Perhaps the most serious weakness of known virus lists is that they can recognize only viruses that have been previously identified by the list provider. Thus, known virus lists generally do not protect against new and unidentified viruses. 19.6.4 Security Patches Operating systems and other software often contain security flaws that are not discovered until after many users have installed them. To address security flaws in a timely manner, the software developers must: 19.6 Attack Prevention and Security Solutions 903 • discover previously unknown flaws, • quickly release patches for the flaws and • establish strong lines of communication with users. To reduce the damage caused by security flaws, developers should address security flaws that have been exploited and actively search and fix those that have not. A code release that addresses a security flaw is often called a security patch. Often, simply releasing a patch for a security flaw is insufficient to improve security. For example, the Slammer worm (Section 19.5.2, Viruses and Worms) exploited a security flaw for which a patch had been released six months earlier. Thus software developers should address security flaws by notifying their users quickly and providing software that facilitates the process of applying security patches. OpenBSD is a BSD UNIX project whose primary goal is to create the most secure (UNIX-compatible) operating system (see the Mini Case Study, OpenBSD). The project includes an audit team that continuously searches for security flaws, a lull-disclosure policy that describes each security flaw in a public forum and a team that releases security patches quickly. Users must subscribe to a mailing list or visit a Web page to learn of new security flaws and download patches. 58 Both the OpenBSD and Apple MacOS X operating systems close all ports and disable all network services by default to improve security (see the Mini Case Study, Macintosh). Microsoft's Trustworthy Computing Initiative has resulted in a steady stream of security patches, called hotfixes, for the Windows line of operating systems. Microsoft offers free software called Automatic Updates that executes in the background of a Windows system to periodically determine if new hotfixes are available. If the Automatic Updates determines that a "critical update" such as a security patch is available, it will display a dialog to prompt the user to download and install the hotfix.59 This software also enables critical updates to be installed without user interaction, which can improve system security. Self Review 1. Why is it important to establish strong lines of communication between software developers and users when releasing security patches? 2. Why is it insufficient to address security flaws only after they are exploited? Ans: 1) Software developers should address security flaws by notifying their users quickly and providing software that facilitates the process of applying security patches, so that users can protect themselves against known security flaws. 2) Some exploit actions of security flaws not detected by developers and could compromise system security and go undetected for days, months or even years. 19.6.5 Secure File Systems Access to system resources, including data stored in a file system, can be regulated by the operating system's access control policy. However, such mechanisms do not nec- 904 Security essarily prevent access to data stored on the hard drive when it is accessed by a different operating system. Consequently, several of today's operating systems support secure file systems that protect sensitive data regardless of how the data is accessed. Windows XP employs the New Technology File System (NTFS), which protects files via access control and encryption (see Section 21.8, File Systems Management). The Encrypting File System (EFS) uses cryptography to protect files and folders in an NTFS file system. EFS uses secret-key and public-key encryption to secure files.60 Each user is assigned a key pair and certificate that are used to ensure that only the user who encrypted the files can access them. File data is lost if the key is lost. EFS is often implemented in multiuser or mobile systems to ensure that protected files would not be accessible to somebody using a stolen or lost machine. Mini Case Study OpenBSD Theo de Raadt, an original member of the NetBSD project, left NetBSD in 1995 to start the OpenBSD project using the open-source NetBSD source code as a base.61, 62 OpenBSD is considered by many to be the most secure OS available. 63, 64, 65 It has been more than four years (as of the time this book was published) since OpenBSD has been cracked while running with default settings. 66, 67 OpenBSD's extraordinary level of security has been achieved primarily by the core team's code-auditing process. Every source file in the system has been analyzed several times by various developers on the core project team—an ongoing process. These audits check for bugs such as classic security holes like buffer overflow opportunities. Any found bugs are posted immediately to the OpenBSD project site and are generally fixed in less thanaday.68,69,70,71 OpenBSD includes several security features, such as built-in support for OpenSSH and OpenSSL (open-source implementations of the SSH and SSL security protocols).72 OpenBSD was also the first operating system to implement IPsec, the IP Security protocol widely used in Virtual Private Networks (VPNs), discussed in Section 19.10.2. It also includes the network authentication protocol Kerberos version five. 73 The OpenBSD group (www.openbsd.org) ensures that its operating system is secure upon installation by shipping OpenBSD w i t h only minimal services enabled and all ports closed (to prevent network attacks). 74, 75 It is interesting to note that Windows Server 2003 follows the same philosophy—a complete reversal from Microsoft's policy in Windows 2000—to enhance security in environments managed by inexperienced administrators. 76 To maintain its secure core, OpenBSD provides a limited set of applications, though the OpenBSD team claims it can run programs written for any of the major UNIX-based systems.77 There is no GUI, making this system an unlikely option for most desktop users but suitable for servers, which must be secure and generally have little direct interaction with users.78 19.6 Attack Prevention and Security Solutions 905 Encryption can be applied to individual files or entire folders; in the latter case, every file inside the folder is encrypted. Applying encryption at the folder level often achieves a higher level of security by preventing programs from creating temporary tiles in plaintext. File encryption in Linux is discussed in Section 20.7, File Systems. Mini Case Study Machintosh cofounder Steve Jobs (now CEO of Apple) visited Xerox's Palo Alto Research Center (PARC).82 During his visit, Jobs saw a demo of the Alto—a personal computer that used a graphical user interface (GUI) and a computer mouse. The Alto, developed at PARC in the 1970s, was designed for personal use, however it was too large and expensive to be practical, therefore it was never sold. all of which worked using a mouse. The Lisa and Macintosh systems were the first personal computers on the market to include a mouse. 83 The Lisa debuted in 1983, but it cost ten thousand dollars and was not successful; Apple discontinued the model t w o years later.84 The Macintosh was announced in 1984 with a famous commercial based on George Orwell's "1984," 8 5 which debuted during the Super Bowl. The Macintosh, which was more space efficient and affordable than the Lisa (it sold for $2,495), became extraordinarily successful.86, 87 Inspired by the Alto, Jobs incorporated GUIs and the computer mouse into both the Apple Lisa and Macintosh operating systems. The Apple GUIs included some features similar to those of the Alto, plus many original features including drag-and-drop file-icon movement, pull-down menus and windows that redraw automatically when uncovered— The first Macintosh operating system was System 1. Although it had GUI capabilities, it was a fairly rudimentary operating system—it had no virtual memory, did not support multiprogramming and had a singlelevel file system.88 These features were updated in later versions; a hierarchical file structure was implemented in System 3, basic In the late 1970s, Apple Computer was working on t w o new computers: the Lisa, a high-end personal computer, and the Macintosh, a more affordable personal computer. 79, 80, 81 In 1979, Apple multiprogramming was added in System 4 and virtual memory was introduced in System 7 which was released in 1990.89 In 1997, after numerous bug fixes and application upgrades, version 7.6 was released and the operating system was renamed Mac OS. 9 0 ' 9 1 Mac OS was completely rebuilt in 2001 for Mac OS X, w i t h the new Darwin kernel incorporating the Mach 3 microkernel and BSD UNIX.92 Darwin is a complete UNIX environment including the X11 windows system as an option. 9 3 Apple also made Darwin an open-source project to make it more robust and stable. 94 Mac OS X has followed OpenBSD's (discussed in the OpenBSD mini case study) lead, improving security by shipping w i t h all ports closed and all network services disabled. 95 Apple also made Mac OS X compatible w i t h the Windows file formats and Windows Exchange mail servers, making it easy for Mac OS X users to integrate into a business environment. 96 906 Security Self Review 1. When are access control mechanisms insufficient to protect file data? 2. What is the primary risk of implementing an encrypted file system? Ans: 1) Access control mechanisms are insufficient to protect file data when the storage device is removed from the computer, because it can be accessed by a system that bypasses the existing access control mechanisms. 2) File data is unrecoverable if the encryption key is lost. 19.6.6 Orange Book Security To evaluate the security features of operating systems, the U.S. Department of Defense (DoD) published a document entitled "Department of Defense Trusted Computer System Evaluation Criteria," also called the Orange Book, in December 1985.97' 98 This document is still used to define levies of security in operating systems. The Orange Book, originally designed for evaluating military systems, classities systems into four levels of security protection —A, B, C and D. The lowest level of security is D and the highest is A. The requirements for each level follow.99, 100 • Level D: Any system that does not meet all the requirements in any other levels. Systems categorized as level D are generally unsecure. • Level C:This level contains two sublevels. Level C1 requires the operating system to separate users and data, which means individuals or groups must log in a username or group name and password to use the system. Private information belonging to an individual or a group is secured to prevent other individuals and groups from reading or modifying it. Early versions of UNIX belong to this level. Level C2 supports only individual login with password, which means users cannot log in with a group name. Authorized individuals can access only certain files and programs. Both C1 and C2 allow individuals to control the access to their files and private information, which means they require only discretionary access control. Most operating systems, such as Windows NT, modern UNIX systems and IBM OS/400101 fall into this category. • Level B: In this level, mandatory access control is required, which means the operating system requires a predefined central permission scheme to determine the permissions assigned to subjects. The creator of an object does not control the permissions for that object. This level contains three sublevels. In addition to the requirements of C2, Bl requires the operating system to contain a predefined central permission scheme and apply sensitivity labels (e.g., "Confidential") on subjects and objects. The access control mechanism must use these sensitivity labels to determine permissions. Operating systems that satisfy Bl requirements include HP-UX BLS 102 , SEVMS 103 and CS/SX104. B2 requires the communication line between the user and the operating system for authentication to be secure, in addition to the requirements from Bl. Operating systems that satisfy B2 requirements include Multics105 and VSLAN 106 . B3 requires all the features present in B2, in addition to implementing protection domains, providing 19.7 Secure Communication 907 secure recovery mechanisms (recovery without compromising protection after a system failure) and monitoring all access to subjects and objects for analysis. The XTS-300 operating system107 satisfies B3 requirements. • Level A: This level contains two sublevels. A1 requires all the features provided by B3, and it requires that the operating system's security is formally verified. An example of an operating system that satisfies the Af requirements is the Boeing MLS LAN 108 . The requirements for A2 are reserved for future use. Self Review 1. Does a system that belongs to category C2 need to satisfy the requirements for level B? 2. How does C2 differ from C1? Ans: 1) No. Level B is higher than level C2, hence level B has more features and requirements. 2) C1 supports both individual and group logins while C2 supports only individual login. 19.7 Secure Communication E-business has profited from the rapidly rising number of consumers that own computers with Internet connections. However, as the number of online transactions increases, so does the volume of sensitive data transmitted over the Internet. Applications that process transactions require secure connections through which sensitive data can be transmitted. Several methods for providing secure transactions have been developed in recent years. In the sections that follow, we describe some of these techniques. There are five fundamental requirements for a successful, secure transaction: • privacy • integrity • authentication • authorization • nonrepudiation The privacy issue is: How do you ensure that the information you transmit over the Internet has not been captured or passed to a third party without your knowledge? The integrity issue is: How do you ensure that the information you send or receive has not been compromised or altered? The authentication issue is: How do the sender and receiver of a message verify their identities to each other? The authorization issue is: How do you manage access to protected resources on the basis of user credentials, which consists of user's identity (e.g., username) and proof of identify (e.g., password)? The nonrepudiation issue is: How do you legally prove that a message was sent or received? Network security must also address the issue of availability: How do we ensure that the network, and the computer systems to which it connects, will operate continuously? In the sections that follow, we discuss 908 Security several implementations of secure communication that derive from encryption techniques discussed in Section 19.2. Self Review 1. For which of the five fundamental requirements for a successful, secure transaction would encryption be useful? 2. Which of the five fundamental requirements deal with user login? Ans: 1) Encryption is useful for all five of the fundamental requirements for a successful, secure transaction. 2) Authorization and authentication. 19.8 Key Agreement Protocols Although public-key algorithms offer flexible and reliable security, they are not efficient for sending large amounts of data. Therefore, public-key algorithms should not be considered replacements for secret-key algorithms. Instead, public-key algorithms are most often employed to exchange secret keys securely. The process by which two parties can exchange keys over an unsecure medium is called a key agreement protocol. The most common key agreement protocol is a digital envelope (Fig. 19.9). The message is encrypted using a secret key (Step 1), and the secret key is then encrypted using public-key encryption (Step 2). The sender attaches the encrypted secret key to the encrypted message and sends the receiver the entire package, or envelope. The sender could also digitally sign the envelope before sending it to prove the sender's identity to the receiver (covered in Section 19.8.2, Digital Signatures). To decrypt the envelope, the receiver first decrypts the secret key using the receiver's private key. Then, the receiver uses the secret key to decrypt the actual message. Because only the receiver can decrypt the encrypted secret key, the sender can be sure that only the intended receiver can read the message. Self Review 1. Why are public-key algorithms employed to exchange secret keys? 2. In the digital envelope protocol, why is it inappropriate to encrypt the entire message using public-key encryption? Ans: 1) Public-key algorithms are efficient for sending small amounts of data, and they prevent attackers from obtaining the secret key while it is transmitted. 2) Public-key encryption is not efficient for sending large amounts of data because it requires significant processing power. 19.8.1 Key Management Maintaining the secrecy of private keys is essential to the maintenance of cryptographic system security. Most security breaches result from poor key management (e.g., the mishandling of private keys, resulting in key theft) rather than cryptanalytic attacks.109 19.8 Key Agreement Protocols 909 Figure 19.9 | Creating a digital envelope. An important aspect of key management is key generation —the process by which keys are created. A malicious third party could try to decrypt a message by using every possible decryption key. Key-generation algorithms are sometimes unintentionally constructed to choose from only a small subset of possible keys. If the subset is too small, then the encrypted data is more susceptible to brute-force attacks. Therefore, it is important to use a key-generation program that can generate a large number of keys as randomly as possible. Key security is improved when key length is large enough that brute-force cracking is computationally infeasible. Self Review 1. Why is it important to use a long key length? 2. Why are most security breaches due to poor key management rather than cryptanalytic attack? Ans: 1) If the total number of decryption keys is small, malicious third parties could generate all possible decryption keys quickly to break the encryption. 2) When long key lengths are used, it is easier to take advantage of poor key management to steal private keys than it is to use a brute-force method such as a cryptanalytic attack. 910 Security 19.8.2 Digital Signatures Digital signatures, the electronic equivalents of written signatures, were developed to address the absence of authentication and integrity in public-key cryptography. A digital signature authenticates the sender's identity and is difficult to forge. To create a digital signature, a sender first applies a hash function to the original plaintext message. Hash functions for secure applications are also typically designed such that it is computationally infeasible to compute a message from its hash value or to generate two messages with the same hash value. A secure hash algorithm is designed so that the probability that two different messages will produce the same message digest is statistically insignificant. In digital signatures, the hash value uniquely identifies a message. If a malicious party changes the message, the hash value also changes, thus enabling the recipient to detect that the message has been altered. Two widely used hashing functions are the Secure Hash Algorithm (SHA-1), developed by National Institute of Standards and Technology (NIST), and the MD5 Message Digest Algorithm. developed by Professor Ronald L. Rivest at MIT. Using SHA-f, the phrase "Buy 100 shares of company X" would produce the hash value D8A9 B6 9F 72 65 OB D5 6D OC 47 00 95 OD FD 31 96 OA FD B5. The hash value is called the message digest. Next, the sender uses the sender's private key to encrypt the message digest. This step creates a digital signature and validates the sender's identity, because only the owner of that private key could encrypt the message. A message that includes the digital signature, hash function and the encrypted message is sent to the receiver. The receiver uses the sender's public key to decipher the digital signature (this establishes the message's authenticity) and reveal the message digest. The receiver then uses his or her own private key to decipher the original message. Finally, the receiver applies the hash function to the original message. If the hash value of the original message matches the message digest included in the signature, the message integrity is confirmed—the message has not been altered in transmission. Digital signatures are significantly more secure than handwritten signatures. A digital signature is based upon the contents of the document, so a digital signature (contrary to a handwritten signature) is different for each document signed. Digital signatures do not provide proof that a message has been sent. Consider the following situation: A contractor sends a company a digitally signed contract, which the contractor later would like to revoke. The contractor could do so by releasing the private key, then claiming that the digitally signed contract was generated by an intruder who stole the contractor's private key. Time stamping, which binds a time and date to a digital document, can help solve the problem of nonrepudiation. For example, suppose the company and the contractor are negotiating a contract. The company requires the contractor to sign the contract digitally and then have the document digitally time stamped by a third party, called a timestamping agency or a digital notary service. The contractor sends the digitally signed contract to the time-stamping agency. The privacy of the message is maintained, because the time-stamping agency sees 19.9 Public-Key Infrastructure, Certificates and Certificate Authorities 911 only the encrypted, digitally signed message. The agency affixes the time and date of receipt to the encrypted, signed message and digitally signs the entire package with the time-stamping agency's private key. The time-stamp cannot be altered by anyone except the time-stamping agency, because no one else possesses the agency's private key. Unless the contractor reports that the private key was compromised before the document was time stamped, the contractor cannot legally prove that the document was signed by an unauthorized third party. The sender could also require the receiver to sign the message digitally and time stamp it as proof of receipt. To learn more about time-stamping, visit www.authentidate.com. The U.S. government's digital authentication standard is called the Digital Signature Algorithm (DSA). Legislation passed in June of 2000 allows digital signatures to be as legally binding as handwritten signatures, which has facilitated many e-business transactions. For the latest news about U.S. government legislation in information security, visit www.itaa.org/infosec. Self Review 1. If digital signatures do not create a message digest, can a malicious third party modify a message's ciphertext? 2. When interpreting messages that are digitally signed, what is the difference between authenticating a message and verifying its integrity? Ans: 1) Although it is possible to modify the ciphertext, it is unlikely that the third party will be able to modify the message in any meaningful way without knowledge of the encryption key. 2) To authenticate a message, the receiver uses the sender's public key to decipher the digital signature. To verify a message's integrity, the receiver checks the hash value of the original message with the message digest included in the signature. 19.9 Public-Key Infrastructure, Certificates and Certificate Authorities A limitation of public-key cryptography is that multiple users might share the same set of keys, making it difficult to establish each party's identity. For example, suppose a customer wishes to place an order with an online merchant. How does the customer know that the Web site being viewed was created by that merchant and not by a third party masquerading as a merchant to steal credit card information? Public Key Infrastructure (PKI) provides a solution to such a problem by integrating public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction. A digital certificate is a digital document that identifies a user and is issued by a certificate authority (CA). A digital certificate includes the name of the subject (the company or individual being certified), the subject's public key, a serial number (which uniquely identifies the certificate), an expiration date, the signature of the trusted certificate authority and any other relevant information. A CA can be a financial institution or any other trusted third party, such as VeriSign. Once issued, the CA makes the digital certificates publicly available in certificate repositories. 912 Security The CA signs the certificate by encrypting either the subject's public key or a hash value of the public key using the CA's own private key, so that a recipient can verify the certificate. The CA must verify every subject's public key. Thus, users must trust the public key of a CA. Usually, each CA is part of a certificate authority hierarchy. This hierarchy can be viewed as a tree structure in which each node relies on its parent for authentication information. The root of the certificate authority hierarchy is the Internet Policy Registration Authority (IPRA). The IPRA signs certificates using the root key, which signs certificates exclusively for policy creation authorities —organizations that set policies for obtaining digital certificates. In turn, policy creation authorities sign digital certificates for CAs. CAs then sign digital certificates for individuals and organizations. The CA is responsible for authentication, so it must check user information carefully before issuing a digital certificate. In one case, human error caused Verisign to issue two digital certificates to an imposter posing as a Microsoft employee. 110 Such an error can be significant; for example, the inappropriately issued certificates can cause users to download malicious code unknowingly onto their machines. VeriSign, Inc., is a leading certificate authority (www.verisi gn. com); other digital certificate vendors are listed in the Web Resources section at the end of the chapter. Digital certificates are created with an expiration date to force users to refresh key pairs periodically. If a private key is compromised before its expiration date, the digital certificate can be cancelled, and the user can obtain a new key pair and digital certificate. Cancelled and revoked certificates are placed on a certificate revocation list (CRL), which is maintained by the certificate authority that issued the certificates. CRLs are analogous to the paper lists of revoked credit card numbers formerly used at the points of sale in retail stores.111 This can substantially lengthen the process of determining the validity of a certificate. An alternative to CRLs is the Online Certificate Status Protocol (OCSP), which validates certificates in real time.112 Many still consider e-commerce unsecure. However, transactions using PKI and digital certificates can be more secure than exchanging private information over public, unencrypted media such as voice phone lines, paying through the mail, or handing a credit card to a sales clerk. In contrast, the key algorithms used in most secure online transactions are nearly impossible to compromise. By some estimates, the key algorithms used in public-key cryptography are so secure that even millions of today's computers working in parallel could not break the codes after a century of dedicated computation. However, as computing power increases, key algorithms considered strong today could be easily compromised in the future. Self Review 1. Describe how CAs ensure the security of their certificates. 2. Why are electronic transactions secured by PKI and digital certificates often more secure than those conducted over the phone or in person? 19.10 Secure Communication Protocols 913 Ans: 1) Each CA is part of a certificate authority hierarchy. This hierarchy can be viewed as a tree structure in which each node relies on its parent for authentication information. Its root is the Internet Policy Registration Authority (IPRA), which signs certificates for policy creation authorities. Policy creation authorities sign digital certificates for CAs. CAs then sign digital certificates for individuals and organizations. 2) The reason is that it is computationally infeasible to compromise the key algorithms used in most secure online transactions. By some estimates, the key algorithms used in public-key cryptography are so secure that even millions of today's computers working in parallel could not break the codes after a century of dedicated computation. By comparison, it is far easier to tap a phone line or write down an individual's credit card number during an in-person transaction. 19.10 Secure Communication Protocols Secure communication protocols have been developed for several layers of the traditional TCP/IP stack. In this section we discuss Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec). We also discuss the protocols that are commonly used to implement secure wireless communication. 19.10.1 Secure Sockets Layer (SSL) The Secure Sockets Layer (SSL) protocol, developed by Netscape Communications, is a nonproprietary protocol that secures communication between two computers on the Internet and the Web. 113 Many e-businesses use SSL for secure online transactions. Support for SSL is included in many Web browsers, such as Mozilla (a stand-alone browser that has been integrated into Netscape Navigator), Microsoft's Internet Explorer and other software products. It operates between the Internet's TCP/IP communications protocol and the application software.114 During typical Internet communication, a process sends and receives messages via a socket. Although TCP guarantees that messages are delivered, it does not determine whether packets have been maliciously altered during transmission. For example, an attacker can change a packet's source and destination addresses without being detected, so fraudulent packets can be disguised as valid ones.115 SSL implements public-key cryptography using the RSA algorithm and digital certificates to authenticate the server in a transaction and to protect private information as it passes over the Internet. SSL transactions do not require client authentication; many servers consider a valid credit card number to be sufficient for authentication in secure purchases. To begin, a client sends a message to a server to open a socket. The server responds and sends its digital certificate to the client for authentication. Using public-key cryptography to communicate securely, the client and server negotiate session keys to continue the transaction. Session keys are secret keys that are used for the duration of that transaction. Once the keys are established, the communication is secured and authenticated using the session keys and digital certificates. Before sending a message over TCP/IP, the SSL protocol breaks the information into blocks, each of which is compressed and encrypted. When the receiver obtains the data through TCP/IP, SSL decrypts the packets, then decompresses and 914 Security assembles the data. SSL is used primarily to secure point-to-point connections — interactions between exactly two computers. 116 SSL can authenticate the server, the client, both or neither. However, in most e-business SSL sessions, only the server is authenticated. The Transport Layer Security (TLS) protocol, designed by the Internet Engineering Task Force (IETF), is the successor to SSL. Conceptually, SSL and TLS are nearly identical—they authenticate using symmetric keys and encrypt data with private keys. The difference between the two protocols is in the implementation of the algorithm and structure of TLS packets.117 Although SSL protects information as it is passed over the Internet, it does not protect private information, such as credit card numbers, once the information is stored on the merchant's server. Credit card information may be decrypted and stored in plaintext on the merchant's server until the order is placed. If the server's security is breached, an unauthorized party can access the information. Self Review 1. Why do many transactions via SSL authenticate only the server? 2. Why is information compressed before encryption under SSL? Ans: 1) The reason is that many servers consider a valid credit card number to be sufficient for authentication in secure purchases. 2) One reason is that compressed data consumes less space, thus requiring less time to transmit over the Internet. Another reason is that compression makes it more difficult for an attacker to interpret the unencrypted message, which provides greater protection against a cryptanalytic attack. 19.10.2 Virtual Private Network (VPMs) and IP Security (IPSec) Most home or office networks are private—computers in the network are connected via communication lines not intended for public use. Organizations can also use the existing infrastructure of the Internet to create Virtual Private Networks (VPNs), which provide secure communication channels over public connections. Because VPNs use the existing Internet infrastructure, they are more economical than wide-area private networks. 118 Encryption enables VPNs to provide the same services and security as private networks. A VPN is created by establishing a secure communication channel, called a tunnel, through which data passes over the Internet. Internet Protocol Security (IPSec) is commonly used to implement a secure tunnel by providing data privacy, integrity and authentication.119 IPSec, developed by the IETF, uses public-key and symmetrickey cryptography to ensure data integrity, authentication and confidentiality. The technology builds upon the Internet Protocol (IP) standard, whose primary security weakness is that IP packets can be intercepted, modified and retransmitted without detection. Unauthorized users can access the network by using a number of well-known techniques, such as IP spoofing—whereby an attacker simulates the IP address of an authorized user or host to obtain unauthorized access to resources. Unlike the SSL protocol, which enables secure, point-to-point connec- 19.10 Secure Communication Protocols 915 tions between two applications, IPSec secures connections among all users of the network on which it is implemented. The Diffie-Hellman and RSA algorithms are commonly used for key exchange in the IPSec protocol, and DES or 3DES are used for secret-key encryption, depending on the system and its encryption needs. [Note: The Diffie-Hellman encryption algorithm is a public-key algorithm.]120 IPSec encrypts packet data, called a payload, then places it in an unencrypted IP packet to establish the tunnel. The receiver discards the IP packet header, then decrypts the payload to access the packet's content.121 VPN security relies on three concepts—user authentication, data encryption and controlled access to the tunnel. 122 To address these three security issues, IPSec is composed of three components. The authentication header (AH) attaches information (such as a checksum) to each packet to verify the identity of the sender and prove that data was not modified in transit. The encapsulating security payload (ESP) encrypts the data using symmetric keys to protect it from eavesdroppers while the IP packet is transmitted across public communication lines. IPSec uses the Internet Key Exchange (IKE) protocol to perform key management, which allows secure key exchange. VPNs are becoming increasingly popular in businesses, but remain difficult to manage. The network layer (i.e., IP) is typically managed by the operating system or hardware, not user applications. So, it can be difficult to establish a VPN because network users must install the proper software or hardware to support IPSec. For more information about IPSec, visit the IPSec Developers Forum at www.ipsec.com and the IETF's IPSec Working Group at w w w . i e t f . o r g / h t m l . c h a r ters/ipsec-charter.html. Self Review 1. Why do companies often choose VPNs over private WANs? 2. Why must the encrypted IP packet be sent inside an unencrypted IP packet? Ans: 1) Because VPNs use the existing infrastructure of the Internet, VPNs are much more economical than WANs. 2) Existing routers must be able to forward the packet. If the entire IP packet were encrypted, routers could not determine its destination. 19.10.3 Wireless Security As wireless devices become increasingly popular, transactions such as stock trading and online banking are commonly transmitted wirelessly. Unlike wired transmissions, which require a third party to physically tap into a network, wireless transmissions enable virtually anyone to access transmitted data. Because wireless devices exhibit limited bandwidth and processing power, high latency and unstable connections, establishing secure wireless communication can be challenging.123 The IEEE 802.11 standard, which defines specifications for wireless LANs, uses the Wired Equivalent Privacy (WEP) protocol to protect wireless communication. WEP provides security by encrypting transmitted data and preventing unau- 916 Security thorized access to the wireless network. To transmit a message, WEP first appends a checksum to the message to enable the receiver to verify message integrity. It then applies the RC4 encryption algorithm, which takes two arguments —a 40-bit secret key (shared between the sender and the receiver) and a random 24-bit value (called the initialization value) —and returns a keystream (a string of bytes). Finally, WEP encodes the message by XORing the keystream and the plain text. Each WEP packet contains the encrypted message and the unencrypted initialization value. Upon receiving a packet, the receiver extracts the initialization value. The receiver then uses the initialization value and the shared secret key to generate the keystream. Finally, it decodes the message by XORing the keystream and the encrypted message. To determine whether the contents of the packet have been modified, the receiver can calculate the checksum and compare it with the checksum appended to the message.124 The security provided by the WEP protocol is inadequate for many environments. For one, the 24-bit initialization value is small, causing WEP to use the same keystream to encode different messages, which can allow a third party to recover plaintext relatively easily. WEP also exhibits poor key management —the secret key can be cracked by brute-force attacks within hours using a personal computer. Most WEP-enabled wireless networks share a single secret key between every client on the network, and the secret key is often used for months and even years. Finally, the checksum computation uses simple arithmetic, enabling attackers to change both the message body and the checksum with relative ease.125, 126 To address these issues, the IEEE and the Wi-Fi Alliance (www.weca.net/ OpenSection/index.asp) developed a specification called Wi-Fi Protected Access (WPA) in 2003. WPA is expected to replace WEP by providing improved data encryption and by enabling user authentication, a feature not supported by WEP WPA also introduces dynamic key encryption, which generates a unique encryption key for each client. Authentication is achieved via an authentication server that stores user credentials.127 Both WEP and WAP are designed for communication using the 802.11 wireless standard. Other wireless technologies —such as Cellular Digital Packet Data (CDPD) for cell phones, PDAs and pagers, Third Generation Global System for Mobile Communications (3GSM) for 3GSM-enabled cell phones, PDAs and pagers, and Time Division Multiple Access (TDMA) for TDMA cell phones —define their own (often proprietary) wireless communication and security protocols.128 Self Review 1. What properties of wireless devices introduce challenges when implementing secure wireless communication? 2. How does WPA address the poor key management in WEP? Ans: 1) Limited bandwidth, limited processing power, high latency and unstable connections. 2) WPA introduces dynamic key encryption, which generates a unique encryption key for each client. 19.11 Steganography 917 19.11 Steganography Steganography is the practice of hiding information within other information, derived from Greek roots meaning "covered writing." Like cryptography, steganography has been used since ancient times. Steganography can be used to hide a piece of information, such as a message or image, within another image, message or other form of multimedia. Consider a simple textual example: Suppose a stock broker's client wishes to perform a transaction via communication over an unsecure channel. The client might send the message "BURIED UNDER YARD." If the client and broker agreed in advance that the message is contained in the first letters of each word, the stock broker extracts the message, "BUY." An increasingly popular application of steganography is digital watermarks for intellectual property protection. Digital steganography exploits unused portions of files encoded using particular formats, such as in images or on removable disks. The insignificant space stores the hidden message, while the digital file maintains its intended semantics.129 A digital watermark can be either visible or invisible. It is usually a company logo, copyright notification or other mark or message that indicates the owner of the document. The owner of a document could show the hidden watermark in a court of law, for example, to prove that the watermarked item was stolen. Digital watermarking could have a substantial impact on e-commerce. Consider the music industry. Music and motion picture distributors are concerned that MP3 (compressed audio) and MPEG-4 (compressed video) technologies facilitate the illegal distribution of copyrighted material. Consequently, many publishers are hesitant to publish content online, as digital content is easy to copy. Also, because CD- and DVD-ROMs store digital information, users are able to copy multimedia files and share them via the Web. Using digital watermarks, music publishers can make indistinguishable changes to a part of a song at a frequency that is not audible to humans, to show that the song was, in fact, copied. Blue Spike's Giovanni™ digital watermarking software uses cryptographic keys to generate and embed steganographic digital watermarks into digital music and images (www.bluespike.com). The watermarks are placed randomly and are undetectable without knowledge of the embedding scheme; thus, they are difficult to identify and remove. Self Review 1. Compare and contrast steganography and cryptography. 2. In grayscale bitmap images, each pixel can be represented by a number between 0 and 255. The number represents a certain shade of gray between 0 (black) and 255 (white). Describe a possible steganography scheme, and explain why it works. Ans: 1) Steganography and cryptography are similar because they both involve sending a message privately. However, with steganography, third parties are unaware that a secret message has been sent, though the message is typically in plaintext. With cryptography, third parties are aware that a secret message is sent, but they cannot decipher the message easily. 2) A 918 Security secret message can be encoded by replacing each of the least significant bits of the picture with one bit of the message. This would not affect the picture much, because each pixel would become at most one shade lighter or darker. In fact, it would be undetectable by most third parties that the picture contained a hidden message. 19.12 Proprietary and Open-Source Security When comparing the merits of open-source and proprietary software, security is an important concern. Supporters of open-source security claim that proprietary solutions rely on security through obscurity. Proprietary solutions assume that because attackers cannot analyze the source code for security flaws, fewer security attacks are possible. Supporters of open-source security claim that proprietary security applications limit the number of collaborative users that can search for security flaws and contribute to the overall security of the application. If the public cannot review the software's source code, certain bugs that are overlooked by developers can be exploited. Developers of proprietary software argue that their software has been suitably tested by their security experts. Open-source security supporters also warn that proprietary security developers are reluctant to publicly disclose security flaws, even when accompanied by appropriate patches, for fear that public disclosure would damage the company's reputation. This may discourage vendors from releasing security patches, thereby reducing their product's security. A primary advantage to open-source applications is interoperability—opensource software tends to implement standards and protocols that many other developers can easily include in their products. Because users can modify the source, they can customize the level of security in their applications. Thus, an administrator can customize the operating system to include the most secure access control policy file system, communication protocol and shell. By contrast, users of proprietary software often are limited to the features provided by the vendor, or must pay additional fees to integrate other features. Another advantage to open-source security is that the application source code is available for extensive testing and debugging by the community at large. For example, the Apache Web server is an open-source application that survived four and half years without a serious vulnerability—an almost unrivaled feat in the field of computer security.130-131 Open-source solutions can provide code that is provably secure before deployment. Proprietary solutions rely on an impressive security track record or recommendations from a small panel of experts to vouch for their security. The primary limitation of security in open-source applications is that opensource software is often distributed without standard default security settings, which can increase the likelihood that security will be compromised due to human error, such as inexperience or negligence. For example, several Linux distributions enable many network services by default, exposing the computer to external attacks. Such services require significant customization and modification to prevent 19.13 Case Study: UNIX Systems Security skilled attackers from compromising the system. Proprietary systems, on the other hand, typically are installed with an appropriate security configuration by default. Unlike most UNIX-based distributions, the open-source OpenBSD operating system claims to install fully secured against external attacks by default. As any user will soon realize, however, significant configuration is required to enable all but the most basic of communication services. It is important to note that proprietary systems can be as secure as opensource systems, despite opposition from the open-source community. Though each method offers relative merits and pitfalls, both require attention to security and timely releases of patches to fix any security flaws that are discovered. Self Review 1. Why might a software development company be reluctant to admit security flaws? 2. How do open-source cryptography software solutions address the fact that the technique they use to generate keys is made public? Ans: 1) Acknowledgement of security flaws can damage a company's reputation. 2) Such software must use a large set of decryption keys to prevent malicious third party brute-force attacks that attempt to compromise security by generating all possible decryption keys. 19.13 Case Study: INUX Systems Security UNIX systems are designed to encourage user interaction, which can make them more difficult to secure.132, 133, 134, 135, 136, 137, 138 UNIX systems are intended to be open systems; their specifications and source code are widely available. The UNIX password file is encrypted. When a user enters a password, it is encrypted and compared to the encrypted password file. Thus, passwords are unrecoverable even by the system administrator. UNIX systems use salting when encrypting passwords.139 The salting is a two-character string randomly selected via a function of the time and the process ID. Twelve bits of the salting then modify the encryption algorithm. Thus, users who choose the same password (by coincidence or intentionally) will have different encrypted passwords (with high likelihood). Some installations modify the password program to prevent users from choosing weak passwords. The password file must be readable by any user because it contains other crucial information (i.e., usernames, user IDs, and the like) that is required by many UNIX tools. For example, because directories employ user IDs to record file ownership, Is (the tool that lists directory contents and file ownership) needs to read the password file to determine usernames from user IDs. If crackers obtain the password file, they could potentially break the password encryption. To address this issue, UNIX protects the password file from crackers by storing information other than the encrypted passwords in the normal password file and storing the encrypted passwords in a shadow password file that can be accessed only by users with root privileges.140 With the UNIX setuid permission feature, a program may be executed by one user using the privileges of another user. This powerful feature has security flaws, 919 920 Security particularly when the resulting privilege is that of the "superuser" (who has access to all files in a UNIX system).141-142 For example, if a regular user is able to execute a shell belonging to the superuser, and for which the setuid bit has been set, then the regular user essentially becomes the superuser. 143 Clearly, setuid should be employed carefully. Users, including those with superuser privileges, should periodically examine their directories to confirm the presence of setuid files and detect any that should not be setuid. A relatively simple means of compromising security in UNIX systems (and other operating systems) is to install a program that prints out the login prompt, copies what the user then types, fakes an invalid login and lets the user try again. The user has unwittingly given away his or her password! One defense is that if you are confident you typed the password correctly the first time, you should log into a different terminal and choose a new password immediately.144 UNIX systems include the crypt command, which allows a user to enter a key and plaintext; ciphertext is output. The transformation can be reversed trivially with the same key. One problem with this is that users tend to use the same key repeatedly; once the key is discovered, all other files encrypted with this key can be read. Users sometimes forget to delete their plaintext files after producing encrypted versions. This makes discovering the key much easier. Often, too many people are given superuser privileges. Restricting superuser privileges can reduce the risk of attackers gaining control of a system due to errors made by inexperienced users. UNIX systems provide a substitute user identity (su) command to enable users to execute shells with a different user's credentials. All su activity should be logged; this command lets any user who types a correct password of another user assume that user's identity, possibly even acquiring superuser privileges. A popular Trojan horse technique is to install a fake su program, which obtains the user's password, e-mails it to the attacker and restores the regular su program. 145 Never allow others to have write permission for your files, especially for your directories; if you do, you're inviting someone to install a Trojan horse. UNIX systems contain a feature called password aging, in which the administrator determines how long passwords are valid; when a password expires, the user receives a message and is asked to enter a new password. 146 - 147 There are several problems with this feature: 1. Users often supply easy-to-crack passwords. 2. The system often prevents a user from resetting to the old (or any other) password for a week, so the user can not strengthen a weak password. 3. Users often switch between only two passwords. Passwords should be changed frequently. A user can keep track of all login dates and times to determine whether an unauthorized user has logged in (which means that his or her password has been learned). Logs of unsuccessful login attempts often store passwords, because users sometimes accidentally type their password when they mean to type their username. Web Resources 921 Some systems disable accounts after a small number of unsuccessful login attempts. This is a defense against the intruder who tries all possible passwords. An intruder who has penetrated the system can use this feature to disable the account or accounts of users, including the system administrator, who might attempt to detect the intrusion. 148 The attacker who temporarily gains superuser privileges can install a trapdoor program with undocumented features. For example, someone with access to source code could rewrite the login program to accept a particular login name and grant this user superuser privileges without even typing a password.149 It is possible for individual users to "grab" the system, thus preventing other users from gaining access. A user could accomplish this by spawning thousands of processes, each of which opens hundreds of files, thus filling all the slots in the openfile table. 150 Installations can guard against this by setting reasonable limits on the number of processes a parent can spawn and the number of files that a process can open at once, but this in turn could hinder legitimate users who need the additional resources. Self Review 1. Why should the setuid feature be used with caution? 2. Discuss how UNIX systems protect against unauthorized logins and how attackers can circumvent and exploit these mechanisms. Ans: 1) When a user executes the program, the user's individual ID or group ID or both are changed to that of the program's owner only while the program executes. If a regular user is able to execute a shell belonging to the superuser, and for which the setuid bit has been set, then the regular user essentially becomes the superuser. 2) To protect against unauthorized logins, UNIX systems use password aging to force users to change passwords frequently, track user logins and disable accounts after a small number of unsuccessful login attempts. Attackers can obtain user passwords using Trojan horses, exploiting the su command and setuid feature and disabling an account (e.g., the system administrator's account) to prevent that user from responding to an intrusion. Web Resources www.nsa.gov/selinux/index.html Lists documents and resources for Security-Enhanced Linux. www.microsoft.com/windowsxp/security Provides links for maintaining security in Windows XP systems. www.sun.com/software/solaris/trustedsolaris/ index.html Provides information about the trusted Solaris operating environment. www.computerworld.com/securitytopics/security/ story/0,10801,69976,00.html Discusses two security flaws in Windows NT and 2000 and how they were fixed. people.freebsd.org/~jkb/howto.html Discusses how to improve FreeBSD system security. www.deter.com/unix Lists links for UNIX-related security papers. www.alw.nih.gov/Security/Docs/networksecurity.html Provides an architectural overview of UNIX network security. www.newsfactor.com/perl/story/7907.html Compares Linux and Windows security. itmanagement.earthweb.com/secu/article.php/641211 Discusses the future of operating systems security. 922 Security www-106.ibm.com/developerworks/security/library/lsec/index.html?dwzone=security Provides links to Linux security issues and how to address these issues. www.ieee-security.org/cipher.html Cipher is an electronic newsletter on security and privacy frorm the Institute of Electrical and Electronics Engineers (IEEE) You can view current and past issues online. www.securemac.com/ Provides security news for Apple systems. www.scmagazine.com SC Magazine has news, product reviews and a conference schedule for security events. www.linuxjournal.com/article.php?sid=1204 Describes how to create a Linux firewall using the TIS Firewall Toolkit. www.rsasecurity.com Homepage of RSA security. www.informit.cotn/isapi/guide~security/seq_id~13/ guide/content.asp Discusses operating system security overview, operating system security weakness. www.ftponline.com/wss/ Homepage of the Windows Server System magazine, which contains various security articles. www.networkcomputing.com/1202/1202fldl.html Provides a tutorial on wireless security. www.cnn.com/SPECIALS/2003/wireless/interactive/ wi reless101/index.html Provides a list of wireless security technologies for computers, PDAs, cell phones and other wireless devices. www.securitysearch.com This is a comprehensive resource for computer security, with thousands of links to products, security companies, tools and more. The site also offers a free weekly newsletter with information about vulnerabilities. www.w3.org/Security/Overview.html The W3C Security Resources site has FAQs, information about W3C security and e-commerce initiatives and links to other security-related Web sites. www.rsasecurity.com/rsalabs/faq This site is an excellent set of FAQs about cryptography from RSA Laboratories, one of the leading makers of public-key cryptosystems. www.authentidate.com Authentidate provides timestamping to verify the security of a file. www.itaa.org/infosec The Information Technology Association of America (ITAA) InfoSec site has information about the latest U.S. government legislation related to information security. www.securitystats.com This computer security site provides statistics on viruses, Web defacements and security spending. www.cdt.org/crypto Visit the Center for Democracy and Technology for U.S. cryptography legislation and policy news. csrc.nist.gov/encryption/aes The official site for the AES includes press releases and a discussion forum. www.pkiforum.org/ The PKI Forum promotes the use of the Public Key Infrastructure. www.Verisign.com VeriSign creates digital IDs for individuals, small businesses and large corporations. Check out its Web site for product information, news and downloads. www.thawte.com Thawte Digital Certificate Services offers SSL, developer and personal certificates. www.belsign.be/ Belsign is the European authority for digital certificates. www.interhack.net/pubs/fwfaq This site provides an extensive list of FAQs on firewalls. www.indentix.com Identix specializes in fingerprinting systems for law enforcement, access control and network security. Using its fingerprint scanners, you can log on to your system, encrypt and decrypt files and lock applications. www.keytronic.com Key Tronic manufactures keyboards with fingerprint recognition systems. www.ietf.org/html.charters/ipsec-charter.html The IPSec Working Group of the Internet Engineering Task Force (IETF) is a resource for technical information related to the IPSec protocol. www.vpnc.org The Virtual Private Network Consortium has VPN standards, white papers, definitions and archives. VPNC also offers compatibility testing with current VPN standards. www.outguess.org Outguess is a freely available steganographic tool. www.weca.net/OpenSection/index.asp Wi-Fi alliance provides information about Wi-Fi and resources for setting up and securing a Wi-Fi network. Summary 923 Summary Computer security addresses the issue of preventing unauthorized access to resources and information maintained by computers. Computer security commonly encompasses guaranteeing the privacy and integrity of sensitive data, restricting the use of computer resources and providing resilience against malicious attempts to incapacitate the system. Protection entails mechanisms that shield resources such as hardware and operating system services from attack. Cryptography focuses on encoding and decoding data so that it can be interpreted only by the intended recipients. Data is transformed by means of a cipher or cryptosystem. Modern cryptosystems rely on algorithms that operate on the individual bits or blocks (a group of bits) of data, rather than letters of the alphabet. Encryption and decryption keys are binary strings of a given length. Symmetric cryptography, also known as secret-key cryptography, uses the same secret key to encrypt and decrypt a message. The sender encrypts a message using the secret key, then sends the encrypted message to the intended recipient, who

decrypts the message using the same secret key. A limitation of secret-key cryptography is that before two parties can communicate securely, they must find a secure way to exchange the secret key. Public-key cryptography solves the problem of securely exchanging symmetric keys. Public-key cryptography is asymmetric in that it employs two inversely related keys: a public key and a private key. The private key is kept secret by its owner, while the public key is freely distributed. If the public key encrypts a message, only the corresponding private key can decrypt it. The most common authentication scheme is simple password protection. The user chooses a password, memorizes it and presents it to the system to gain admission to a resource or system. Password protection introduces several weaknesses to a secure system. Users tend to choose passwords that are easy to remember, such as the name of a spouse or pet. Someone who has obtained personal information about the user might try to log in several times using passwords that are characteristic of the user; several repeated attempts might result in a security breach. Biometrics uses unique personal information—such as fingerprints, eyeball iris scans or face scans—to identify a user. A smart card is often designed to resemble a credit card and can serve many different functions, from authentication to data storage. The most popular smart cards are memory cards and microprocessor cards. Single sign-on systems simplify the authentication process by allowing the user to log in once using a single password. Users authenticated via a single sign-on system can then access multiple applications across multiple computers. It is important to secure single sign-on passwords, because if a password becomes available to hackers, all applications protected by that password can be accessed and attacked. Single sign-on systems are available in forms of workstation login scripts, authentication server scripts and token-based authentication. The key to operating system security is to control access to system resources. The most common access rights are read, write and execute. Techniques that an operating system employ to manage access rights include the access control matrices, access control lists and capability lists. There are several types of security attacks against computer systems, including cryptanalytic attacks, viruses and worms (such as the ILOVEYOU virus and Sapphire/ Slammer worm), denial-of-service attacks (such as a domain name system (DNS) attack), software exploitation (such as buffer overflow) and system penetration (such as Web defacing). Several common security solutions include firewalls (including packet-filtering firewall and application-level gateways), intrusion detection systems (including hostbased intrusion detection and network-based intrusion detection), antivirus software using signature scanning and heuristic scanning, security patches (such as hotfixes) and secure file systems (such as the Encrypting File System). The five fundamental requirements for a successful, secure transaction are privacy, integrity, authentication, authorization and nonrepudiation. The privacy issue deals with ensuring that the information transmitted over the Internet has not been viewed by a third party. The integrity issue deals with ensuring that the information sent or received has not been altered. The authentication issue deals with verifying the identities of the sender and receiver. The authorization issue deals with managing access to protected resources on the basis of user credentials. The nonrepudiation issue deals with ensuring that the network will operate continuously. Public-key algorithms are most often employed to exchange secret keys securely. The process by which two parties can exchange keys over an unsecure medium is called a key agreement protocol. Common key agreement protocols are the digital envelopes and digital signatures (using the SHA-1 and MD5 hash algorithms). 924 Security A limitation of public-key cryptography is that multiple users might share the same set of keys, making it difficult to establish each party's identity. Public Key Infrastructure (PKI) provides a solution by integrating public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction. The Secure Sockets Layer (SSL) protocol is a nonproprietary protocol that secures communication between two computers on the Internet. SSL implements public-key cryptography using the RSA algorithm and digital certificates to authenticate the server in a transaction and to protect private information as it passes over the Internet. SSL transactions do not require client authentication; many servers consider a valid credit card number to be sufficient for authentication in secure purchases. Virtual Private Networks (VPNs) provide secure communications over public connections. Encryption enables VPNs to provide the same services and security as private networks. A VPN is created by establishing a secure communication channel over the Internet. IPSec (Internet Protocol Security), which uses public-key and symmetrickey cryptography to ensure data integrity, authentication and confidentiality, is commonly used to implement a secure tunnel. Wireless devices have limited bandwidth and processing power, high latency and unstable connections, so establishing secure wireless communication can be challenging. The Wired Equivalent Privacy (WEP) protocol protects wireless communication by encrypting transmitted data and preventing unauthorized access to the wireless network. WEP has several drawbacks, which make it to be too weak for many environments. Wi-Fi Protected Access (WPA) addresses these issues by providing improved data encryption and by enabling user authentication, a feature not supported by WEP. A primary advantage to open-source security applications is interoperability—open-source applications tend to implement standards and protocols that many developers include in their products. Another advantage to opensource security is that an application's source code is available for extensive testing and debugging by the community at large. The primary weaknesses of proprietary security are nondisclosure and the fact that the number of collaborative users that can search for security flaws and contribute to the overall security of the application is limited. Proprietary systems, however, can be equally as secure as open-source systems. The UNIX password file is stored in encrypted form. When a user enters a password, it is encrypted and compared to the password file. Thus, passwords are unrecoverable even by the system administrator. With the UNIX setuid permission feature, a program may be executed by one user using the privileges of another user. This powerful feature has security flaws, particularly when the resulting privilege is that of the "superuser" (who has access to all files in a UNIX system). Key Terms 3DES-See Triple DES. Advanced Encryption Standard (AES)—Standard for symmetric encryption that uses Rijndael as the encryption method. AES has replaced the Data Encryption Standard (DES) because AES provides enhanced security. they may be physical objects that correspond to disks processors or main memory or abstract objects that correspond to data structures, processes or services. Subjects are also considered to be objects of the system; one subject may have rights to access another. access control list—List that stores one entry for each access right granted to a subject for an object. An access control list consumes less space than an access control matrix. air gap technology—Network security solution that complements a firewall. It secures private data from external users accessing the internal network. access control matrix—Matrix that lists system's subjects in the rows and the objects to which they require access in the columns. Each cell in the matrix specifies the actions that a subject (defined by the row) can perform on an object (defined by the column). Access control matrices typically are not implemented because they are sparsely populated. antivirus software—Program that attempts to identify, remove and otherwise protect a system from viruses. access right—Defines how various subjects can access various objects. Subjects may be users, processes, programs or other entities. Objects are information-holding entities; application-level gateway—Hardware or software that protects the network against the data contained in packets. If the message contains a virus, the gateway can block it from being sent to the intended receiver. authentication (secure transaction) —One of the five fundamental requirements for a successful, secure transaction Key Authentication deals with how the sender and receiver of a message verify their identities to each other. Terms 925 cipher—Mathematical algorithm for encrypting messages. Also called cryptosystem. authentication header (AH) (IPSec)—Information that verifies the identity of a packet's sender and proves that a packet's data was not modified in transit. ciphertext—Encrypted data. authentication server scripts—Single sign-on implementation that authenticates users via a central server, which establishes connections between the user and the applications the user wishes to access. credential—Combination of user identity (e.g., username) and proof of identify (e.g., password). authorization (secure transaction) —One of the five fundamental requirements for a successful, secure transaction. Authorization deals with how to manage access to protected resources on the basis of user credentials. back-door program—Resident virus that allows an attacker complete, undetected access to the victim's computer resources. biometrics—Technique that uses an individual's physical characteristics, such as fingerprints, eyeball iris scans or face scans, to identify the user. block cipher—Encryption technique that divides a message into fixed-size groups of bits to which an encryption algorithm is applied. boot sector virus—Virus that infects the boot sector of the computer's hard disk, allowing it to load with the operating system and take control of the system. brute-force cracking—Technique to compromise a system simply by attempting all possible passwords or by using every possible decryption key to decrypt a message. buffer overflow—Attack that sends input that is larger than the space allocated for it. If the input is properly coded and the system's stack is executable, buffer overflows can enable an attacker to execute malicious code. capability (access control mechanism)—Token that grants a subject privileges for an object. It is analogous to a ticket the bearer may use to gain access to a sporting event. certificate authority (CA)—Financial institution or other trusted third party, such as VeriSign, that issues digital certificates. certificate authority hierarchy—Chain of certificate authorities, beginning with the root certificate authority, that authenticates certificates and CAs. certificate repositories—Locations where digital certificates are stored. certificate revocation list (CRL)—List of cancelled and revoked certificates. A certificate is cancelled/revoked if a private key is compromised before its expiration date. cracker—Malicious individual that is usually interested in breaking into a system to disable services or steal data. cryptanalytic attack—Technique that attempts to decrypt ciphertext without possession of the decryption key. The most common such attacks are those in which the encryption algorithm is analyzed to find relations between bits of the encryption key and bits of the ciphertext. cryptography—Study of encoding and decoding data so that it can be interpreted only by the intended recipients. cryptosystem—Mathematical algorithm for encrypting messages. Also called a cipher. Data Encryption Standard (DES)—Symmetric encryption algorithm that use a 56-bit key and encrypts data in 64-bit blocks. For many years, DES was the encryption standard set by the U.S. government and the American National Standards Institute (ANSI). However, due to advances in computing power, DES is no longer considered secure— in the late 1990s, specialized DES cracker machines were built that recovered DES keys after a period of several hours. denial-of-service (DoS) attack—Attack that prevents a system from properly servicing legitimate requests. In many DoS attacks, unauthorized traffic saturates a network's resources, restricting access for legitimate users. Typically, the attack is performed by flooding servers with data packets. digital certificate—Digital document that identifies a user or organization and is issued by a certificate authority. A digital certificate includes the name of the subject (the organization or individual being certified), the subject's public key, a serial number (to uniquely identify the certificate), an expiration date, the signature of the trusted certificate authority and any other relevant information. digital envelope—Technique that protects message privacy by sending a package including a message encrypted using a secret key and the secret key encrypted using public-key encryption. digital notary service—See time-stamping agency. digital signature—Electronic equivalent of awritten signature. To create a digital signature, a sender first applies a hash function to the original plaintext message. Next, the sender uses the sender's private key to encrypt the message digest (the hash value). This step creates a digital sig- 926 Security nature and validates the sender's identity, because only the owner of that private key could encrypt the message. Digital Signature Algorithm (DSA)—U.S. government's digital authentication standard. digital watermark—Popular application of steganography that hides information in unused (or rarely used) portions of a file. discretionary access control (DAC)—Access control model in which the creator of an object controls the permissions for that object. distributed denial-of-service attack—Attack that prevents a system from servicing requests properly by initiating packet flooding from separate computers sending traffic in concert. DNS (domain name system) attack—Attack that modifies the address to which network traffic for a particular Web site is sent. Such attacks can be used to redirect users of a particular Web site to another, potentially malicious Web site. Internet Protocol Security (IPSec)—Transport-layer security protocol that provides data privacy, integrity and authentication. intrusion detection system (IDS)—Application that monitors networks and application log files, which record information about system behavior, such as the time at which operating services are requested and the name of the process that requests it. IP spoofing—Attack in which an attacker simulates the IP address of an authorized user or host to obtain unauthorized access to resources. Kerberos—Freely available, open-source authentication and access control protocol developed at MIT that provides protection against internal security attacks. It employs secret-key cryptography to authenticate users in a network and to maintain the integrity and privacy of network communications. key—Input to a cipher to encrypt data; keys are represented by a string of digits. encapsulating security payload (ESP) (IPSec)—Message data encrypted using symmetric-key ciphers to protect the data from eavesdroppers while the IP packet is transmitted across public communication lines. key agreement protocol—Rules that govern key exchange between two parties over an unsecure medium. Encrypting File System (EFS)—NTFS feature that uses cryptography to protect files and folders in Windows XP Professional and Windows 2000. EFS uses secret-key and public-key encryption to secure files. Each user is assigned a key pair and certificate that are used to ensure that only the user that encrypted the files can access them. key generation—Creation of encryption keys. firewall—Software or hardware that protects a local area network from packets sent by malicious users from an external network. hacker—Experienced programmer who often program as much for personal enjoyment as for the functionality of the application. This term is often used when the term cracker is more appropriate. heuristic scanning—Antivirus technique that detects viruses by their program behavior. host-based intrusion detection—IDS that monitors system and application log files. integrity (secure transaction) —One of the five fundamental requirements for a successful, secure transaction. Integrity deals with how to ensure that the information you send or receive has not been compromised or altered. Internet Key Exchange (IKE) (IPSec)—Key-exchange protocol used in IPSec to perform key management, which allows secure key exchange. key distribution center (KDC)—Central authority that shares a different secret key with every user in the network. layered biometric verification (LBV)—Authentication technique that uses multiple measurements of human features (such as face, finger and voice prints) to verify a user's identity. log file—Records information about system behavior, such as the time at which operating services are requested and the name of the process that requests them. logic bomb—Virus that executes its code when a specified condition is met. Lucifer algorithm—Encryption algorithm created by Horst Feistel of IBM, which was chosen as the DES by the United States government and the National Security Agency (NSA) in the 1970s. Macintosh—Apple Computer's PC line that introduced the GUI and mouse to mainstream computer users. Mac OS—Line of operating systems for Apple Macintosh computers first introduced in 1997. mandatory access control (MAC)—Access control model in which policies predefine a central permission scheme by which all subjects and objects are controlled. MD5 Message Digest Algorithm—Hash algorithm developed by Professor Ronald L. Rivest at MIT that is widely used to implement digital signatures. Key Terms 927 message digest—Hash value produced by algorithms such as SHA-1 and MD5 when applied to a message. message integrity—Property indicating whether message has been altered in transmission. National Institute of Standards and Technology (NIST)— Organization that sets cryptographic (and other) standards for the U.S. government. network-based intrusion detection—IDS that monitors traffic on a network for any unusual patterns that might indicate DoS attacks or attempted entry into a network by an unauthorized user. nonrepudiation—Issue that deals with how to prove that a message was sent or received. Online Certificate Status Protocol (OCSP)-Protocol that validates certificates in real time. OpenBSD—BSD UNIX system whose primary goal is security. Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) method—Technique for evaluating security threats of a system developed at the Software Engineering Institute at Carnegie Mellon University. Orange Book—Document published by U.S. Department of Defence (DoD) to establish guidelines for evaluating the security features of operating systems. packet-filtering firewall—Hardware or software that examines all data sent from outside its LAN and rejects data packets based on predefined rules, such as reject packets that have local network addresses or reject packets from certain addresses or ports. password aging—Technique that attempts to improve security by requiring users to change their passwords periodically. password protection—Authentication technique that relies on a user's presenting a username and corresponding password to gain access to a resource or system. password salting—Technique that inserts characters at various positions in the password before encryption to reduce vulnerability to brute-force attacks. payload—Code inside a logic bomb that is executed when a specified condition is met. plaintext—Unencrypted data. Platform for Privacy Preferences (P3P)—Protects the privacy of information submitted to single sign-on and other applications by allowing users to control the personal information that sites collect. policy creation authority—Organization that sets policies for obtaining digital certificates. polymorphic virus—Virus that attempts to evade known virus lists by modifying its code (e.g., via encryption, substitution, insertion, and the like) as it spreads. Pretty Good Privacy (PGP)—Public-key encryption system primarily used to encrypt e-mail messages and files, designed in 1991 by Phillip Zimmermann. privacy (secure transaction)—One of the five fundamental requirements for a successful, secure transaction. Privacy deals with how to ensure that the information transmitted over the Internet has not been captured or passed to a third party without user knowledge. private key—Key in public-key cryptography that should be known only by its owner. If its corresponding public key encrypts a message, only the private key should be able to decrypt it. privilege (access right)—The manner in which a subject can access an object. protection—Mechanism that prevents unauthorized use of resources such as hardware and operating system services. protection domain—Collection of access rights. Each access right in a protection domain is represented as an ordered pair with fields for the object name and applicable privileges. public key—Key in public cryptography that is available to all users that wish to communicate with its owner. If the public key encrypts a message, only the corresponding private key can decrypt it. public-key cryptography—Asymmetric cryptography technique that employs two inversely related keys: a public key and a private key. To transmit a message securely, the sender uses the receiver's public key to encrypt the message. The receiver then decrypts the message using his or her unique private key. Public Key Infrastructure (PKI)—Technique that integrates public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction. resident virus—Virus that, once loaded into memory, executes until the computer is powered down. restricted algorithm—Algorithm that provides security by relying on the sender and receiver to use the same encryption algorithm and maintain its secrecy. random-scanning algorithm—Algorithm that uses pseudorandom numbers to generate a broad distribution of IP addresses. real-time scanner—Software that resides in memory and actively prevents viruses. 928 Security root key—Used by the Internet Policy Registration Authority (IPRA) to sign certificates exclusively for policy creation authorities. Rijndael—Block cipher developed by Dr. Joan Daemen and Dr. Vincent Rijmen of Belgium. The algorithm can be implemented on a variety of processors. role (RBAC) — Represents a set of tasks assigned to a member of an organization. Each role is assigned a set of privileges, which define the objects that users in each role can access. role-based access control (RBAC)—Access control model in which users are assigned roles. RSA—Popular public-key algorithm, which was developed in 1977 by MIT professors Ron Rivest, Adi Shamir and Leonard Adleman. secret-key cryptography—Technique that performs encryption and decryption using the same secret key to encrypt and decrypt a message. The sender encrypts a message using the secret key, then sends the encrypted message to the intended recipient, who decrypts the message using the same secret key. Also called symmetric cryptography. Secure Hash Algorithm (SHA-1)—Popular hash function used to create digital signatures; developed by NIST Secure Sockets Layer (SSL)—Nonproprietary protocol developed by Netscape Communications that secures communication between two computers on the Internet. smart card—Credit card size data store that serves many functions, including authentication and data storage. steganography—Technique that hides information within other information, derived from Latin roots meaning "covered writing." substitution cipher—Encryption technique whereby every occurrence of a given letter is replaced by a different letter. For example, if every "a" were replaced by a "b,' every "b" by a "c," and so on the word "security" would encrypt to "tfdvsjuz." static analysis—Intrusion detection method which attempts to detect when applications have been corrupted by a hacker. system penetration—Successful breach of computer security by an unauthorized external user. symmetric cryptography—See secret-key cryptography. Ticket Granting Service (TGS) (Kerberos) —Server that authenticates client's rights to access specific network services. Ticket-Granting Ticket (TGT) (Kerberos)—Ticket returned by Kerberos's authentication server. It is encrypted with the client's secret key that is shared with the authentication server. The client sends the decrypted TGT to the Ticket Granting Service to request a service ticket. time bomb—Virus that is activated when the clock on the computer matches a certain time or date. Security mechanism—Method by which the system implements its security policy. In many systems, the policy changes over time, but the mechanism remains unchanged. timestamping (nonrepudiation)—Technique that binds a time and date to a digital document, which helps solve the problem of nonrepudiation. Security model—Entity that defines a system's subjects, objects and privileges. time-stamping agency—Organization that digitally time stamps a document that has been digitally signed. token (in token-based authentication)—Unique identifier for authentication. Security patch—Code that addresses a security flaw. Security policy—Rules that govern access to system resources. Service ticket (Kerberos)—Ticket that authorizes the client's access to specific network services. Session key—Secret key that is used for the duration of a transaction (e.g., a customer's buying merchandise from an online store). Shadow password file (UNIX)—Protects the password file from crackers by storing information other than the encrypted passwords in the normal password file and storing the encrypted passwords in the shadow password file that can be accessed only by users with root privileges. Signature-scanning virus detection—Antivirus technique that relies on knowledge of virus code. Single sign-on— Simplifies the authentication process by allowing the user to log in once, using a single password. token-based authentication—Authentication technique that issues a token unique to each session, enabling users to access specific applications. transient virus—Virus that attaches itself to a particular computer program. The virus is activated when the program is run and deactivated when the program is terminated. transposition cipher—Encryption technique whereby the ordering of the letters is shifted. For example, if every other letter, starting with "s," in the word "security" creates the first word in the ciphertext and the remaining letters create the second word in the ciphertext, the word "security" would encrypt to "scrt euiy." Triple DES—Variant of DES that can be thought of as three DES systems in series, each with a different secret key that operates on a block. Also called 3DES. Exercises 929 Trojan horse—Malicious program that hides within a trusted program or simulates the identity of a legitimate program or feature, while actually causing damage to the computer or network when the program is executed. virus signature—Segment of code that does not vary between virus generations. two-factor authentication—Authentication technique that employs two means to authenticate the user, such as biometrics or a smart card used in combination with a password. Wi-Fi Protected Access (WPA)—Wireless security protocol intended to replace WEP by providing improved data encryption and by enabling user authentication. Web defacing—Attack that maliciously modifies an organization's Web site. variant—Virus whose code has been modified from its original form, yet still retains its malicious payload. Wired Equivalent Privacy (WEP)—Wireless security protocol that encrypts transmitted data and prevents unauthorized access to the wireless network. Virtual Private Network (VPN)—Technique that securely connects remote users to a private network using public communication lines. VPNs are often implemented using IPSec. workstation login scripts—Simple form of single sign-on in which users log in at their workstations, then choose applications from a menu. virus—Executable code (often sent as an attachment to an email message or hidden in files such as audio clips, video clips and games) that attaches to or overwrites other files to replicate itself, often harming the system on which it resides. worm—Executable code that spreads and infects files over a network. Worms rarely require any user action to propagate, nor do they need to be attached to another program or file to spread. Exercises 19.1 Why is a precise statement of security requirements critical to the determination of whether a given system is secure? 19.2 Sharing and protection are conflicting goals. Give three significant examples of sharing supported by operating systems. For each, explain what protection mechanisms are necessary to control the sharing. 19.3 Give several reasons why simple password protection is the most common authentication scheme in use today. Discuss the weaknesses inherent in password protection schemes. 19.4 An operating systems security expert has proposed the following economical way to implement a reasonable level of security: Simply tell everyone who works at an installation that the operating system contains the latest and greatest in security mechanisms. Do not actually build the mechanisms, the expert says, just tell everyone that the mechanisms are in place. Who might such a scheme deter from attempting a security violation? Who would not likely be deterred? Discuss the advantages and disadvantages of the scheme. In what types of installations might this scheme be useful? Suggest a simple modification to the scheme that would make it far more effective, yet still far more economical than a comprehensive security program. 19.5 What is the principle of least privilege? Give several reasons why it makes sense. Why is it necessary to keep protection domains small to effect the least-privilege approach to security? Why are capabilities particularly useful in achieving small protection domains? 19.6 How does a capabilities list differ from an access control list? 19.7 Why is an understanding of cryptography important to operating systems designers? List several areas within operating systems in which the use of cryptography would greatly improve security. Why might a designer choose not to include encryption in all facets of the operating system? 19.8 Why is it desirable to incorporate certain operating systems security functions directly into hardware? Why is it useful to microprogram certain security functions? 19.9 Give brief definitions of each of the following terms. a. cryptography b. Data Encryption Standard (DES) c. privacy problem d. integrity problem e. problem of nonrepudiation f. plaintext g. ciphertext h. unsecure channel i. eavesdropper j . cryptanalysis 930 Security 19.10 Give brief definitions of each of the following terms. a. public-key system 19.14 Explain how cryptography is useful in each of the following. b. public key a. protecting a system's master list of passwords c. private key b. protecting stored files d. digital signature c. protecting vulnerable transmissions in computer networks 19.11 Why are denial-of-service attacks of such great concern to operating systems designers? List a few types of denial-ofservice attacks. Why is it difficult to detect a distributed denialof-service attack on a server? 19.15 The UNIX systems administrator cannot determine the password of a user who has lost his or her password. Why? How can that user regain access to the computer? 19.12 What is a log file? What information might an operating system deposit in a log file? How does a log act as a deterrent to those who would commit a security violation? 19.16 How could a UNIX system user seize control of a system, thus locking out all other users? What defense may be employed against this technique? 19.13 Explain how public-key cryptography systems provide an effective means of implementing digital signatures. 19.17 Why do antivirus software companies and high-security installations hire convicted hackers? Suggested Projects 19.18 At the moment, it appears that our computing systems are easy targets for penetration. Enumerate the kinds of weaknesses that are prevalent in today's systems. Suggest how to correct these weaknesses. 19.19 Design a penetration study for a large computer system with which you are familiar. Adhere to the following: a. Perform the study only with the full cooperation of the computer center administrators and personnel. b. Arrange to have ready access to all available system documentation and software source listings. f. You may access the system only through conventional mechanisms available to nonprivileged users. g. Your attacks must be software and/or firmware oriented. 19.20 Prepare a research paper on the RSA algorithm. What mathematical principles aid the RSA algorithm in being difficult to crack? 19.21 Prepare a research paper on how antivirus software provides security while being minimally invasive. c. Agree that any flaws you discover will not be made public until they have been corrected. 19.22 Prepare a research paper on the implementation of PGP Explain why the name "Pretty Good Privacy" is appropriate. d. Prepare a detailed report summarizing your findings and suggesting how to correct the flaws. 19.23 Prepare a research paper on the most costly viruses and worms. How did these worms or viruses spread? What flaws did they exploit? e. Your primary goals are to achieve i. total system penetration ii. denial of access to legitimate system users iii. the crashing of the system. 19.24 Prepare a research paper on how IP spoofing is performed. 19.25 Discuss the ethical and legal issues surrounding the practice of hacking. Suggested Simulation 19.26 Create an application to encode and decode secret messages within bitmap pictures. When encoding, modify the least significant bit in each byte to store information about the mes- sage. To decode, just reassemble the message using the least significant bit of each byte. Recommended Reading Computer security has been researched extensively in academic and commercial environments. Most topics in the field are constantly evolving. Although the problems of cryptography and access control are well characterized, techniques that Works Cited 931 address security threats (such as software exploitation and network attacks) must continually evolve as new hardware, software and protocols with different vulnerabilities emerge. A survey of cryptography and the related problem of computability is provided in Jorg Rothe's "Some Facets of Complexity Theory and Cryptography: A Five-Lecture Tutorial."151 The problem of access control is actively discussed in the literature; a survey of access control can be found in Sandhu and Samarati's "Access Control: Principles and Practice."152 and publications have emerged since the 1990s. Commercial trade journals include Information Security, SC (Secure Computing) Magazine and Access Control and Security Systems. Academic security journals include the Journal of Computer Security and the ACM Transactions on Information and System Security (TISSEC). Dieter Gollmann provides a thorough discussion of computer security theory.153 The bibliography for this chapter is located on our Web site at www.deitel.com/ books/os3e/Bibliography.pdf. Due to the importance of security to e-business and the Internet revolution, a large number of security trade journals Works Cited 1. Gates, B., "Bill Gates: Trustworthy Computing," January 17,2002, . 2. McFarland, M., "Ethical Issues of Computer Use," Syllabus for course Mc690, Boston College, Chestnut Hill, MA, Spring 1989. 16. Krieger, D., and W. E. Miles, "Innerview: Leonard Adleman," Networker, Vol. 7, No. 1, September 1996, . 3. Parnas, D. L., "Software Aspects of Strategic Defense Systems," Communications of the ACM, Vol. 28, No. 12, December 1985, pp. 1326-1335. 17. ACM, "ACM: A. M. Turing Award," April 14, . 4. "Cryptography—Caesar Cipher," <http://www.trincoll.edu/ depts/cpsc/cryptography/caesar.html>. 5. "Methods of Transposition," <http://home.ecn.ab.ca/~jsa vard/crypto/pp0102.htm>. 6. RSA Laboratories, "RSA Laboratories' Frequently Asked Questions About Today's Cryptography," Version 4.1, 2000, RSA Security Inc., . 7. Cherowitzo, B., "Math 5410 Data Encryption Standard (DES)," February 6, 2002, . 8. Dworkin, M., "Advanced Encryption Standard (AES) Fact Sheet," March 5, 2001, <csrc.nist.gov/CryptoToolkit/aes/ round2/aesfact.html>. 9. Rijmen, V., "The Rijndael Page," May 20, 2003, ro. McNett, D., "RC5-64 HAS BEEN SOLVED!" September 27, 2002, . 11. RSA Laboratories, "RSA-Based Cryptographic Schemes," 2003, . 12. Ytteborg, S. S., "Overview of PGP," . 13. RSA Security Inc., "RSA Security at a Glance," 2003, . 14. Rivest, R. L.; A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, Vol. 21, No. 2, February 1978, pp. 120-126. 15. Bass, T. A., "Gene Genie," Wired, August 1995, . 2003, 18. RSA Security Inc., "The Authentication Scorecard," 2003, . 19. Lamport, L., "Password Authentication with Insecure Communication," Communications of the ACM, Vol. 24, No. 11, November 1981, pp. 770-772. 20. National Infrastructure Protection Center, "Password Protection 101," May 9, 2002, . 21. Deckmyn, D., "Companies Push New Approaches to Authentication," Computerworld, May 15,2000, p. 6. 22. Keyware, "Keyware Launches Its New Biometric and Centralized Authentication Products," January 2001, . 23. Vijayan, X, "Biometrics Meet Wireless Internet," Computerworld, July 17,2000, p. 14. 24. Smart Card Forum, "What's So Smart About Smart Cards?" 2003, . 25. Gaudin, S., "The Enemy Within," Network World, May 8, 2000, pp. 122-126. 26. Needham, R. M. and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, Vol. 21, No. 12, December 1978, pp.993- 999. 27. Steiner, X; C. Neuman and X Schiller, "Kerberos: An authentication service for open network systems," In Proceedings of the Winter 1988 USENIX, February 1988, pp. 191-202. 28. Steiner, X; C. Neuman and J. Schiller, "Kerberos: An authentication service for open network systems," In Proceedings of the Winter 1988 USENIX, February 1988, pp. 191-202. 932 Security 29. Trickey, F., "Secure Single Sign-On: Fantasy or Reality," Computer Security Institute, 1997. 30. Musthaler, L., "The Holy Grail of Single Sign-On," Network World, January 28,2002, p. 47. 31. Press Release, "Novell Takes the Market Leader in Single SignOn and Makes It Better," . 32. "Protection and Security," <http://cs.nmu.edu/~randy/C1asses/CS426/protection_and_security.html>. 33. Lampson, B., "Protection," ACM Operating Systems Review 8, January 1,1974, pp. 18-24. 34. Sandhu, R., and E. Coyne, "Role-Based Access Control Models," IEEE Computer, February 1996, p. 39. 35. Sandhu, R., and E. Coyne, "Role-Based Access Control Models," IEEE Computer, February 1996, p. 40. 36. "Handbook of Information Security Management: Access Control," . 37. Chander, A.; D Dean; and J. C. Mitchell, "A State-Transition Model of Trust Management and Access Control," IEEE Computer, in Proceedings of the 14th Computer Security Foundations Workshop (CSFW), June 2001, pp. 27-43. 38. Lampson, B., "Protection," Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems, 1971, pp. 437443. 39. Linden, T A., "Operating System Structures to Support Security and Reliable Software," ACM Computing Surveys (CSUR), Vol. 8, No. 6, December 1976. 40. Moore, D., et al., "The Spread of the Sapphire/Slammer Worm," CAIDA, February 3, 2003, . 41. "Securing B2B," Global Technology Business, July 2000, pp. 50-51. 42. Computer Security Institute, "Cyber Crime Bleeds U.S. Corporations, Survey Shows; Financial Losses from Attacks Climb for Third Year in a Row," April 7, 2002, . 43. Connolly, P., "IT Outlook Appears Gloomy," InfoWorld, November 16, 2001, <www.infoworld.com/articles/tc/xml/01/ll/ 19/011119tcstate.xrrfl>. 44. Bridis, T, "U.S. Archive of Hacker Attacks to Close Because It Is Too Busy," The Wall Street Journal, May 24,2001, p. BIO. 45. Andress, M., "Securing the Back End," InfoWorld, April 5, 2002, . 46. Marshland, R., "Hidden Cost of Technology," Financial Times, June 2,2000, p. 5. 47. Spangler, T, "Home Is Where the Hack Is," Week, April 10,2000, pp. 28-34. 48. Whale Communications, "Air Gap Architecture," 2003, . 49. Azim, O., and P. Kolwalkar, "Network Intrusion Monitoring," Business Security Advisor, March/April 2001, pp. 16-19, . 50. Wagner, D., and D. Dean, "Intrusion Detection via Static Analysis," IEEE Symposium on Security and Privacy, May 2001. 51. Alberts, C, and A. Dorofee, "OCTAVE Information Security Risk Evaluation," January 30, 2001, . 52. Delio, M, "Find the Cost of (Virus) Freedom," Wired News, January 14, 2002, . 53. Bridwell, L. M., and P. Tippett, "ICSA Labs Seventh Annual Computer Virus Prevalence Survey 2001," ICSA Labs, 2001. 54. Helenius, M., "A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities," Academic dissertation. Dept. of Computer and Information Sciences, University of Tampere, Finland, May 2002. 55. Solomon, A., and T. Kay, Dr. Solomon's PC Anti-virus Book. Butterworth-Heinemann, 1994, pp. 12-18. 56. University of Reading, "A Guide to Microsoft Windows XP." October 24, 2003, . 57. Helenius, M., "A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities," Academic dissertation. Dept. of Computer and Information Sciences, University of Tampere, Finland, May 2002. 58. OpenBSD Team, "OpenBSD Security," September 17, 2003, . 59. Microsoft Corporation, "Top 10 Reasons to Get Windows XP for Home PC Security," April 16, 2003, . 60. Microsoft Corporation, "What's New in Security for Windov>> XP Professional and Windows XP Home Edition," July 2001. . 61. Security Electronics Magazine, "OpenBSD: Secure by Default." January 2002, . 62. Howard, X, "Daemon News: The BSD Family Tree," April 2001. . 63. Security Electronics Magazine, "OpenBSD: Secure by Default." January 2002, . 64. Jorm, D, "An Overview of OpenBSD Security," August 8, 2000. . 65. Howard, J., "Daemon News: The BSD Family Tree," April 2001. . 66. Howard, X, "Daemon News: The BSD Family Tree," April 2001. . 67. Security Electronics Magazine, "OpenBSD: Secure by Default." January 2002, . 68. OpenBSD Team, "OpenBSD Security," September 17, 2003. . 69. Security Electronics Magazine, "OpenBSD: Secure by Default." January 2002, . 70. Jorm, D., "An Overview of OpenBSD Security," August 8, 2000. . Works Cited 933 71. Howard, X, "Daemon News: The BSD Family Tree," April 2001, <www.daemonnews.org/200104/bsd_ family.html >. 95. Apple Computer, Inc., "Apple —Mac OS X—Features —Security," 2003,. 72. OpenBSD Team, "Cryptography in OpenBSD," September 25, 2003, . 96. Apple Computer, Inc., "Apple—Mac OS X—Features—Windows," 2003,. 73. OpenBSD Team, "OpenBSD Security," September 17, 2003, . 97. Dynamoo, "Orange Book FAQ," June 2002, . 74. Howard, X, "Daemon News: The BSD Family Tree," April 2001, . 98. The Jargon Dictionary, "Orange Book," <info.astrian.net/ jargon/terms/o/Orange_Book.html>. 75. OpenBSD Team, "OpenBSD Security," September 17, 2003, . 99. Dynamoo, "Orange Book Summary," . 76. Mullen, T, "Windows Server 2003 - Secure by Default," April 27, 2003, . 100. Dynamoo, "Orange Book—Full Text," . 77. OpenBSD Team, "OpenBSD," September 27, 2003, . 101. "EPL Entry CSC-EPL-95/006.D," January 2000, . 78. Security Electronics Magazine, "OpenBSD: Secure by Default," January 2002, . 102. "EPL Entry CSC-EPL-93/008.A," July 1998, . 79. Sanford, G., "Lisa/Lisa 2/Mac XL," . 103. "EPL Entry CSC-EPL-93/003.A," July 1998, . 80. Sanford, G., "Macintosh 128k," . 104. "EPL Entry CSC-EPL-93/006.A," July 1998, . 81. Trotot, X, "Mac™ OS History," <perso.club-internet.fr/ jctrotot/Perso/History.html>. 105. Corbato, F. J., and V. A. Vyssotsky, "Introduction and Overview of the Multics System," . 82. Horn, B., "On Xerox, Apple, and Progress," . 106. "EPL Entry CSC-EPL-90/001.A," July 1998, . 83. Horn, B., "On Xerox, Apple, and Progress," . 107. "EPL Entry CSC-EPL-92/003.E," June 2000, . 84. Sanford, G, "Lisa/Lisa 2/Mac XL," . 108. "EPL Entry CSC-EPL-94/006," July 1998, ncsc.mil/tpep/epl/entries/CSC-EPL-94-006.html>. 85. Apple Computer, "1984 Commercial," . 109. RSA Laboratories, "4.1.3.14 How Are Certifying Authorities Susceptible to Attack?" April 16, 2003, . 86. Sanford, G., "Macintosh 128k," . 87. Sanford, G, "Company History: 1985-1993," . 88. University of Utah, "Mac OS History," September 23, 2003, . 89. University of Utah, "Mac OS History," September 23, 2003, . 90. Trotot, X, "Mac™ OS History," <perso.club-internet.fr/ jctrotot/Perso/History.html>. March 2003, June 2002, 1 10 137 19902 2 20473 20484 20485 20489 20505 20507 20522 20525 20527 20529 20534 root> ls /proc 20535 20656 20538 20657 20539 20658 20540 20696 20572 20697 20576 20750 20577 3 20578 4 20579 469 20581 5 20583 536 20586 541 20587 561 20591 589 20621 6 20624 7 751 792 8 811 829 883 9 919 940 960 961 962 963 964 965 966 978 acpi asound buddyinfo bus cmdiine cpuinfo crypto devices dma dri driver execdomains filesystems fs ide interrupts iomem ioports irq kcore kmsg ksyms loadavg locks meminfo misc modules mounts mtrr net partitions Figure 20.18 | Sample contents of the /proc directory. pci self slabinfo stat swaps sys sysvipc tty uptime version vmstat 20.8 Input/Output Management 983 uptime (which displays the amount of time the system has been running). Data provided by procfs is particularly useful for driver developers and system administrators who require detailed information about system usage. In this section, we limit our discussion to the implementation of procfs. A detailed explanation of the /proc directory's contents can be found in the Linux source code under Documentation/ filesystems/proc.txt. Procfs is a file system that exists only in main memory. The contents of files in the proc file system are not stored persistently on any physical medium—procfs files provide users an access point to kernel information, which is generated on demand. When a file or directory is registered with the proc file system, a proc directory entry is created. Proc directory entries, unlike VFS directory entries, allow each directory to implement its own read function. This enables a proc directory to generate its contents each time the directory entry is accessed. When a process accesses a particular procfs file, the kernel calls the corresponding file operation specified by the file. These functions allow each file to respond differently to read and write operations.126 The kernel creates many procfs entries by default. Additional files and directories can be created using loadable kernel modules. When a user attempts to read data from a procfs file, the VFS calls the procfs read function, which accesses a proc directory entry. To complete a read request, procfs calls the read function defined for the requested file. Procfs read functions typically gather status information from a resource, such as the amount of time the system has been running. Once information has been retrieved by a read function, procfs passes the output to the process that requested it.127 Procfs files can be used to send data to the kernel. Some system variables, such as the network host name of a machine, can be modified at runtime by writing to procfs files. When a process writes to a procfs file, the data provided by the process may be used to update the appropriate kernel data structures.128 20.8 Input/Output Management This section explains how the kernel accesses system devices using the I/O interface. The kernel abstracts the details of the hardware in a system, providing a common interface for I/O system calls. The kernel groups devices into classes; members of each device class perform similar functions. This allows the kernel to address the performance needs of certain devices (or classes of devices) individually. 20.8.1 Device Drivers Support for devices such as graphics cards, printers, keyboards and other such hardware is a necessary part of any operating system. A device driver is the software interface between system calls and a hardware device. Independent Linux developers, not device manufacturers, have written most of the drivers that operate devices commonly found in Linux systems. This generally limits the number of devices that are compatible with the Linux operating system. As the popularity of Linux increases, so does the number of vendors that ship Linux drivers with their devices. 984 Case Study: Linux Typically, device drivers are implemented as loadable kernel modules. Drivers implemented as modules can be loaded and unloaded as they are needed, avoiding the need to have them permanently loaded in the kernel. 129 Most devices in a system are represented by device special files. A device special file is an entry in the /dev directory that provides access to a particular device. Each file in the /dev directory corresponds to a block device or a character device.130 A list of block and character device drivers that are currently loaded on a particular system can be found in the file /proc/devices (Fig. 20.19). Devices that perform similar functions are grouped into device classes. For example, each brand of mouse that connects to the computer may belong to the input device class. To uniquely identify devices in the system, the kernel assigns each device a 32-bit device identification number. Device drivers identify their devices using a major identification number and a minor identification number. Major and minor identification numbers for all devices supported by Linux are located in the Linux Device List, which is publicly available online.131 Driver developers must use the numbers allocated to devices in the Linux Device List to ensure that devices in a system are properly identified. Devices that are assigned the same major identification number are controlled by the same driver. Minor identification numbers allow the system to distinguish root> cat /proc/devices Character devices: Block devices: root> Figure 20.19 | /proc/devices file contents.132 20.8 Input/Output Management 985 individual devices that are assigned the same major identification number (i.e., belong to the same device class).133 For example, a hard disk is assigned a major identification number, and each partition on the hard disk is assigned a device minor number. Device special files are accessed via the virtual file system. System calls pass to the VFS, which in turn issues calls to device drivers. Figure 20.20 illustrates the interaction between system calls, the VFS, device drivers and devices. Drivers implement generic virtual file system functions so that processes may access /dev files using standard library calls. For example, standard library calls for accessing files (such as printing to a file using the standard C library function fprintf) are implemented on top of lower-level system calls (such as read and write). 134 Individual device characteristics determine the drivers and their corresponding system calls necessary to support the device. Most devices that Linux supports belong to three primary categories: character devices, block devices and network devices. Each class of device requires its corresponding device drivers to implement a set of functions common to the class. For example, a character device driver must implement a write function to transfer data to its device.135 Further, if a device is attached to a kernel subsystem (e.g., the SCSI subsystem), the device's drivers must interface with the subsystem to control the device. For example, a SCSI device driver passes I/O requests to the SCSI subsystem, which then interacts directly with devices attached to the SCSI interface.136 Subsystem interfaces exist to reduce redundant code; for example, each SCSI device driver need only provide access to a particular Figure 20.20 | I/O interface layers. SCSI device, not to the SCSI controller to which it is attached. The SCSI subsystem provides access to common SCSI components, such as the SCSI controller. Most drivers implement common file operations such as read, write and seek. These operations allow drivers to transfer data to and from devices, but do not allow them to issue hardware-specific commands. To support tasks such as ejecting a CD-ROM tray or retrieving status information from a printer, Linux provides the ioctl system call, ioctl allows developers to send control messages to am device in the /dev directory. The kernel defines several default control messages and allows a driver to implement its own messages for hardware-specific operations.137 The set of messages supported by a device driver is dependent on the driver's device and implementation. A character device transmits data as a stream of bytes. Devices that fall under this category include printers, consoles, mice, keyboards and modems. Because they transfer data as streams of bytes, most character devices support only sequential access to data. 138 Most character device drivers implement basic operations such as opening, closing, reading from and writing to a character device. Each device in the system is represented by a device_struct structure that contains the driver name and a pointer to the driver's file_operations structure, which maintains the operations supported by the device driver. To initialize a character device, a device driver must register its operations with the virtual file system, which appends a device_struct structure to the array of registered drivers stored in chrdevs. 139 Figure 20.21 Figure 20.21 | chrdevs vector. 20.8 Input/Output Management 987 describes the contents of vector chrdevs. Each entry in chrdevs corresponds to a device driver major identification number. For example, the fifth entry in chrdevs is the device_struct for the driver with major number five.140 When a system call accesses a device special file, the VFS calls the appropriate function in the device's file_operations structure. This structure includes functions that perform read, write and other operations on the device. The inode representing the file stores a device special file's file_operations structure.141 After a device's file operations have been loaded into its inode, the VFS will use those operations whenever system calls access the device. The system can access this inode until a system call closes the device special file. However, once a system call closes the file, the inode must be recreated and initialized the next time the file is opened. 142 20.8.3 Block Device I/O Unlike character devices, block devices allow data stored in fixed-sized blocks of bytes to be accessed at any time, regardless of where those blocks are stored on the device. To facilitate nonsequential (i.e., random) access to a large amount of data (e.g., a file on a hard drive), the kernel must employ a more sophisticated system for handling block device I/O than it does for handling character device I/O. For example, the kernel provides algorithms that attempt to optimize moving-head storage (i.e., hard disks). Like character devices, block devices are identified by major and minor numbers. The kernel's block I/O subsystem contains a number of layers to modularize block I/O operations by placing common code in each layer. Figure 20.22 depicts the layers through which block I/O requests pass. To minimize the amount of time spent accessing block devices, the kernel uses two primary strategies: caching data and clustering I/O operations. Buffering and Caching To reduce the number of block I/O operations for disk devices, the kernel buffers and caches I/O requests. When a process requests data from a block device (typically Figure 20.22 | Block I/O subsystem layers. 988 Case Study: Linux a hard disk), the kernel searches the page cache for the requested blocks. (Recall from Section 20.6.3, Page Replacement, that the page cache is a region of main memory that stores buffered and cached data from I/O requests.) If the page cache contains an entry for the requested block, the kernel will copy that page to the user's virtual space, provided there are no errors (e.g., improper permission). When a process attempts to write data to a block device, the request is typically placed on a list of pending requests that is sorted according to the kernel's disk scheduling strategy. By performing direct I/O, a driver can bypass kernel caches when reading from (or writing to) devices. Some applications, such as high-end database applications, implement their own caching mechanisms; as a result, it would be wasteful for the kernel to maintain its own cache of the application's data. 143 When direct I/O is enabled, the kernel performs I/O directly between a process's user address space and the device, eliminating the overhead caused by copying data from a user address space to the kernel caches, then to the device. Request Lists If an I/O request corresponds to data that is not cached or data that must be written to secondary storage, the kernel must perform an I/O operation. Instead of submitting I/O requests to devices in the order in which they are received, the kernel adds a request to a request list. A request list, which contains pending I/O operations, is created for each block device in the system. The list allows the kernel to order requests to take into account factors such as the location of the disk head if the block device is a hard disk. As discussed in Chapter 12, the kernel can improve the performance of all block I/O operations by sorting requests for I/O on each block device. To associate entries in the request list with page frames, each request contains a bio structure, which maps to a number of pages in memory corresponding to the request. The kernel maintains at least one request list per driver; each request corresponds to a read or write operation. 144 ' 145 Block drivers do not define read and write operations, but rather must implement a request function that the kernel calls, once it has queued requests. 146 This allows the kernel to improve I/O performance by sorting the list of requests according to its disk scheduling algorithm (discussed in the next section) before submitting requests to a block device. When the kernel calls a request function, the block device must perform all I/O operations in the list of I/O requests the kernel provides. Although the kernel often reduces seek time by sorting block device requests, in some cases the request list is detrimental to performance. For example, certain device drivers, such as RAID drivers, implement their own methods for managing requests (see Section 12.10, Redundant Arrays of Independent Disks (RAID)). Such device drivers operate on bios, unlike traditional block device drivers (e.g.. IDE), which are passed a list of requests via a request function.147 Elevator Disk Scheduling Algorithm Linux provides several disk scheduling algorithms to allow users to customize I/O performance to meet the individual needs of each system. The default disk schedul- 20.8 Input/Output Management 989 ing algorithm is a variation of the elevator algorithm (i.e., the LOOK variation of the SCAN strategy presented in Section 12.5.6, LOOK and C-LOOK Disk Scheduling). To minimize disk seek time, the kernel arranges the entries in the list according to their location on disk. The request at the head of the list is closest to the disk head, which reduces the amount of time the disk spends seeking and increases I/O throughput. When an I/O request is submitted to a disk's request list, the kernel determines the location on disk corresponding to the request. The kernel then attempts to merge requests to adjacent locations on disk by combining two I/O requests into a single, larger request. Merging requests improves performance by reducing the number of I/O requests issued to a block device. If a request cannot be merged, the kernel attempts to insert that request in the sorted list in the position that maintains the list's least-seek-time-first ordering. 148 Although the elevator algorithm provides high throughput by reducing disk seek latency, the algorithm allows requests at the end of the queue to be indefinitely postponed. 149 For example, consider two processes: process P1 writes 200MB of data to a file and process P2 recursively reads the contents of a directory on disk and prints the result to the terminal. Assume that the system is using an ext2 file system and the request list is initially empty. As P1 executes, it may submit several write requests without blocking during its time slice—processes rarely block as result of write requests because they do not rely on the completion of write operations to execute subsequent instructions. P1 is eventually preempted, at which point the request list contains several write requests. Many of the write requests will have been merged by the kernel because the ext2 file system attempts to locate file data within block groups, as discussed in Section 20.7.3, Second Extended File System (ext2fs). Process P 1 's requests are then submitted to the block device, which moves the disk head to the location of the data blocks to be written. When P2 executes, it submits a request to read the contents of a directory. This request is a synchronous read operation because P 2 cannot print the directory contents until the read operation completes. Consequently, P2 will submit a single I/O request and block. Unless the read request corresponds to a location adjacent to the disk head (which is now servicing P 1 's write requests), the read request is placed after the pending write requests in the request list. Because P 2 has blocked, process Pi may eventually regain control of the processor and submit additional write requests. Each subsequent write request will likely be merged with the previous requests at the front of the request list. As a result, process P 2 's read request is pushed further back in the request list while P 2 remains blocked—meaning that P 2 cannot submit additional I/O requests. As long as process Pi continues to submit write requests, process P 2 's read request is indefinitely postponed. Deadline and Anticipatory Disk Scheduling Algorithms To eliminate indefinite postponement, the kernel provides two LOOK disk scheduling algorithms: deadline scheduling and anticipatory scheduling. The deadline scheduler prevents read requests from being indefinitely postponed by assigning each request a deadline—the scheduler attempts to service each request before its 990 Case Study: Linux deadline passes. When the request has been waiting for the longest time permitted by the deadline, the request expires. The deadline scheduler processes requests from the head of the request list unless a request expires. At this point, the deadline scheduler services any requests that have expired and some that will soon expire. Servicing requests that will soon expire reduces the number of times the scheduler will be interrupted from servicing requests at the head of the list, which improves throughput. After all expired requests have been serviced, the scheduler continues by servicing requests from the front of the request list.150 To meet its deadlines, the scheduler must be able to quickly determine if any requests have expired. Using a single request list, the deadline scheduler would need to perform a linear search of requests to determine if any had expired. If the number of requests in the request list is large, this could require a significant amount of time, leading to poor performance and missed deadlines. Consequently, the deadline scheduler maintains two FIFO queues, one each for read and write requests. When a request is added to the request list, a reference to the request is added to the appropriate FIFO queue. Therefore, the request that has waited for the longest time is always located at the front of the FIFO queue. This means that the deadline I/O scheduler can quickly determine if a request is near its deadline by accessing the pointer to the front of each of the FIFO queues.151, 152 Because the deadline scheduler is designed to prevent read starvation, the deadline for read requests is shorter than the deadline for write requests. By default, read requests must be serviced 500ms after insertion into the request list, whereas write requests must be serviced after 5 seconds. These values were chosen because they provide good performance in general, but they can be modified by a system administrator at run time. 153 Consider how the deadline scheduler performs given processes P1 and P2 from the previous section (process P1 writes 200MB of data to a file and process P2 recursively reads the contents of a directory on disk and prints the result to the screen). Recall from the previous section that process P 1 's write requests are typically performed before read requests because its requests are merged. As a result, the disk will most likely be servicing a write request when P 2 's read request's deadline expires. This means that the disk will likely perform a seek operation to perform the read request. Also, because synchronous read requests require process P 2 to block, the number of read requests in the request list is small. Consequently, the next request in the request list following the read operation will likely be a write operation that requires another seek operation. If several processes issue read requests while one or more processes submit write requests, several such pairs of seek operations (due to expired read requests) may occur within a short period of time. Thus, to reduce the number of seeks, the deadline I/O scheduler attempts to group several expired requests so that they will be serviced together before expired write requests are serviced, and vice versa.154, 155 The anticipatory scheduler eliminates read request starvation by preventing excessive seek activity and further improves performance by anticipating future 20.8 Input/Output Management 991 requests. Recall that synchronous read requests often occur once per time slice because they require processes to block. However, similar to process P 2 's requests for directory entries, many processes issue a series of synchronous read requests for contiguous data (or data on a single track). Consequently, if the disk scheduler paused briefly after completing a process's read request, that process may issue an additional read request that does not require a seek operation. 156 Even when there are several other requests in the request list, this read request would be placed at the head of request list and serviced without causing excessive seek activity. By default, the amount of time during which the anticipatory I/O scheduler waits for a new request is 6ms —a pause that occurs only after completing a read request.157 The 6ms pause (the value of the pause can be modified at run time) corresponds to the seek latency for many of today's hard disks, or roughly half the amount of time required to perform a request located on another track and return to the location of the previous read. If a read request is issued during the 6ms pause, the anticipatory scheduler can perform the request before seeking to another location on disk to perform requests for other processes. In this case, a traditional elevator scheduler would have performed two seek operations: one to perform the next request from another process and one to service the next read request. Therefore, the anticipatory I/O scheduler improves overall I/O throughput if it receives a read request within the 6ms pause more than 50 percent of the time. The anticipatory scheduler has been shown to perform 5 to 100 times better than the traditional elevator algorithm when performing synchronous reads in the presence of write requests. However, the anticipatory I/O scheduler can introduce significant overhead due to its 6ms pause, leading to reduced I/O throughput. This occurs when the I/O scheduler does not receive a request for data near the disk's read/write head during the 6ms that it waits for a read request. To minimize this overhead, the scheduler maintains a history of process behavior that it uses to predict whether a process will benefit from the 6ms pause. 158 20.8.4 Network Device I/O The kernel's networking subsystem provides an interface for exchanging data with other hosts. This interface, however, cannot be accessed directly by user processes, which must send and receive data via the IPC subsystem's socket interface (discussed in Section 20.10.3, Sockets). When processes submit network data to the socket interface, they specify the network address of the destination, not the network device through which to deliver the data. The networking subsystem then determines which network device will deliver the packet. An important difference between network devices and block or character devices is that the kernel does not request data from a network device. Instead, network devices use interrupts to notify the kernel as they receive packets. Because network traffic travels in packets, which can arrive at any time, the read and write operations of a device special file are not sufficient to access data from network devices. Instead, the kernel uses net_device structures to describe 992 Case Study: Linux the network devices.159 These objects are similar to the device_struct objects that represent block and character devices; however, because network devices are not represented as files, the net_device structure does not include a file_operations structure. Instead, it contains a number of functions defined by drivers that allow the kernel to perform actions such as starting a device, stopping a device and sending packets to a device.160 Once the kernel has prepared packets to transmit to another host, it passes them to the device driver for the appropriate network interface card (NIC). To determine which NIC will send the packet, the kernel examines an internal routing table that lists the destination addresses each network interface card can access. Once the kernel matches the packet's destination address to the appropriate interface in the routing table, the kernel passes the packet to the device driver. Each driver processes packets according to a queuing discipline, which specifies the order in which its device processes packets, such as the default FIFO policy, or other, more sophisticated, priority-based policies. By enabling priority queuing disciplines, the system can deliver higher-priority content, such as streaming media, more quickly than other network traffic.161 After passing packets to a network device's queue, the kernel wakes the device so that the driver may begin removing packets from the device's queue according to its queuing discipline. As packets are removed from the queue, they are passed to a packet transmission function specified by the device's driver.162 When a network interface receives a packet from an external source, it issues an interrupt. The interrupt causes processor control to pass to the appropriate interrupt handler for packet processing. The interrupt handler allocates memory for the packet, then passes the packet to the kernel's networking subsystem. In Section 20.11, Networking, we discuss the path taken by packets as they travel through the networking subsystem. 20.8.5 Unified Device Model The unified device model is an attempt to simplify device management in the kernel. At the physical level, devices are attached to an interface (e.g., a PCI slot or a USB port) that is connected to the rest of the system via a bus. As discussed in the previous sections, Linux represents devices as members of device classes. For example, a mouse connected to a USB port and a keyboard connected to a PS/2 port both belong to the input device class, but each device connects to the computer via a different bus. Whereas a description of device interfaces and buses is a physical view of the system, a device class is a software (i.e., abstract) view of the system that simplifies device management by grouping devices of a similar type. Before the unified device model, device classes were not related to system buses, meaning that it was difficult for the system to determine where in the system a device was physically located. This was not a problem when computers did not support hot swappable devices (i.e., devices that can be added and removed while the computer is running). In the absence of hot swappable devices, it is sufficient for the 20.8 Input/Output Management 993 system to detect devices exactly once (at boot time). Once the kernel loads a driver corresponding to each device, the kernel rarely needs to access a device's physical location in the system. However, in the presence of hot swappable devices, the kernel must be aware of the physical layout of the system so it can detect when a device has been added or removed. Once it has located a new device, the kernel must be able to identify its class so the kernel can load the appropriate device driver.163 For example, devices are commonly added and removed from the USB bus. To detect such changes, the kernel must periodically poll the USB interface to determine which devices are attached to it.164 If a new device is found, the kernel should identify it and load a device driver that supports it so that processes may use the device.

If a device has been removed, the system should unregister the driver so that attempts to access the device are denied. Thus, the kernel must maintain a layout of the physical location of devices in the system so that it knows when the set of devices in the system changes. To support hot swappable devices, the kernel uses the unified device model to access the physical location of a device in addition to its device class representation. 165 The unified device model defines data structures to represent devices, device drivers, buses, and device classes. The relationship between these structures is shown in Fig. 20.23. Each bus data structure represents a particular bus (e.g., PCI) and contains pointers to a list of devices attached to the bus and drivers that operate devices on the bus. Each class data structure represents a device class and contains pointers to a list of devices and device drivers that belong to that class. The unified device model associates each device and device driver with a bus and class. As a result, the kernel can access a bus, determine a list of devices and drivers on that bus and then determine the class to which each device and device driver belongs. Similarly, the kernel can access a device class, follow the pointers to its list of devices and device drivers and determine the bus to which each device is attached. As Fig. 20.23 demonstrates, the kernel requires a reference only to a single data structure to access all other data structures in the unified device model. This simplifies device management for the kernel as devices are added to and removed from the system. When devices are registered with the system, these data structures are initialized and corresponding entries are placed in the system file system, sysfs, Sysfs provides an interface to devices described by the unified device model. Sysfs allows user applications to view the relationship between entities (devices, device drivers, buses and classes) in the unified device model.166, 167, 168, 169 Sysfs, typically mounted at /sys, organizes devices according to both the bus to which they are attached and the class to which they belong.The /sys/bus directory contains entries for each bus in the system (e.g., /sys/bus/pci). Within each bus subdirectory is a list of devices and device drivers that use the bus. Sysfs also organizes devices by class in the directory /sys/class. For example, the /sys/class/input contains input devices, such as a mouse or keyboard. Within each class subdirectory is a list of devices and device drivers that belong to that class.170-171 994 Case Study: Linux Figure 20.23 | Unified device model organization. Power Management The unified device model has also simplified how the kernel performs power management, an important consideration for battery-powered systems. As the number and power of mobile computers increases, a system's ability to manage power to increase battery life has become more important. Power management standards such as the Advanced Configuration and Power Interface (ACPI) specify several device power states, each of which results in different power consumption. For example, the ACPI defines four power states from fully on (DO) to fully off (D3). When a device transitions from fully on to fully off, the system does not provide any power to the device and any volatile data stored in the device, known as the device context, is lost.172 The unified device model simplifies power management for the 20.8 Input/Output Management 995 kernel by providing a data structure to store the context for each device when its power state changes. There are several devices that provide powered connections to other devices. For example, a PCI card that contains USB ports may contain connections to USB devices. Because some USB devices are powered through the USB cable, if power is removed from the PCI card, then power is removed from each of its attached devices. Consequently, the PCI card should not enter the fully off state until each of its attached devices has entered the fully off state. The unified device model allows the kernel to detect such power dependencies by exposing the physical structure of devices in the system. As a result, the kernel can prevent a device from transitioning to a different power state if the transition will prevent other devices from properly functioning.173 20.8.6 Interrupts The kernel requires that each device driver register interrupt handlers when the driver is loaded. As a result, when the kernel receives an interrupt from a particular device, the kernel passes control to its corresponding interrupt handler. Interrupt handlers do not belong to any single process context because they are not themselves programs. Because an interrupt handler is not identified as any task_struct object's executable code, the scheduler cannot place it in any run queues. This characteristic places some restrictions on interrupt handlers; lacking its own execution context, an interrupt handler cannot sleep or call the scheduler. If an interrupt handler were permitted to sleep or call the scheduler, it never would regain control of the processor. Similarly, interrupt handlers cannot be preempted, as doing so would invoke the scheduler.174 Any preemption requests received during interrupt handling are honored when the interrupt handler completes execution. Finally, interrupt handlers cannot cause exceptions or faults while executing. In many architectures, the system aborts when an exception is raised during an interrupt handler. 175 To improve kernel performance, most drivers attempt to minimize the processor cycles required to handle hardware interrupts. This is another reason why kernel memory is never swapped to disk—loading a nonresident page while an interrupt handler is executing takes substantial time. In Linux, a driver handling one interrupt cannot be preempted by other interrupts that use the same interrupt line. If this were not the case, any device driver containing nonreentrant code might perform operations that leave a device in an inconsistent state. As a result, when one driver is processing an interrupt, the kernel queues or drops any other interrupts it receives that use the same interrupt line.176 Therefore, driver developers are encouraged to write code that processes interrupts as quickly as possible. The kernel helps improve interrupt-handling efficiency by dividing interrupthandling routines into two parts—the top half and the bottom half. When the kernel receives an interrupt from a hardware device, it passes control to the top half of the driver's interrupt handler. The top half of an interrupt handler performs the mini- 996 Case Study: Linux mum work required to acknowledge the interrupt. Other work (such as manipulating data structures)—which should be located in the bottom half of the interrupt handler—is scheduled to be performed later by a software interrupt handler. Top halves of interrupt routines cannot be interrupted by software interrupt handlers. Two primary software interrupt-handler types are softirqs and tasklets. Softirqs can be executed concurrently on multiple processors (up to one per processor), making them ideal for SMP systems.177 When a device driver allocates a softirq, it specifies the action to be performed each time the softirq is scheduled. Because multiple copies of a softirq can run simultaneously softirq actions must be reentrant to perform reliably. Network devices on Web servers, which constantly receive packets of data from external sources, benefit from softirqs because multiple packets can be processed simultaneously on different processors.178 However, softirqs do not improve performance for several types of interrupt handlers. For example, a driver that requires exclusive access to data would need to enforce mutual exclusion if its code were executed simultaneously on multiple processors, fn some cases, the overhead due to enforcing mutual exclusion can outweigh the benefits of multiprocessing. Other devices transfer data as a series of bits, requiring device drivers to sequentialize access to such data. Such devices cannot benefit from parallel processing and consequently use tasklets to perform bottomhalf interrupt-handling routines. 179 Tasklets are similar to softirqs but cannot run simultaneously on multiple processors and therefore cannot take advantage of parallel processing. As a result. most drivers use tasklets instead of softirqs to schedule bottom halves. Although multiple instances of a single tasklet cannot execute simultaneously, several different tasklets can execute simultaneously in SMP systems.180 Softirqs and tasklets normally are handled in interrupt context or in a process's context immediately after the top-half interrupt handler completes, executing with higher priority than user processes. If the system experiences a large number of softirqs that reschedule themselves, user processes might be indefinitely postponed. Thus, when user processes have not executed for a significant period of time, the kernel assigns softirqs and tasklets to be executed by the kernel thread ksoftirqd, which executes with low-priority (+19). When the kernel is loaded, the Linux kernel creates an instance of the kernel thread ksoftirqd for each processor. These threads remain sleeping until the kernel wakes them. Once scheduled, ksoftirqd enters a loop in which it processes pending tasklets and softirqs sequentially, ksoftirqd continues processing tasklets and softirqs until the tasklets and/or softirqs have completed execution or until ksoftirqd is preempted by the scheduler.181 Partitioning interrupt handling into top and bottom halves minimizes the amount of time that hardware interrupts are disabled. Once a driver handles a hardware interrupt (the top half), the kernel can run the software interrupt handler (the bottom half), during which incoming interrupts can preempt the software interrupt handler. 182 This division of driver code improves the response time of a system by reducing the amount of time during which interrupts are disabled. 20.9 20.9 Kernel Synchronization A process executing in user mode cannot directly access kernel data, hardware, or other critical system resources—such processes must rely on the kernel to execute privileged instructions on their behalf. These operations are called kernel control paths. If two kernel control paths were to access the same data concurrently, a race condition could result.183 To prevent this, the kernel provides two basic mechanisms for providing mutually exclusive access to critical sections: locks and semaphores. 20.9.1 Spin Locks Spin locks allow the kernel to protect critical sections in kernel code executing on SMP-enabled systems. Before entering its critical section, a kernel control path acquires a spin lock. The region remains protected by the spin lock until the kernel control path releases the spin lock. If a second kernel control path attempts to acquire the same spin lock to enter its critical section, it will enter a loop in which it busy waits, or "spins," until the first kernel control path releases the spin lock. Once the spin lock becomes available, the second kernel control path can acquire it.184 Proper use of spin locks prevents race conditions among multiple kernel control paths executing concurrently in an SMP system, but serves no purpose in a uniprocessor system in which two kernel control paths cannot simultaneously execute. Consequently, kernels configured for uniprocessor systems exclude the locking portion of spin lock calls.185 This improves performance by eliminating the costly instructions executed to acquire mutually exclusive access to a critical section in multiprocessor systems. The kernel provides a set of spin lock functions for use in interrupt handlers. Because a hardware interrupt can preempt any execution context, any data shared between a hardware interrupt handler and a software interrupt handler must be protected using a spin lock. To address this issue, the kernel provides spin locks that disable interrupts on the local processor while still allowing concurrent execution on SMP systems. On uniprocessor systems, the spin lock code is removed when the kernel is compiled, but the code for enabling and disabling interrupts remains intact.186 To protect data shared between user contexts and software interrupt handlers, the kernel uses bottom-half spin locks. These functions disable software interrupt handlers in addition to acquiring the requested spin lock.187 All spin lock variations disable preemption in both single and multiprocessor systems. Although disabling preemption could lead to indefinite postponement or even deadlock, allowing code protected by spin locks to be preempted introduces the same race conditions spin locks are designed to avoid. The kernel uses a preemption lock counter to determine if a kernel control path can be preempted. When a kernel control path acquires a spin lock, the preemption lock counter is incremented; the counter is decremented when the kernel control path releases the spin lock. Code executing in kernel mode can be preempted only when the preemption lock counter is reduced to zero. When a spin lock is released and its associated Kernel Synchronization 997 998 Case Study: Linux counter becomes zero, the kernel honors any pending preemption requests by invoking the scheduler. Kernel developers must abide by certain rules to avoid deadlock when using any spin lock variant. First, if a kernel control path has already acquired a spin lock, the kernel control path must not attempt to acquire the spin lock again before releasing it. Attempting to acquire the spin lock a second time will cause the kernel control path to busy wait for the lock it controls to be released, causing deadlock. Similarly, a kernel control path must not sleep while holding a spin lock. If the next task that is scheduled attempts to acquire the spin lock, deadlock will occur.188 20.9.2 Reader/Writer Locks In some cases, multiple kernel control paths need only to read (not write) the data accessed inside a critical section. When no kernel control path is modifying that data, there is no need to prevent concurrent read access to the data (see Section 6.2.4, Monitor Example: Readers and Writers). To optimize concurrency in such a situation, the kernel provides reader/writer locks. Reader/writer spin locks and kernel semaphores (Section 20.9.4, Kernel Semaphores) allow multiple kernel control paths to hold a read lock, but permit only one kernel control path to hold a write lock with no concurrent readers. A kernel control path that holds a read lock on a critical section must release its read lock and acquire a write lock if it wishes to modify data. 189 An attempt to acquire a write lock succeeds only if there are no other readers or writers concurrently executing inside their critical sections. Reader/writer locks effect a higher level of concurrency by limiting access to a critical section only when writes occur. Depending on the kernel control paths accessing the lock, this can lead to improved performance. If readers do not release their read locks, it is possible for writers to be indefinitely postponed. To prevent indefinite postponement and provide writers with fast access to critical sections, kernel control paths use the seqlock. 20.9.3 Seqlocks In some situations, the kernel employs another locking mechanism designed to allow writers to access data immediately without waiting for readers to release the lock. This locking primitive, called a seqlock, represents the combination of a spin lock and a sequence counter. Writing to data protected by a seqlock is initiated by calling function write_seqlock. This function acquires the spin lock component of the seqlock (so that no other writers can enter their critical section) and it increments the sequence counter. After writing, function write_sequnlock is called. which once again increments the sequence counter, then releases the spin lock.190 To enable writers to access data protected by a seqlock immediately, the kernel does not allow readers to acquire mutually exclusive access to that data. Thus, a reader executing its critical section can be preempted, enabling a writer to modify the data protected by a seqlock. The reader can detect if a writer has modified the value of the data protected by the seqlock by examining the value of the seqlock's 20.9 Kernel Synchronization 999 sequence counter as shown in the following pseudocode. The value of the seqlock's sequence counter is initialized to zero. Do Store value ofseqlock's sequence counter in local variable seqTemp Execute instructions that read the value of the data protected by the seqlock While seqTemp is odd or not equal to the value of the seqlock's sequence counter After entering its critical section, the reader stores the value of the seqlock's sequence counter. Let us assume that this value is stored in variable seqTemp. The reader then accesses the data protected by the sequence counter. Consider what occurs if the system preempts the reader and a writer enters its critical section to modify the protected data. Before modifying the protected data, the writer must acquire the seqlock, which increments the value of the sequence counter. When the reader next executes, it compares the value stored in seqTemp to the current value of the sequence counter. If the two values are not equal, a writer must have entered its critical section. In this case, the value read by the reader may not be valid. Therefore, the loop continuation condition in the preceding pseudocode determines if the value of the sequence counter has changed since the reader accessed the protected data. If so, the reader continues the loop until it has read a valid copy of the protected data. Because the value of the seqlock's sequence counter is initialized to zero, a write is being performed when that value is odd. Thus, if seqTemp is odd, the reader will read the protected data while a writer is in the process of modifying it. In this case, the loop continuation condition ensures that the reader continues the loop to ensure that it reads valid data. When no writers have attempted to modify the data while the reader executes inside its critical section, the reader exits the do. . .while loop. Because writers need not wait for readers to release a lock, seqlocks are appropriate for interrupt handling and other instances when writers must execute quickly to improve performance. In most cases, readers successfully read data on the first try. However, it is possible for readers to be indefinitely postponed while multiple writers modify shared data, so seqlocks should be used in situations where protected data is read more often than it is written.191 20.9.4 Kernel Semaphores Spin locks and seqlocks perform well when the critical sections they protect contain few instructions. However, as the size of the critical section increases, the amount of time spent busy waiting increases, leading to significant overhead and performance degradation. Also, spin locks can lead to deadlock if a process sleeps while holding the lock. When a critical section must be protected for a long time, kernel semaphores are a better choice for implementing mutual exclusion. For example, only one process at a time should be able to read an image from a scanner. Because scanners may take several seconds to scan an image, a scanner device driver typically enforces mutually exclusive access to its scanner using semaphores. 1000 Case Study: Linux Kernel semaphores are counting semaphores (see Section 5.6.3, Counting Semaphores) represented by a wait queue and a counter. The wait queue stores processes that are waiting on the kernel semaphore. The value of the counter determines how many processes may simultaneously access their critical sections. When a kernel semaphore is created, the counter is initialized to the number of processes allowed to access the semaphore concurrently. For example, if a semaphore protects access to three identical resources, then the initial counter value would be set to 3.192 When a process attempts to execute its critical section, it calls function down on the kernel semaphore. Function down, which corresponds to the P operation (see Section 5.6, Semaphores), checks the current value of the counter and responds according to the following rules: • If the value of the counter is greater than 0, down decrements the counter and allows the process to execute. • If the value of the counter is less than or equal to 0, down decrements the counter, and the process is added to the wait queue and enters the sleeping state. By putting a process to sleep, the kernel reduces the overhead due to busy waiting because sleeping processes are not dispatched to a processor. When a process exits its critical section, it releases the kernel semaphore by calling function up. This function inspects the value of the counter and responds according to the following rules: • If the value of the counter is greater than or equal to 0, up increments the counter. • If the value of the counter is less than 0, up increments the counter, and a process from the wait queue is awakened so that it can execute its critical section.193 Kernel semaphores cause processes to sleep if placed in the semaphore's wait queue, so they cannot be used in interrupt handlers or when a spin lock is held. However, the kernel provides an alternative solution that allows interrupt handlers to access semaphores using function down_trylock. If the interrupt handler cannot enter the critical section protected by the kernel semaphore, function down_trylock will return to the caller instead of causing the interrupt handler to sleep.194 Kernel semaphores should be used only for process that need to sleep while holding the semaphore; processes that sleep while holding a spin lock can lead to deadlock. 20.10 Interprocess Communication Many of the interprocess communication (IPC) mechanisms available in Linux are derived from traditional UNIX IPC mechanisms, and they all have a common goal: to allow processes to exchange information. Although all IPC mechanisms accomplish this goal, some are better suited for particular applications, such as those that communicate over a network or exchange short messages with other local applications. In this section, we discuss how IPC mechanisms are implemented in the Linux kernel and how they are employed in Linux systems. 20.10 Interprocess Communication 1001 20.10.1 Signals Signals were one of the first interprocess communication mechanisms available in UNIX systems —the kernel uses them to notify processes when certain events occur. In contrast to the other Linux IPC mechanisms we discuss, signals do not allow processes to specify more than a word of data to exchange with other processes; signals are primarily intended to alert a process that an event has occurred.195 The signals a Linux system supports depend on the processor architecture. Figure 20.24 lists the first 20 signals identified by the POSIX specification (all of today's architectures support these signals).196 Signals, which are created by the kernel in response to interrupts and exceptions, are sent to a process or thread either as a result of executing an instruction (such as a segmentation fault, SIGSEGV), from another process (such as when one process terminates another, SIGKILL) or from an asynchronous event (e.g., an I/O completion signal).The kernel delivers a signal to a process by pausing its execution Default Signal Type Action Description 1 SIGHUP Abort Hang-up detected on terminal or death of controlling process 2 SICINT Abort Interrupt from keyboard 3 4 SIGQUIT Dump Quit from keyboard SIGILL Dump Illegal instruction 5 SIGTRAP Dump Trace/breakpoint trap 6 SIGABRT Dump Abort signal from abort function 7 SIGBUS Dump Bus error 8 SIGFPE Dump Floating point exception 9 10 SIGKILL Abort Kill signal SIGUSR1 Abort User-defined signal 1 11 12 SIGSEGV Dump Invalid memory reference SIGUSR2 Abort User-defined signal 2 13 14 SIGPIPE Abort Broken pipe: write to pipe with no readers SIGALRM Abort Timer signal from alarm function 15 16 SIGTERM Abort Termination signal SIGSTKFLT Abort Stack fault on coprocessor 17 SIGCHLD Ignore Child stopped or terminated 18 SIGCONT Continue Continue if stopped 19 SIGSTOP Stop Stop process 20 SIGTSTP Stop Stop typed at terminal device Figure 20.24 | POSIX signals.197 1002 Case Study: Linux and invoking the process's corresponding signal handler. Once the signal handler completes execution, the process resumes execution.198 A process or thread can handle a signal in one of three ways. (1) Ignore the signal—processes can ignore all but the SIGSTOP and SIGKILL signals. (2) Catch the signal—when a process catches a signal, it invokes its signal handler to respond to the signal. (3) Execute the default action that the kernel defines for that signal—by default, the kernel defines one of five actions that is performed when a process receives a signal.199 The first default action is to abort, which causes the process to terminate immediately. The second is to perform a memory dump. A memory dump is similar to an abort; it causes a process to terminate, but before it does so, the process generates a core file that contains the process's execution context, which includes the process's stack, registers and other information useful for debugging. The third default action is simply to ignore the signal. The fourth is to stop (i.e., suspend) the process—often used to debug a process. The fifth is continue, which reverses the fourth by switching a process from the suspended state to the ready state. 200 Processes can choose not to handle signals by blocking them. If a process blocks a specific signal type, the kernel does not deliver the signal until the process stops blocking it. Processes block a signal type by default while handling another signal of the same type. As a result, signal handlers need not be reentrant (unless the default behavior is not used), because multiple instances of the process's signal handler cannot be executed concurrently. It is, however, possible for a signal handler to interrupt a signal handler of a different type.201 Common signals, such as those shown in Fig. 20.24, are not queued by the kernel. If a signal is currently being handled by a process and a second signal of the same type is generated for that process, the kernel discards the latter signal. If two signals are generated simultaneously by an SMP system, the kernel simply drops one as a result of the race condition. In certain circumstances, dropped signals do not affect system behavior. For example, a single SIGKILL signal is sufficient for the system to terminate a process. In mission-critical systems, however, dropped signals could be disastrous. For example, a user-defined signal may be used to monitor safety systems that protect human life. If such signals were dropped, people's lives could be at risk. To ensure that such systems do not miss a signal, Linux supports real-time signals. These are queued by the kernel; therefore, multiple instances of the same signal can be generated simultaneously and not be discarded.202-203 By default the kernel queues up to 1,024 real-time signals of the same type; any further signals are dropped. 20.10.2 Pipes Pipes enable two processes to communicate using the producer/consumer model. The producer process writes data to the pipe, after which the consumer process reads data from the pipe in first-in-first-out order. When a pipe is created, an inode is allocated and assigned to the pipe. Similar to procfs inodes (see Section 20.7.4, Proc File System), pipe inodes do not point to 20.10 Interprocess Communication 1003 disk blocks. Rather, they point to one page of data called a pipe buffer that the kernel uses as a circular buffer. Each pipe maintains a unique pipe buffer that stores data that is transferred between two processes.204 When pipes are created, the kernel allocates two file descriptors (see Section 20.7.1, Virtual File System) to allow access to the pipe: one for reading from the pipe and one for writing to the pipe. Pipes are represented by flies and accessed via the virtual file system. To initiate communication using a pipe, one process must create the pipe, then fork a child process with which to communicate via the pipe. The fork system call enables pipe communication because it allows the child process to inherit the parent process's file descriptors. Alternatively, two processes can share a file descriptor using sockets, discuss in the section that follows. Although the kernel represents pipes as files, pipes cannot be accessed from the directory tree. This prevents a process from accessing a pipe unless it has obtained the pipe's file descriptor from the process that created it.205 One limitation of pipes is that they support communication only between processes that share file descriptors. Linux supports a variation of a pipe, called a named pipe or FIFO, that can be accessed via the directory tree. When a FIFO is created, its name is added to the directory tree. Processes can access the FIFO by pathname as they would any other file in the directory tree (the location and name of the file are typically known before the processes execute). Therefore, processes can communicate using a named pipe the same way they access data in a file system—by supplying the correct pathname and appropriate file permissions. However, unlike data files, FIFOs point to a buffer located in memory, not on disk. Therefore, FIFOs provide the simplicity of sharing data in files without the latency overhead created by disk access.206 Another limitation of pipes that the fixed-size buffer can result in suboptimal performance if a producer and consumer work at different speeds, as discussed in Section 6.2.3, Monitor Example: Circular Buffer, 20.10.3 Sockets The Linux socket IPC mechanism allows pairs of processes to exchange data by establishing direct bidirectional communication channels. Each process can use its socket to transfer data to and from another process. One limitation of pipes is that communication occurs in a single direction (from producer to consumer); however, many cooperating processes require bidirectional communication. In distributed systems, for example, processes may need to be able to send and receive remote procedure calls. In this case, pipes are insufficient, because they are limited to communication in one direction and are identified by file descriptors, which are not unique among multiple systems. To address this issue, sockets are designed allow communication between unrelated processes so that such processes can exchange information both locally and across a network. Because sockets allow such flexibility, they may perform worse than pipes in some situations. For example, if an application requires unidirectional communication between two processes in one system 1004 Case Study: Linux (i.e., sending the output of a decompression program to a file on disk), pipes should be used instead of sockets. There are two primary socket types that processes can employ. Stream sockets transfer information as streams of bytes. Datagram sockets transfer information in independent units called datagrams, discussed in Section 16.6.2, User Datagram Protocol (UDP). 207 Stream Sockets Processes that communicate via stream sockets follow the traditional client/server model. The server process creates a stream socket and listens for connection requests. A client process can then connect to the server process and begin exchanging information. Because data is transferred as a stream of bytes, processes communicating with stream sockets can read or write variable amounts of data. One useful property of stream sockets is that, unlike datagram sockets, they use TCP to communicate, which guarantees that all data transmitted will eventually arrive and will be delivered in the correct order. Because stream sockets inherently provide data integrity, they are typically the better choice when communication must be reliable. 208 Datagram Sockets Although stream sockets provide powerful IPC features, they are not always necessary or practical. Enabling reliable stream sockets creates more overhead than some applications can afford. Faster, but less reliable communication can be accomplished using datagram sockets. For example, in some distributed systems, a server with many clients periodically broadcasts status information to all of its clients. In this case, datagram sockets are preferable to stream sockets, because they require only a single message to be sent from the server socket and do not require any responses from clients. Datagrams may also be sent periodically to update client information, such as for clock synchronization purposes. In this case, each subsequent datagram is intended to replace the information contained in previous datagrams. Therefore, the clients can afford not to receive certain datagram packets, provided that future datagram packets arrive eventually. In such situations, where data loss is either unlikely or unimportant, applications can use datagram sockets in lieu of stream sockets to increase performance. Socketpairs Although sockets are most often used for Internet communication, Linux enables bidirectional communication between multiple processes on the same system using sockets. When a process creates a socket in the local system, it specifies a file name that is used as the socket's address. Other sockets on that system can use the file name to communicate with that socket by reading from, and writing to, a buffer. Like many data structures in the Linux kernel, sockets are stored internally as files, and therefore can be accessed via the virtual file system using the read and write system calls.209 20.10 Interprocess Communication 1005 Linux provides another IPC mechanism that is implemented using sockets, called a socketpair. A socketpair is a pair of connected, unnamed sockets. When a process creates a socketpair, the kernel creates two sockets, connects them, and returns a file descriptor for each socket.210 Similar to pipes, unnamed sockets are limited to use by processes that share file descriptors. Socketpairs are traditionally employed when related processes require bidirectional communication. 20.10.4 Message Queues Messages are an IPC mechanism to allow processes to transmit information that is composed of a message type and a variable-length data area. Message types are not defined by the kernel; when processes exchange information, they specify their own message types to distinguish between messages. Messages are stored in message queues, where they remain until a process is ready to receive them. Message queues, unlike pipes, can be shared by processes that are not parent and child. When the kernel creates a message queue, it assigns it a unique identifier. Related processes can search for a message queue identifier in a global array of message queue descriptors. Each descriptor contains a queue of pending messages, a queue of processes waiting for messages (message receivers), a queue of processes waiting to send messages (message senders), and data describing the size and contents of the message queue. 211 When a process adds a message to a message queue, the kernel checks the queue's list of receivers for a process waiting for messages of that type. If it finds any such processes, the kernel delivers the message each of them. If no receiver is waiting for a message of the specified type and enough space is available in the message queue, the kernel adds the message to a queue of pending messages of that type. If insufficient space is available, the message sender adds itself to the queue of message senders. Senders wait in this queue until space becomes available (i.e., when a message is removed from the queue of pending messages). 212 When a process attempts to receive a message, the kernel searches for messages of a specified type in the appropriate message queue. If it finds such a message, the kernel removes the message from the queue and copies it to a buffer located in the address space of the process receiving the message. If no messages of the requested type are found, the process is added to the queue of message receivers where it waits until the requested type of message becomes available.213 20.10.5 Shared Memory The primary advantage of shared memory over other forms of IPC is that, once a region of shared memory is established, access to memory is processed in user space and does not require the kernel to access the shared data. Thus, because processes do not invoke the kernel for each access to shared data, this type of IPC improves performance for processes that frequently access shared data. Another advantage of shared memory is that processes can share as much data as they can address, potentially eliminating the time spent waiting when a producer and con- 1006 Case Study: Linux sumer work at different speeds using fixed-size buffers, as discussed in Section 6.2.3. Linux supports two standard interfaces to shared memory that are managed via tmpfs: System V and POSIX shared memory. [Note: Processes can also share memory using memory-mapped files.] The Linux implementation of System V shared memory employs four standard system calls (Fig. 20.25). When a process has successfully allocated and attached a region of shared memory, it can reference data in that region as it would reference data using a pointer. The kernel maintains a unique identifier that describes the physical region of memory to which each shared memory segment belongs, deleting the shared memory region only when a process requests its deletion and when the number of processes to which it is attached is zero.214 POSIX shared memory requires the use of the system call shm_open to create a pointer to the region of shared memory and the system call shm_unlink to close the region. The shared memory region is stored as a file in the system's shared memory file system, which must be mounted at /dev/shm; in Linux, a tmpfs file system is typically mounted there (tmpfs is described in the next section). The shm_open call is analogous to opening a file, whereas the shm_unlink call is analogous to closing a link to a file. The file that represents the shared region is deleted when it is no longer attached to any processes. Both System V and POSIX shared memory allow processes to share regions of memory and map that memory to each process's virtual address space. However, because POSIX shared memory does not allow processes to change privileges for shared segments, it is slightly less flexible than System V shared memory.215 Neither POSIX nor System V shared memory provides any synchronization mechanisms to protect access to memory. If synchronization is required, processes typically employ semaphores. Shared Memory Implementation The goal of shared memory in an operating system is to provide access to shared data with low overhead while rigidly enforcing memory protection. Linux implements shared memory as a virtual memory area that is mapped to a region of physical memory. When a process attempts to access a shared memory region, the kernel System V Shared Memory System Calls Purpose shmget Allocates a shared memory segment. shmat Attaches a shared memory segment to a process. shmctl Changes the shared memory segment's properties (e.g., permissions). shmdt Detaches (i.e., removes) a shared memory segment from a process. Figure 20.25 | System V shared memory system calls. 20.10 Interprocess Communication 1007 first determines if the process has permission to access it. If so, the kernel allocates a virtual memory area that is mapped to the region of shared memory, then attaches the virtual memory area to the process's virtual address space. The process may then access shared memory as it would any other memory in its virtual address space. The kernel keeps track of shared memory usage by treating the region as a file in tmpfs, the temporary file system. Tmpfs has been designed to simplify shared memory management while maintaining good performance for the POSIX and System V shared memory specifications. As its name suggests, tmpfs is temporary, meaning that shared memory pages are not persistent. When a file in tmpfs is deleted, its page frames are freed. Tmpfs is also swappable; that is, data stored in the tmpfs can be swapped to the backing store when available memory becomes scarce. The page(s) containing the file can then be loaded from the backing store when referenced. This allows the system to fairly allocate page frames to all processes in the system. Tmpfs also reduces shared memory overhead because it does not require mounting or formatting for use. Finally, the kernel can set permissions of tmpfs files, which enables the kernel to implement the shmctl system call in System V shared memory.216 When the kernel is loaded, an instance of tmpfs is created. If the user wishes to mount a tmpfs file system to the local directory tree (which, as previously discussed, is required for POSIX shared memory), the user can mount a new instance of the file system and access it immediately. To further improve shared memory performance, tmpfs interfaces directly with the memory manager—it has minimal interaction with the virtual file system. Although tmpfs creates dentries, inodes and file structures that represent shared memory regions, generic VFS file operations are ignored in favor of tmpfs-specific routines that bypass the VFS layer. This relieves tmpfs of certain constraints typically imposed by the VFS (e.g., the VFS does not allow a file system to grow and shrink while it is mounted). 217 20.10.6 System V Semaphores Linux implements two types of semaphores: kernel semaphores (discussed in Section 20.9.4, Kernel Semaphores) and System V semaphores. Kernel semaphores are synchronization mechanisms employed throughout the kernel to protect critical sections. System V semaphores also protect critical sections and are implemented using similar mechanisms; however, they are designed for user processes to access via the system call interface. For the remainder of this discussion, we refer to System V semaphores simply as semaphores. Because processes often need to protect a number of related resources, the kernel stores semaphores in semaphore arrays. Each semaphore in an array protects a particular resource. 218, 219 Before a process can access resources protected by a semaphore array, the kernel requires that there be sufficient available resources to satisfy the process's request. Thus, semaphore arrays can be implemented as a deadlock prevention mechanism by denying Havender's "wait-for" condition (see Section 7.7.1, Denying 1008 Case Study: Linux the "Wait-For" Condition). If a requested resource in semaphore array has been allocated to another process, the kernel blocks the requesting process and places its resource request in a queue of pending operations for that semaphore array. When a resource is returned to the semaphore array, the kernel examines the semaphore array's queue of pending operations for processes, and if a process can proceed, it is unblocked. 220 To prevent deadlock from occurring if a process terminates prematurely while it holds resources controlled by a semaphore, the kernel tracks the operations each process performs on a semaphore. When a process exits, the kernel reverses all the semaphore operations it performed to allocate its resources. Finally, note that semaphore arrays offer no protection against indefinite postponement caused by poor programming in the user processes that access them. 20.11 Networking The networking subsystem performs operations on network packets, each of which is stored in a contiguous physical memory area described by an sk_buff structure. As a packet traverses layers of the network subsystem, network protocols add and remove headers and trailers containing protocol-specific information (see Chapter 16, Introduction to Networking). 221 20.11.1 Packet Processing Figure 20.26 illustrates the path taken by network packets as they travel from a network interface card (NIC) through the kernel. When a NIC receives a packet, it issues an interrupt, which causes the NIC's interrupt handler to execute. The interrupt handler calls the network device's driver routine that allocates an sk_buff for the packet, then copies the packet from the network interface into the sk_buff and adds the packet to a queue of packets pending processing. A queue of pending packets is assigned to each processor; the interrupt handler assigns a packet to the queue belonging to the processor on which it executes.222 At this point, the packet resides in memory where it awaits further processing. Because interrupts are disabled while the top half of the interrupt handler executes. the kernel delays processing the packet. Instead, the interrupt handler raises a softirq to continue processing the packet. (Section 20.8.6, Interrupts, discussed softirqs.) After raising the softirq, the device driver routine returns and the interrupt handler exits.223 A single softirq processes all packets that the kernel receives on that processor. Because the kernel uses softirqs to process packets, network routines can execute concurrently on multiple processors in SMP systems, resulting in increased performance. When the scheduler dispatches the network softirq, the softirq processes packets in the processor's queue until either the queue is empty, a predefined maximum number of packets are processed or a time limit is reached. If one of the latter two conditions is met, the softirq is rescheduled and returns control of the processor to the scheduler.224 20.11 Networking 1009 Figure 20.26 | Path followed by network packets received by the networking subsystem. To process a packet, the network device softirq (called NET_RX_SOFTIRQ) removes the packet from the current processor's queue and passes it to the appropriate network layer protocol handler—typically the IP protocol handler. Although Linux supports other network layer protocols, they are rarely used. Therefore, we limit our discussion to the IP protocol handler. When the IP protocol handler receives a packet, it first determines its destination. If the packet destination is another host, the handler forwards the packet to the appropriate host. If the packet is destined for the local machine, the IP protocol handler strips the IP protocol-specific header from the sk_buff and passes the packet to the appropriate transport layer packet handler. The transport layer packet handler supports the Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP). 225 The transport layer packet handler determines the port specified by the TCP header and delivers the packet data to the socket that is bound to that port. The packet data is then transmitted to the process via the socket interface. 1010 Case Study: Linux 20.11.2 Netfilter Framework and Hooks As packets traverse the networking subsystem they encounter elements of the netfilter framework. Netfilter is a mechanism designed to allow kernel modules to directly inspect and modify packets. At various stages of the IP protocol handler, software constructs called hooks enable modules to register to examine, alter and/ or discard packets. At each hook, modules can pass packets to user processes.226 Figure 20.27 lists the netfilter hooks and the packets that pass through each hook. The first hook packets encounter is NF_IP_PRE_ROUTING. All incoming packets pass it as they enter the IP protocol handler. One possible use for this hook is to enable a system that multiplexes network traffic (e.g., a load balancer). For example, a load balancer attempts to evenly distribute requests to a cluster of Web servers to improve average response times. Thus, the load balancer can register the NF_IP_PRE_ROUTINC hook to intercept packets and reroute them according to the load on each Web server. After a packet passes the NF_IP_PRE_ROUTING hook, the next hook it encounters depends on its destination. If the packet destination is the current network interface, it passes through the NF_IP_L0CAL_IN hook. Otherwise, if a packet needs to be passed to another network interface, it passes through the NF_IP_F0RWARD hook. One possible use for these two hooks is to limit the amount of incoming traffic from a particular host by discarding packets once a certain threshold is reached. All locally generated packets pass through the NF_IP_LOCAL_OUT hook, which, similar to the NF_IP_LOCAL_IN and NF_IP_F0RWARD hooks, can be used to filter packets before they are sent across a network. Finally, immediately before leaving the system, all packets pass through the NF_IP_POST_ROUTTNG hook. A firewall can use this hook to modify the outgoing traffic to make it appear to have come from the firewall instead of from the original source. 20.12 Scalability The early development of the Linux kernel focused on desktop systems and lowend servers. As additional features were implemented, Linux's popularit) increased. This led to new interest in scaling Linux to larger systems (even mainframe computers) at large computer companies, such as IBM (www.ibm.com) and Hook Packets handled NF_IP_PRE_ROUTING All incoming packets. NF_IP_LOCAL_IN Packets sent to the local host. NF_IP_LOCAL_OUT Locally generated packets. NF_IP_FORWARD Packets forwarded to other hosts. NF_IP_POST_ROUTINC All packets sent from the system. Figure 20.27 | Netfilter hooks. 20.12 Scalability 1011 Hewlitt-Packard (www.hp.com), which cooperated with independent developers to scale Linux to be competitive in the high-end server market. As Linux's scalability improves, designers must decide how far to enable the standard Linux kernel to scale. Increasing its scalability might negatively affect its performance on desktop systems and low-end servers. With this in mind, companies such as Red Hat (www.redhat.com), SuSE (www.suse.com) and Conectiva (www.conectiva.com) provide Linux distributions designed for high-performance servers and sell support services tailored to those distributions. By providing kernel modifications in a distribution, companies can tailor the Linux kernel to high-end servers without affecting users in other environments. High-end servers are not the only reason to improve Linux's scalability— embedded-device manufacturers also use Linux to manage their systems. To satisfy the lesser needs of these limited-capability systems, software companies and independent developers create modified Linux kernels and applications designed for embedded devices. In addition to "home grown" embedded Linux kernels, companies such as Red Hat and projects such as uCLinux (www.uclinux.org) have developed Linux solutions for embedded systems. These modifications allow the kernel to execute in systems that do not support virtual memory, in addition to reducing resource consumption and enabling real-time execution. As developers continue to address these issues, Linux is becoming a viable choice for use in many embedded systems. 20.12.1 Symmetric Multiprocessing (SMP) Much of the effort to increase Linux's performance on servers focuses on improved support for SMP systems. Version 2.0 was the first stable kernel release to support SMP systems.227 Adding a global spin lock—called the big kernel lock (BKL)—was an early attempt at SMP support. When a process acquired the BKL in version 2.0, no process on any other processor could execute in kernel mode. Other processes were, however, free to execute in user mode. 228 The BKL enabled developers to serialize access to kernel data structures, which allowed the kernel to execute multiple processes concurrently in SMP systems. As one study showed, serializing access to the entire kernel meant that the system could not scale effectively to more than four processors per system.229 Locking the entire kernel is usually not required, because multiple processes can execute concurrently in kernel mode, provided they do not modify the same data structures. Linux kernel version 2.4 replaced most uses of the BKL with finegrained locking mechanisms. This change allows SMP systems running Linux to scale effectively to 16 processors.230 Fine-grained locks, although a performance enhancement, tend to make developing and debugging the kernel more difficult. These locks force developers to carefully code the acquisition of the appropriate locks at the appropriate times to avoid causing deadlock (for example, by writing code that attempts to acquire the same spin lock twice). As a result, the use of fine-grained locks has slowed kernel development in many subsystems, as would be expected with the increased software complexity.231 1012 Case Study: Linux Performance gains similar to those of large SMP systems have been achieved using alternative solutions, such as clustering (see Section 18.4, Clustering). Recall that a cluster of computer systems consists of several computers that cooperate to perform a common set of tasks. Such clusters of computers are typically connected by a dedicated, high-speed network. To perform work cooperatively using Linux systems, each system must run a modified kernel. Examples of features such a kernel might include are routines to balance workloads within the cluster and data structures to simplify remote interprocess communication, such as global process identifiers. If each machine in the Linux cluster contains a single processor, the complexity of fine-grained locks can be avoided, because only one kernel control path can execute at a time on each machine. Clusters can equal or outperform SMP systems. often at a lower cost of development and hardware. 232 For example, Beowulf clusters, a popular form of Linux clusters, have been used by NASA and the Department of Energy (DoE) to build high-performance systems at relatively low cost when compared to proprietary, multiprocessor supercomputer architectures.233 As the number of processors in high-end systems increases, buses that connect each processor to components such as memory become increasingly congested. Consequently, many system designers have implemented nonuniform memory access (NUMA) architectures to reduce the amount of bandwidth necessary to maintain high levels of performance in large multiprocessor systems. Recall from Section 15.4.2, Nonuniform Memory Access, that NUMA architectures divide a system into nodes, each of which provides high-performance interconnections between a set of processors, memory and/or I/O devices. The devices within a node are called local resources, while resources outside the node are called remote resources. Connections to remote resources are typically significantly slower than connections to local devices. To achieve high performance, the kernel must be aware of the layout of the NUMA system (i.e., the location and contents of each node) to reduce unnecessary internode access. When the kernel detects a NUMA system, it must initially determine its layout (i.e., which devices correspond to which nodes) so that it can better allocate resources to processes. Most NUMA systems provide architecture-specific hardware that indicates the contents of each node. The kernel uses this information to assign each device an integer value indicating the node to which it belongs. Most NUMA systems partition a single physical memory address space into regions corresponding to each node. To support this feature, the kernel uses a data structure to associate a range of physical memory addresses with a particular node. 234 To maximize performance, the kernel uses the layout of the NUMA system to allocate resources that are local to the node in which a process executes. For exampie, when a process is created on a particular processor, the kernel allocates the process local memory (i.e., memory that is assigned to the node containing the processor) using the data structure we mentioned. If a process were to subsequently 20.12 execute on a processor on a different node, its data would be stored in remote memory, leading to poor performance due to high memory access latency. Recall from Section 20.5.2, Process Scheduling, that the process scheduler will dispatch a process to the same processor (to improve cache performance) unless the number of processes running on each processor becomes unbalanced. To support NUMA systems, the load balancing routines attempt to migrate a process only to processors within the process's current node. In the case that a processor in another node is idle; however, the process scheduler will migrate a process to another node. Although this results in high memory access latency for the migrated process, overall system throughput increases due to increased resource utilization.235 Several other kernel components support the NUMA architecture. For example, when a process requests memory and available local memory is low, the kernel should swap out pages only in local memory. As a result, the kernel creates a kswapd thread for each node to perform page replacement. 236 Despite the kernel's broad support for the NUMA architecture, there are limitations to the current implementation. For example, if a process is migrated to a processor on a remote node, the kernel provides no mechanisms to migrate the process's pages to local memory (i.e., memory in the process's current node). Only when pages are swapped from disk does the kernel move the process's pages to its processor's local memory. Several development projects exist to improve kernel support for NUMA systems, and extensions are expected to be released in future versions of the kernel. 20.12.3 Other Scalability Features Developers have increased the size of several fields to accommodate the growing demands of computer systems. For example, the maximum number of users in a system increased from 16-bit field (65,536 users) to a 32-bit field (over four billion users). This is necessary for institutions such as large universities that have more than 100,000 users. Similarly, the number of tasks a system can execute increased from 32,000 to 4 million, which was necessary to support mainframe and other highend systems, which often execute tens or hundreds of thousands of threads. 237 Also, the variable that stores time, jiffies, has been increased from a 32-bit to a 64-bit value. This means that the value, which is incremented at every timer interrupt, will not overflow at the current timer frequency (1,000Hz) for over 2 billion billion years. Using a 32-bit number, jiffies would overflow after approximately 50 days with a timer frequency of l,000Hz.238, 239, 240 Fields related to storage also increased in size to accommodate large memories. For example, Linux can reference disk blocks using a 64-bit number, allowing the system to access 16 quintillion disk blocks—corresponding to exabytes (billions of gigabytes) of data. Linux also supports Intel's Physical Address Extension (PAE) technology, which allows systems to access up to 64GB of data (corresponding to a 36-bit address) using a 32-bit processor.241-242 Scalability 1013 1014 Case Study: Linux Prior to version 2.6, the kernel could not be preempted by a user process. However, the 2.6 kernel is preemptible, meaning that the kernel will be preempted if an event causes a high-priority task to be ready, which improves response times for realtime processes. To ensure mutual exclusion and atomic operations, the kernel disables preemption while executing a critical section. Finally, Linux includes support for several high-performance architectures, such as 64-bit processors (both the Intel Itanium processors, www.intel.com/products/server/processors/server/itanium2/, and AMD Opteron processors, www.amd.com/us-en/Processors/ ProductInformation/0,,30_118_8825,00.html) and Intel's HyperThreadine technology (see www.intel.com/info/hyperthreading/). 243-244 20.12.4 Embedded Linux Porting Linux to embedded devices introduces design challenges much different from those in SMP and NUMA systems. Embedded systems provide architectures with limited instruction sets, small memory and secondary storage sizes and devices that are not commonly found in desktops and workstations (e.g., touch-sensitive displays and device-specific input buttons). A variety of Linux distributions are tailored to meet the needs of embedded systems. Often, providers of embedded Linux distributions must implement hard realtime process scheduling. Examples of systems requiring real-time embedded device management include cell phones, digital video recorders (e.g., TiVO; www.tivo.com) and network gateways.245 To provide real-time execution in the Linux kernel, companies such as MontaVista Software (www.mvista.com) modify a few key components of the kernel. For example, developers must reduce scheduling overhead so that realtime process scheduling occurs quickly enough that the kernel meets real-time processes' timing constraints.The standard kernel's policy, although somewhat appropriate for real-time processes, is not sufficient for providing hard real-time guarantees (see Section 8.9, Real-Time Scheduling). This is because the Linux scheduler by default does not support deadlines. Embedded-device developers modify the scheduler to support additional priority levels, deadlines and lower scheduling latency.246 Other concerns specific to embedded systems also require modification to the Linux kernel. For example, some systems may include a relatively small amount of memory compared to desktop systems, so developers must reduce the size of the kernel footprint. Also, some embedded devices do not support virtual memory. As a result, the kernel must be modified to perform additional memory management operations (e.g., protection) in software.247 20.13 Security The kernel provides a minimal set of security features, such as discretionary access control. Authentication is performed outside the kernel by user-level applications such as login. This simple security infrastructure has been designed to allow system administrators to redefine access control policies, customize the way Linux authenticates users and specify encryption algorithms that protect system resources. 20.13 Security 20.17.1 Authentication By default, Linux authenticates its users by requiring them to provide a username and password via the login process. Each username is linked to an integer user ID. Passwords are hashed using the MD5 or DES algorithms, then stored in entries corresponding to user IDs in either the /etc/passwd or /etc/shadow file. [Note: Although DES was originally developed as an encryption algorithm, it can be used as a hash algorithm.] The choice of encryption algorithm and location of the password file can be modified by a system administrator. Unlike encryption algorithms, which can reverse the encryption operation using a decryption key, hash algorithms are not reversible. Consequently, Linux verifies a password entered at the login prompt by passing it through the hash algorithm and comparing it to the entry that corresponds to the user's ID number in the /etc/passwd or /etc/shadow file. 248 - 249 As discussed in Section 19.3.1, Basic Authentication, username and password authentication is susceptible to brute-force cracking, such as dictionary attacks. Linux addresses such problems by allowing system administrators to load pluggable authentication modules (PAMs). These modules can reconfigure the system at run time to include enhanced authentication techniques. For example, the password system can be strengthened to disallow terms found in a dictionary and require users to choose new passwords regularly. PAM also supports smart card, Kerberos and voice authentication systems.250 System administrators can use PAMs to select an authentication system that is most suitable for their environment without modifying the kernel or utility programs such as login. 251 20.13.2 Access Control Methods Linux secures system resources by controlling access to files. As described in Section 20.3, Linux Overview, the root user is given access to all resources in the system. To control how other users access resources, each file in the system is assigned access control attributes that specify file permissions and file attributes, as discussed in Section 20.7.3. In Linux, file permissions consist of a combination of read, write and/or execute permissions specified for three categories: user, group and other. The user file permissions are granted to the owner of the file. By default, a Linux file's owner is initially the user that created the file. Group permissions are applied if the user requesting the file is not the owner of the file, but is a member of group. Finally, other permissions are applied to users that are members of user neither nor group.252 File attributes are an additional security mechanism that is supported by some file systems (e.g., the ext2 file system). File attributes allow users to specify constraints on file access beyond read, write and execute. For example, the appendonly file attribute specifies that any changes to the file must be appended to the end of the file. The immutable file attribute specifies that a file cannot be modified (including renaming and deletion) or linked (i.e., referenced using symbolic or hard links).253, 254 1015 1016 Case Study: Linux Linux Security Modules (LSM) Framework In many environments, security provided by the default access control policy (i.e.. discretionary access control) is insufficient. Thus, the kernel supports the Linux security modules (LSM) framework to allow a system administrator to customize the access control policy for a particular system using loadable kernel modules. To choose a different access control mechanism, system administrators need only install the kernel module that implements that mechanism. The kernel uses hooks inside the access control verification code to allow an LSM to enforce its access control policy. As a result, an LSM is invoked only if a process has been granted access to a resource via the default access control policy. If a process is denied access by the default access control policy, the registered LSM does not execute, reducing the overhead caused by an LSM.255 One popular LSM is SELinux, developed by the National Security Agency (NSA). SELinux replaces Linux's default discretionary access control policy with a mandatory access control (MAC) policy (see Section 19.4.2, Access Control Models and Policies). Such a policy allows the system administrator to set the security rules for all files; these rules cannot be overridden by malicious or inexperienced users. The disadvantages of MAC policies result from the need for a greater number of complex rules. More information about the LSM framework and modules such as SELinux can be found at the official LSM Web site (lsm.imtnunnx.org). 256 Privilege Inheritance and Capabilities When a process is launched by a user, normally it executes with the same privileges as the user who launched it. It is sometimes necessary for users to execute applications with privileges other than those defined by their username and group. For example, many systems allow users to change their passwords using the passwd program. This program modifies the /etc/passwd or /etc/shadow file, which can be read by everyone, but written only with root privileges. To allow users to execute such programs, Linux provides the setuid and setgid permission bits. If the setuid permission bit for an executable file is set, the process that executes that file is assigned the same privileges as the owner of the file. Similarly, if the setgid permission bit for an executable file is set, the process that executes that file is assigned the same privileges as the group specified in the file attributes. Thus, users can modify the /etc/passwd file if the passwd program is owned by a user with root privileges and its setuid permission bit is set.257 Poorly written programs that modify the setuid or setgid bits can allow users access to sensitive data and system resources. To reduce the possibility of such a situation occurring, Linux provides the LSM Capabilities module to implement capabilities (see Section 19.4.3, Access Control Mechanisms). This allows Linux administrators greater flexibility in assigning access control privileges, such as the ability to assign privileges to applications rather than to users, which promotes finegrained security.258 20.13 Security 1017 20.13.3 Cryptography Although PAM and the LSM framework allow system administrators to create secure authentication systems and customized access control policies, they cannot protect data that is not controlled by the Linux kernel (e.g., data transmitted over a network or stored on disk). To enable users to access several forms of encryption to protect their data, Linux provides the Cryptographic API. Using this interface, processes can encrypt information using powerful algorithms such as DES, AES and MD5 (see Section f 9.2, Cryptography). The kernel uses the Cryptographic API to implement secure network protocols such as IPSec (see Section 19.10, Secure Communication Protocols). 259 The Cryptographic API also allows users to create secure file systems without modifying the existing file system's code. To implement such a file system, encryption is implemented using a loopback device (Fig. 20.28), which is a layer between the virtual file system and the existing file system (e.g., ext2). When the virtual file system issues a read or write call, control passes to the loopback device. If the VFS issues a read call, the loopback device reads the requested (encrypted) data from the underlying file system. The loopback device then uses the Cryptographic API to decrypt the data and returns that data to the VFS. Similarly, the loopback device Figure 20.28 | Loopback device providing an encrypted file system using the Cryptographic API. 1018 Case Study: Linux uses the Cryptographic API to encrypt data before transferring it to the file system. This technique can be applied to individual directories or the entire file system, so that data is protected even if an unauthorized user accesses the hard disk using another operating system.260 Web Resources www.kernel.org/ www.csn.ul.ie/~mel/projects/vm/ Official site for hosting the latest Linux kernel releases. It also includes a brief description of Linux and contains links to Linux resources. Provides documentation of the Linux version 2.4 virtual memory system, which remains largely intact in version 2.6. The documents include a comprehensive discussion of Linux virtual memory and a companion code commentary. lxr.linux.no/ Contains a cross-reference to recent releases of the Linux source code. Users can navigate the source code specific to an architecture and version number, search the source for identifiers and other text, and compare two different versions of the source side by side. The /Documentation directory provides links to files referenced in the chapter. loll.sou reeforge.net/linux/links/ Provides an index of categorized links to a selection of online Linux resources. Categories include kernel, documentation, distributions, security and privacy, graphics and software applications. www.linux.org/ Provides updated Linux information including news, development status and links to documentation, distributions, applications, source code and other resources. www.kernelnewbies.org/ Provides information for people new to the Linux kernel. Features include a glossary, FAQ, articles, documentation, useful scripts and mailing lists. www.spinics.net/linux/ Contains a large number of Linux resources, including links to information on kernels, security, embedded Linux and more. www.tldp.org/LDP/lki/index.html Part of the Linux Documentation Project hosted at www.tldp.org that discusses Linux kernel internals. The Linux Documentation Project provides many guides, HOWTOs and FAQs for those interested in the kernel, and for users and administrators. www.tldp.org/LDP/lkmpg/index.html Focuses on programming for the Linux environment, including details on programming kernel modules and adding entries to the proc file system. www.tldp.org/HOWTO/Module-HOWTO/ Describes how to program Linux loadable kernel modules (LKMs) and discusses the use and behavior of LKMs. www.linuxjournal.com/search.php?query=&topic=4 The Linux Journal Web site provides numerous articles on Linux. Be sure to search the "Linux Kernel" category. www.linuxsymposium.org/2003/ Provides information about the Linux Symposia, annual gatherings of the top Linux kernel developers from over 30 countries. lse.sourceforge.net/ Home to the Linux Scalability Effort, an organization dedicated to porting Linux to larger and more complex computer systems. www.linux-mag.com/ Online version of Linux Magazine, which provides articles. guides and tutorials for Linux systems. www.linuxdevices.com/ Provides articles and resources on Linux for embedded systems and discusses how Linux scales to devices such as cell phones and PDAs. www.linuxsecurity.com/ Contains articles, documentation and other links to information regarding Linux security for both users and developers. kernelnewbies.org/documents/kdoc/kernel-locking/lklockingguide.html Contains information written by a kernel developer about kernel locks and semaphores. www.plig.org/xwinman/index.html Describes the window managers available for the X Window Manager as well as some of the desktop environments that run on X. user-mode-linux.sourceforge.net/ Explains how to download, install and use User-Mode Linux (UML). lsm.immunix.org Provides links to documentation, downloads and mailing lists for the Linux Security Modules project. Key Terms 1019 Key Terms access control attribute (Linux)—Specifies the access rights for processes attempting to access a particular resource. active list (Linux) —Scheduler structure that

contains processes that will control the processor at least once during the current epoch. active page (Linux)—Page of memory that will not be replaced the next time pages are selected for replacement. active state (Linux)—Task state describing tasks that can compete for execution on a processor during the current epoch. append-only file attribute (Linux)—File attribute that limits users to appending data to existing file contents. architecture-specific code—Code that specifies instructions unique to a particular architecture. Beowulf cluster (Linux)—High-performance parallel-processing system consisting of a cluster of computers each running the Beowulf modification to the Linux kernel. big kernel lock (BKL) (Linux) — Global spin lock that served as an early implementation of SMP support in the Linux kernel. binary buddy algorithm (Linux)—Algorithm that Linux uses to allocate physical page frames. The algorithm maintains a list of groups of contiguous pages; the number in each group is a power of two. This facilitates memory allocation for processes and devices that require access to contiguous physical memory. bio structure (Linux)—Structure that simplifies block I/O operations by mapping I/O requests to pages. block allocation bitmap (Linux)—Bitmap that tracks the usage of blocks in each block group. block group (Linux) —Collection of contiguous blocks managed by groupwide data structures so that related data blocks, inodes and other file system metadata are contiguous on disk. bottom half of an interrupt handler (Linux)—Portion of interrupt-handling code that can be preempted. bounce buffer (Linux)—Region of memory that allows the kernel to map data from the high memory zone into memory that it can directly reference. This is necessary when the system's physical address space is larger than kernel's virtual address space. capability—Security mechanism that assigns access rights to a subject (e.g., a process) by granting it a token for an object (i.e., a resource). This enables administrators to specify and enforce fine-grained access control. code freeze (Linux)—Point at which no new code should be added to the kernel unless the code fixes a known bug. core file (Linux)—File that contains the execution state of a process, typically used for debugging purposes after a process encounters a fatal exception. Cryptographic API (Linux)—Kernel interface through which applications and services (e.g., file systems) can encrypt and decrypt data. datagram socket—Socket that uses the UDP protocol to transmit data. deadline scheduler (Linux)—Disk scheduling algorithm that eliminates indefinite postponement by assigning deadlines by which I/O requests are serviced. daemon (Linux)—Process that runs periodically to perform system services. deactivated process (Linux)—Process that has been removed from the run queues and can therefore no longer contend for processor time. dcache (directory entry cache) (Linux)—Cache that stores directory entries (dentries), which enables the kernel to quickly map file descriptors to their corresponding inodes. default action for a signal handler (Linux)—Predefined signal handler that is executed in response to a signal when a process does not specify a corresponding signal handler. dentry (directory entry) (Linux) — Structure that maps a file to an inode. desktop environment—GUI layer above a window manager that provides tools, applications and other software to improve system usability. device class—Group of devices that perform similar functions. device special file (Linux)—Entry in the /dev directory that provides access to a particular device. direct I/O—Technique that performs I/O without using the kernel's buffer cache. This leads to more efficient memory utilization in database applications, which typically maintain their own buffer cache. cache-cold process—Process that contains little, if any, of its data or instructions in the cache of the processor to which it will be dispatched. discretionary access control—Access control policy that specifies the owner of a file as the user that can assign access rights to that file. cache-hot process—Process that contains most, if not all, of its data and instructions in the cache of the processor to which it will be dispatched. distribution (Linux)—Software package containing the Linux kernel, user applications and/or tools that simplify the installation process. 1020 Case Study: Linux DMA memory (Linux)—Region of physical memory between zero and 16MB that is typically reserved for kernel bootstrapping code and legacy DMA devices. doubly indirect pointer—Inode pointer that locates a block of (singly) indirect pointers. effective priority (Linux)—Priority assigned to a process by adding its static priority to its priority boost or penalty. epoch (Linux)—Time during which all processes move from the scheduler's active list to its expired list. This ensures that processes are not indefinitely postponed. expired list (Linux)—Structure containing processes that cannot contend for the processor until the next epoch. Processes are placed in this list to prevent others from being indefinitely postponed. To quickly begin a new epoch, this list becomes the active list. expired state (Linux)—Task state that prevents a task from being dispatched until the next epoch. ext2 inode (Linux) — Structure that stores information such as file size, the location of a file's data blocks and permissions for a single file or directory in an ext2 file system. ext2fs (Linux)—Popular inode-based Linux file system that enables fast access to small files and supports large file sizes. feature freeze (Linux)—State of kernel development during which no new features should be added to the kernel, in preparation for a new kernel release. FIFO (Linux)—Named pipe that enables two unrelated processes to communicate via the producer/consumer relationship using a page-size buffer. file attribute—File metadata that implements access control information, such as whether a file is append-only or immutable, that cannot be specified using standard Linux file permissions. file permission—Structure that determines whether a user may read, write and or execute a file. group descriptor (Linux)—Structure that records information regarding a block group, such as the locations of the inode allocation bitmap, block allocation bitmap and inode table. high memory (Linux)—Region of physical memory (which begins at 896MB on the IA-32 architecture) beginning at the largest physical address that is permanently mapped to the kernel's virtual address space and extending to the limit of physical memory (64GB on Intel Pentium 4 processors). Because the kernel must perform expensive operations to map pages in its virtual address space to page frames in high memory, most kernel data structures are not stored in high memory. hot swappable device—Device that can be added to, or removed from, a computer while it is running. hook —Software feature that enables developers to add features to an existing application without modifying its source file. An application uses a hook to call a procedure that can be defined by another application. immutable attribute (Linux)—Attribute specifying that a file can be read and executed, but cannot be copied, modified or deleted. inactive list (Linux)—See expired list. inactive page (Linux)—Page in main memory that can be replaced by an incoming page. indirect pointer—Inode pointer that points to a block of inode pointers. inode (Linux)—Structure that describes the location of data blocks corresponding to a file, directory or link in a file system. In the VFS, this structure represents any file in the system. An ext2 inode represents a file in the ext2 file system. inode allocation bitmap (Linux)—Bitmap that records a block group's inode usage. inode cache (Linux) — Cache that improves inode lookup performance. inode table (Linux) — Structure that contains an entry for each allocated inode in a block group. kernel control path (Linux) —A kernel execution context that may perform operations requiring mutual exclusive access to kernel data structures. kernel semaphore (Linux) — Semaphore implemented by the kernel to provide mutual exclusion. kernel thread (Linux)—Thread that executes kernel code. ksoftirqd (Linux)—Daemon that schedules software interrupt handlers when softirq load is high. kswapd (Linux)—Daemon that swaps pages to disk. Linux security modules (LSM) framework (Linux)—Framework that allows system administrators to specify the access control mechanism employed by the system. Linux Standard Base (Linux)—Project that aims to specify a standard Linux interface to improve application portability between kernel versions (and distributions). loadable kernel module (Linux)—Software that can be integrated into the kernel at runtime. load balancing—Operation that attempts to evenly distribute system load between processors in the system. loopback device (Linux)—Virtual device that enables operations to be performed on data between layers of a system service (e.g., the file system). Key Terms 1021 mandatory access control—Access control policy that relegates assignment of access rights to the system administrator. major device identification number (Linux)—Value that uniquely identifies a device in a particular device class. The kernel uses this value to determine a device's driver. major version number (Linux)—Value that uniquely identifies a significant Linux release. memory dump (Linux)—Action that generates a core file before terminating a process. memory footprint (Linux) —Size of unswappable memory consumed by the kernel. memory pool (Linux) —Region of memory reserved by the kernel for a process to ensure that the process's future requests for memory are not denied. merge I/O requests (Linux)—To combine two I/O requests to adjacent locations on disk into a single request. message queue (Linux)—Structure that stores messages that have yet to be delivered to processes. message queue descriptor (Linux)—Structure that stores data regarding a message queue. message—IPC mechanism that allows data to be transmitted by specifying a message type and variable-length field of data. minor device identification number (Linux)—Value that uniquely identifies devices that are assigned the same major number (e.g., a hard drive partition). minor version number (Linux)—Value that identifies successive stable (even) and development (odd) versions of the Linux kernel. mount—Insert a file system into a local directory structure. named pipe (Linux)—Pipe that can be accessed via the directory tree, enabling processes that are not parent and child to communicate using pipes. See also pipe. netfiiter framework (Linux)—Mechanism that allows kernel modules to directly inspect and modify packets. This is useful for applications such as firewalls, which modify each packet's source address before the packet is transmitted. nice value (Linux)—Measure of a process's scheduling priority. Processes with a low nice value receive a greater share of processor time than other processes in the system and are therefore "less nice" to other processes in the system. normal memory (Linux)—Physical memory locations beyond 16MB that the kernel can directly map to its virtual address space. This region is used to store kernel data and user pages. package (Linux)—Portion of a distribution containing an application or service. Users can customize their Linux systems by adding and removing packages. page cache (Linux)—Cache storing pages of data from disk. When a process requests data from disk, the kernel first determines if it exists in the page cache, which can eliminate an expensive disk I/O operation. page global directory (Linux)—Virtual memory structure that stores addresses of second-level page-mapping tables. page middle directory (Linux)—Virtual memory structure that stores addresses of third-level page-mapping tables (also called page tables). page table (Linux)—Virtual memory structure that contains direct mappings between virtual page numbers and physical page numbers. PID (process identifier)—Integer that uniquely identifies a process. pipe—Interprocess communication mechanism that uses a page of memory as a first-in-first-out buffer to transfer information between processes. pipe buffer (Linux)—Page of data that is used to buffer data written to a pipe. pluggable authentication module (PAM) (Linux)—Module that can be installed at runtime to incorporate enhanced authentication techniques in the Linux system. port of Linux—Version of the Linux kernel that is modified to support execution in a different environment. preemption lock counter (Linux)—Integer that is used to determine whether code executing in kernel mode may be preempted. The value of the counter is incremented each time a kernel control path enters a critical section during which it cannot be preempted. priority array (Linux)—Structure within a run queue that stores processes of the same priority. procfs (proc file system) (Linux)—File system built directly into the kernel that provides real-time information about the status of the kernel and processes, such as memory utilization and system execution time. ramfs (Linux)—Region of main memory treated as a block device. The ramfs file system must be formatted before use. reader/writer lock (Linux)—Lock that allows multiple threads to concurrently hold a lock when reading from a resource, but only one thread to hold a lock when writing to that resource. real-time signal (Linux) — Signal implementation that helps to implement a real-time system by ensuring that no signals are dropped. request list (Linux)—Structure that stores pending I/O requests. This list is sorted to improve throughput by reducing seek times. 1022 Case Study: Linux reverse mapping (Linux)—Linked list of page table entries that reference a page of memory. This facilitates updating all PTEs corresponding to a shared page that is about to be replaced. root user (Linux)—See superuser. run queue (Linux) —List of processes waiting to execute on a particular processor. second extended file system (ext2fs) (Linux) — See ext2fs. semaphore array (Linux) —Linked list of semaphores that protect access to related resources. seqlock (Linux)—Mutual exclusion structure that combines a spin lock with a sequence counter. Seqlocks are used by interrupt handlers, which require immediate exclusive access to data. Single UNIX Specification—Specification (created by The Open Group) to which an operating system must conform to earn the right to display the UNIX trademark (see www.unix.org/version3/overview.html). slab (Linux)—Page of memory that reduces internal fragmentation due to small structures by storing multiple structures smaller than one page. slab allocator (Linux)—Kernel entity that allocates memory for objects placed in the slab cache. slab cache (Linux) —Cache that stores recently used slabs. socket—Interprocess communication mechanism that allows processes to exchange data by establishing direct communication channels. Enables processes to communicate over a network using read and write calls. socketpair (Linux)—Pair of connected, unnamed sockets that can be used for bidirectional communication between processes on a single system. socket address—Unique identifier for a socket. software interrupt handler (Linux)—Interrupt-handling code that can be performed without masking interrupts and can therefore be preempted. softirq (Linux) — Software interrupt handler that is reentrant and not serialized, so it can be executed on multiple processors simultaneously. source tree (Linux) —Structure that contains source code files and directories. Provides a logical organization to the monolithic Linux kernel. spin lock—Lock that provides mutually exclusive access to critical sections. When a process holding the lock is executing inside its critical section, any process concurrently executing on a different processor that attempts to acquire the lock before entering its critical section is made to busy wait. starvation limit (Linux)—Time at which high-priority processes are placed in the expired list to prevent low-priority processes from being indefinitely postponed. static priority level (Linux) — Integer value assigned to a process when it is created that determines its scheduling priority. stream socket—Socket that transfers data using the TCP protocol. superblock (Linux) —Block containing information regarding a mounted file system, such as the root inode and other information that protects the file system's integrity. superuser (root user) (Linux) —User that may perform restricted operations (i.e., those that may damage the kernel and/or the system). swap cache (Linux) — Cache of page table entries that describes whether a particular page exists in the system swap file on secondary storage. If a page table entry is present in the swap cache, then its corresponding page exists in the swap file and does not need to be written to the swap file. system file system (sysfs) (Linux) —File system that allows processes to access structures defined by the unified device model. task (Linux)—User execution context (i.e., process or thread) in Linux. tasklet (Linux)—Software interrupt handler that cannot be executed simultaneously on multiple processors. Tasklets are used to execute nonreentrant bottom halves of interrupt handlers. time slice (Linux)—Another term for quantum. tmpfs (temporary file system) (Linux)—Similar to ramfs, but does not require formatting before use, meaning that the system can store files in the tmpfs without the organizational overhead typical of most file systems. top half of an interrupt handler (Linux)—Nonpreemptible portion of interrupt-handling code that performs the minimum work required to acknowledge an interrupt before transferring execution to the preemptible bottom-half handler. triply indirect pointer—Pointer in an inode that locates a block of doubly indirect pointers. unified device model (Linux)—Internal device representation that relates devices to device drivers, device classes and system buses. The unified device model simplifies power management and hot swappable device management. unmap a page (Linux)—To update page table entries to indicate that the corresponding page is no longer resident. Exercises User-Mode Linux (UML) (Linux)—Linux kernel that executes as a user process within a host Linux system. virtual file system (Linux) — Interface that provides users with a common view of files and directories stored across multiple heterogeneous file systems. virtual memory area (Linux) —Structure that describes a contiguous region of a process's virtual address space so that the kernel can perform operations on this region as a unit. 1023 window manager (Linux)—Application that controls the placement, appearance, size and other attributes of windows in a GUI. zone (memory) (Linux)—Region of physical memory. Linux divides main memory in the low, normal and high zones to allocate memory according to the architectural limitations of a system. zone allocator—Memory subsystem that allocates pages from the zone to which it is assigned. Exercises 20.1 [Section 20.4.1, Hardware Platforms] Describe several applications of User-Mode Linux (UML). 20.2 [Section 20.4.2, Loadable Kernel Modules] Why is it gen : erally unsafe to load a kernel module written for kernel versions other than the current one? 20.3 [Section 20.5.1, Process and Thread Organization] Which threading model (see Section 4.6, Threading Models) can threads created using the clone system call implement? How do Linux threads differ from traditional threads? Discuss the benefits and drawbacks of this implementation. 20.4 [Section 20.5.2, Process Scheduling] How are the Linux process scheduler's run queues similar to multilevel feedback queues (see Section 8.7.6, Multilevel Feedback Queues)? How are they different? 20.5 [Section 20.5.2, Process Scheduling] Why should the Linux process scheduler penalize processor-bound processes? 20.6 [Section 20.5.2, Process Scheduling] Why does Linux prevent users without root privileges from creating real-time processes? 20.7 [Section 20.6.1, Memory Organization] Why does the kernel allocate page frames to processes from normal and high memory before allocating pages from DMA memory? 20.8 [Section 20.6.1, Memory Organization] What are the benefits and drawbacks of embedding the kernel virtual address space in each process's virtual address space? 20.9 [Section 20.6.1, Memory Organization] The x86-64 architecture uses four levels of page tables; each level contains 512 entries (using 64-bit PTEs). However, the kernel provides only three levels of page tables. Assuming that each PTE points to a 4KB page, what is the largest address space the kernel can allocate to processes in this architecture? 20.10 [Section 20.6.2, Physical Memory Allocation and Deallocation] How does the kernel reduce the amount of internal fragmentation caused by allocating memory to structures that are much smaller than a page? 20.11 [Section 20.6.3, Page Replacement] When is the kernel unable to immediately free a page frame to make room for an incoming page? 20.12 [Section 20.6.4, Swapping] When nonresident pages are retrieved from the backing store in Linux, the memory manager retrieves not only the requested page but also up to eight pages contiguous to it in the running process's virtual address space. Identify the type of prepaging implemented and describe the benefits and disadvantages of such a policy. 20.13 [Section 20.7.1, Virtual File System] List at least four different objects a VFS file can represent and describe the usage of each object. 20.14 [Section20.7.1, Virtual File System] What role does the dentry object serve in the Linux VFS? 20.15 [Section 20.7.2, Virtual File System Caches] Is it possible for a file's inode to exist in the inode cache if its dentry is not located in the dentry cache? 20.16 [Section 20.7.3, Second Extended File System (ext2fs)] Compare and contrast the VFS and ext2 representation of an inode. 20.17 [Section20.7.3, Second Extended File System (ext2fs)] Why do most file systems maintain redundant copies of their superblock throughout the disk? 20.18 [Section 20.7.3, Second Extended File System (ext2fs)] What are the primary contents of a block group and what purpose do they serve? 20.19 [Section 20.7.4, Proc File System] In what ways does the proc file system differ from a file system such as ext2fs? 20.20 [Section 20.8.1, Device Drivers] Explain the concept and usage of device special files. 20.21 [Section 20.8.3, Block Device I/O] Identify two mechanisms employed by the Linux block I/O subsystem that improve performance. Discuss how they improve performance and how they may, if ever, degrade performance. 1024 Case Study: Linux 20.22 [Section 20.8.4, Network Device I/O] Compare and contrast networking I/O operations and block/character I/O operations. 20.23 [Section 20.8.5, Unified Device Model] How does the unified device model facilitate kernel support for hot swappable devices? 20.24 [Section 20.8.6, Interrupts] Why does the networking subsystem employ softirqs to process packets? 20.25 [Section 20.9.1, Spin Locks] What occurs if multiple kernel control paths concurrently attempt to acquire the same spin lock? 20.26 [Section 20.10.1, Signals] What problems can result from dropping a signal while a process handles a signal of the same type? 20.27 [Section 20.10.4, Message Queues] What could happen if a message's size is greater than a message queue's buffer? 20.28 [Section 20.10.6, System V Semaphores] Name a potential problem that occurs when processes are made to wait on a semaphore array. 20.29 [Section 20.11.2, Netfilter Framework and Hooks] What purpose does the netfilter framework serve? 20.30 [Section 20.12.2, Nonuniform Memory Access (NUMA)] Which kernel subsystems were modified to support NUMA? 20.31 [Section 20.13.1, Authentication] How does Linux protect user passwords from attackers, even if the attacker acquires the password file? How can this be circumvented? 20.32 [Section 20.13.2, Access Control Methods] Why might it be dangerous to set the setuid or setgid bits for an executable file? Recommended Reading Linux development is an ongoing process that includes contributions from developers worldwide. Because Torvalds frequently releases new kernel versions, some documentation can be outdated as soon as it is published. Hence, the most current information is usually found on the Web. Useful resources include the magazines Linux Journal and Linux Magazine. Monthly issues cover a variety of Linux topics such as desktop and server applications, programming and, of course, the kernel. Selected articles and information about subscribing can be found at www.linuxjournal.com and www.linux-mag.com, respectively. Readers looking to dig deeper into the Linux kernel should consider the book Understanding the Linux Kernel, 2nd ed., by Bovet and Cesati.261 This book includes in-depth discussions of nearly all kernel subsystems. It explains kernel version 2.4—and, while some material has changed in kernel 2.6. much of the book's content is still relevant. Another notable book is Linux Device Drivers, 2nd ed.. by Rubini and Corbet. 262 The title can be misleading, as the book also explains a number of kernel subsystems of concern to device driver developers. It provides an in-depth explanation of the I/O subsystem as well as information about synchronization, memory management and networking. Like most Linux-related literature, the book is somewhat outdated compared to the most recent release of the kernel. The second edition discusses kernel version 2.4. Works Cited 1. Kuwabara, K., "Linux: A Bazaar at the Edge of Chaos," First Monday, Vol. 5, No. 3, March 2000. 2. "Linux History," viewed July 8, 2003, . 3. "Linux History," viewed July 8, 2003, . 4. Torvalds, L., . 5. Quinlan, D., "The Past and Future of Linux Standards," Linux Journal, Issue 62, June 1999. 8. Wheeler, D., "More Than a Gigabuck: Estimating GNU/Linux's Size," June 30, 2001 (updated July 29, 2002), version 1.07. . 9. McHugh, J., "Linux: The Making of a Global Hack," Forbes Magazine, August 1998. 10. Pranevich, J., "The Wonderful World of Linux 2.2," January 26. 1999, . n. McCarty, B., and P. McCarty, "Linux 2.4," January 2001. . 6. Linux README, . 12. McCarty, B., and P. McCarty, "Linux 2.4," January 2001. . 7. Wilburn, G., "Which Linux OS Is Best for You," Computing Canada, August 1999, p. 26. 13. "Whatever Happened to the Feature Freeze?" December 18. 2002, . Works Cited 1025 14- Index, . 15. "LWN Distributions List," updated May 2003, <old.lwn.net/Distributions>. 16. . 17. . 18. . 19. . 20. . 21. . 22. 23. 24. 25. . . slashdot.org/askslashdot/99/03/07/1357235.shtml. Casha, R., "The Linux Terminal —A Beginners' Bash," November 8,2001, . 26. "CG252-502 X Windows: History of X," modified June 19,1996, . 44- SchlapBach, A., "Linux Process Scheduling," May 2, 2000, . 45. Aivazian, T., "Linux Kernel 2.4 Internals," August 23, 2001, . 46. Walton, S., "Linux Threads Frequently Asked Questions," January 21,1997, . 47. McCracken, D., "POSIX threads and the Linux Kernel," Proceedings of the Ottawa Linux Symposium, 2002, p. 332. 48. Arcomano, R., "KernelAnalysis-HOWTO," June 2, 2002, . 49. Walton, S., "Linux Threads Frequently Asked Questions" . 50. Drepper, U., and I. Molnar, "The Native POSIX Thread Library for Linux," January 30, 2003, . 51. Molnar, I., Announcement to Linux mailing list, . 27. Manrique, D., "X Window System Architecture Overview HOWTO," 2001, . 28. "The Single UNIX Specification, Version 3 —Overview," modified January 27,2002, . 52. Cross-Referencing Linux, <lxr.linux.no/source/include/asm-i386/param.h?v=2.6.0-test7#L5> and <lxr.linux.no/source/kernel/sched.c?v=2.6.0-test7#L1336>. 53. Cross-Referencing Linux, scried.c?v=2. 5. 56>. . 54. Cross-Referencing Linux, scried.c?v=2.5.56>. . 33. Linux kernel source code, version 2.5.56, . 34. "LinuxHQ:Distribution Links," . 57. Linux kernel source code, version 2.5.75, <lxr.linux.no/source/kernel/sched.c?v=2.5.75>. 35. "The User-Mode Linux Kernel Home Page," July 23, 2003, . 36. Dike, X, "A User-Mode Port of the Linux Kernel," August 25, 2000, . 58. Linux kernel source code, version 2.5.75, <lxr.linux.no/source/kernel/sched.c?v=2.5.75>. 37- Rusling, D., "The Linux Kernel," 1999, . 38. Welsh, M., "Implementing Loadable Kernel Modules for Linux," Dr. Dobb's Journal, May 1995, . 39. Henderson, B., "Linux Loadable Kernel Module HOWTO," May 2002,. 40. Henderson, B., "Linux Loadable Kernel Module HOWTO," May 2002,. 41. Petersen, K, "Kmod: The Kernel Module Loader," Linux kernel source file, Linux/Documentation/kmod.txt . 59. Linux kernel source code, version 2.6.0-test2, <lxr.linux.no/source/arch/i386/mm/pageattr.c?v=2.6.0-test2>. 60. Linux kernel source code, version 2.5.75, . 61. Eranian, S., and David Mosberger, "Virtual Memory in the IA-64 Linux Kernel," informIT.com, November 8, 2002, . 42. Aivazian, T, "Linux Kernel 2.4 Internals," August 23, 2001, . 62. Rusling, D., "The Linux Kernel," 1999, . 63. Gorman, M., "Understanding the Linux Virtual Memory Manager," . 43- Rusling,D.,"The Linux Kernel," 1999, . 64. Linux kernel source code, version 2.5.75, . 1026 Case Study: Linux 65. Linux kernel source code, version 2.6.0-test2, . 66. Linux source code, . 67. Van Riel, R., "Page Replacement in Linux 2.4 Memory Management" . 68. Gorman, M., "Understanding the Linux Virtual Memory Manager," . 69. Van Riel, R., "Page Replacement in Linux 2.4 Memory Management" . 70. Linux kernel source code, version 2.5.56, <lxr.linux.no/source/include/linux/mmzone.h?v=2.5.56>. 71. Linux kernel source code, version 2.5.75, /Documentation/ block/biodoc.txt. 72. Bovet, D., and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001. 73. Gorman, M., "Understanding the Linux Virtual Memory Manager," . 74. Rusling, D., "The Linux Kernel," 1999 . 75. Knowlton, K. C, "A Fast Storage Allocator," Communications of the ACM, Vol. 8, No. 10, October 1965, pp. 623-625. 76. Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, MA, 1968, pp. 435455. 77. Rusling, D., "The Linux Kernel," 1999, . 78. Rusling, D., "The Linux Kernel," 1999, . 79. Nayani, A.; M. Gorman; and R. S. de Castro, "Memory Management in Linux: Desktop Companion to the Linux Source Code," May 25, 2002, . 80. Gorman, M., "Slab Allocator," . 81. "Driver Porting: Low-Level Memory Allocation," LWN.net, February 2003, . 82. Knapka, X, "Outline of the Linux Memory Management System," . 83. Knapka, J., "Outline of the Linux Memory Management System," . 84. Arcangeli, A., "Le novita' nel Kernel Linux," December 7, 2001, . 124. Appleton, R., "A Non-Technical Look Inside the Ext2 File System," Linux Journal, August 1997, . 125. Bowden,T; B. Bauer; and J. Nerin, "The /proc Filesystem," Linux kernel source file, Linux/Documentation/filesystems/proc.txt. 126. Rubini, A., "The Virtual File System in Linux," Linux Journal, May 1997, . 127. Linux kernel source code, version 2.5.75, . 128. Mouw, E., "Linux Kernel Procfs Guide," June 2001, . 129. Rusling, D., "Hie Linux Kernel," 1999, . 130. Rusling, D., "The Linux Kernel," 1999, . 131. "Linux Allocated Devices," . 132. <www.lanana.org/docs/device-list/devices.txt>. 133. Rubini, A., and J. Corbet, Linux Device Drivers, O'Reilly, 2001, pp. 55-57. 134. Matia, F, "Kernel Korner: Writing a Linux Driver," Linux Journal, April 1998, . 135. "The HyperNews Linux KHG Discussion Pages, Device Driver Basics," December 30, 1997, <users.evtek.fi/~tk/rt_html/ASICS.HTM>. 136. Rusling, D.,"The Linux Kernel," 1999, 162. Rubini, A., and J. Corbet, Linux Device Drivers, O'Reilly, 2001, pp. 445-448. 163. Corbet, X, "Porting Drivers to the 2.5 Kernel," Proceedings of the Linux Symposium, 2003, p. 149. 164. "Universal Serial Bus Specification," Compaq, Hewlitt-Packard, Intel, Lucent, Microsoft, NEC, Phillips, rev. 2.0, April 27,2002, p. 18. 165. Corbet, X, "Driver Porting: Device Model Overview," , May 2003. 166. Corbet, X, "Driver Porting: Device Model Overview," , May 2003. 167. Mochel, P., "Sysfs—The Filesystem for Exporting Kernel Objects," Linux kernel source code, version 2.5.75, Documentation/filesystems/sysfs.txt, January 10,2003. 168. Linux kernel source code, version 2.5.75, Documentation/ driver-model/overview.txt. 169. Linux kernel source code, version 2.5.75, . 170. Linux kernel source code, version 2.5.75, Documentation/ driver-model/bus.txt. 171. Linux kernel source code, version 2.5.75, Documentation/ driver-model/class.txt. 172. Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, "Advanced Configuration and Power Management," rev. 2.0b, October 11,2002,p.26. 173. Mochel, P., "Linux Kernel Power Management," Proceedings of the Linux Symposium, 2003, , pp. 344,347. 174. Russell, P., "Unreliable Guide to Hacking the Linux Kernel," 2000, . 175. Intel Corporation, IA-32 Intel Architecture Software Developer's Manual, Vol. 3, System Programmer's Guide, 2002, pp. 5-32 176. Russell, P., "Unreliable Guide to Hacking the Linux Kernel," 2000, . 177. Gatliff, W„ "The Linux Kernel's Interrupt Controller API," 2001, . 178. Linux kernel source code, version 2.5.75, . 179. Linux kernel source code, version 2.5.75, . 180. Oatliff, W., "The Linux Kernel's Interrupt Controller API," 2001. . 181. Linux kernel source code, version 2.5.75, . 182. Rubini, A., and J. Corbet, Linux Device Drivers, O'Reilly, 2001. p. 19. 183. Bovet, D., and M. Cesati, Understanding the Linux Kernel O'Reilly, 2001, pp. 300-301,305-306,523-532,545. 184. Love, R., "Kernel Korner: Kernel Locking Techniques," Linux Journal, August 2002, . 185. Love, R., "Kernel Korner: Kernel Locking Techniques," Linux Journal, August 2002, . 186. Torvalds, L., "/Documentation/spinlocks.txt," Linux kernel source code, version 2.5.75, . 187. Russell, P., "Unreliable Guide to Locking," 2000, . 188. Russell, P., "Unreliable Guide to Locking," 2000, . 189. Torvalds, L., "/Documentation/spinlocks.txt," Linux source code, version 2.5.75, . kernel 190. "Driver Porting: Mutual Exclusion with Seqlocks," . 191. "Driver Porting: Mutual Exclusion with Seqlocks," . 192. Bovet, D., and M. Cesati, Understanding the Linux Kernel O'Reilly, 2001, pp. 305-306. 193. Love, R., "Kernel Korner: Kernel Locking Techniques," Linux Journal, August 2002, . 194. Love, R., "Kernel Korner: Kernel Locking Techniques," Linux Journal, August 2002, . 195. Bar, M., "Kernel Korner: The Linux Signals Handling Model," Linux Journal, May 2000, . 196. Linux kernel source code, version 2.5.75, . 197. Linux kernel source code, version 2.5.75, 198. Troan, E., "A Look at the Signal API," Linux Magazine, January 2000, . 199. Bovet, D., and M. Cesati, Understanding the Linux Kernel O'Reilly, 2001, p. 253. 200. Bar, M., "Kernel Korner: The Linux Signals Handling Model." Linux Journal, May 2000, . Works Cited 1029 201. Rusling, D., "The Linux Kernel," 1999, . 202. Bovet, D., and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001, p. 253. 203. Linux kernel source code, version 2.6.0-test2, signal.c, line 38 204. Bovet, D., and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001, pp. 524-532. 205. Chelf, B., "Pipes and FIFOs," Linux Magazine, January 2001, . 206. Chelf, B., "Pipes and FIFOs," Linux Magazine, January 2001, . 207. Free Software Foundation, "The GNU C Library," 1998, . 208. Sechrest, S., "An Introductory 4.4BSD Interprocess Communication Tutorial," <docs.freebsd.org/44doc/psd/20.ipctut/paper.html>. 209. Free Software Foundation, "The GNU C Library," 1998, . 210. Sechrest, S., "An Introductory 4.4BSD Interprocess Communication Tutorial," <docs.freebsd.org/44doc/psd/20.ipctut/paper.html>. 211. Aivazian, T, "Linux Kernel 2.4 Internals," August 23, 2001, . 212. Bovet, D., and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001, p. 545. 213. Aivazian, T, "Linux Kernel 2.4 Internals," August 23, 2001 . 214. Goldt, S., et al., "Shared Memory," The Linux Programmer's Guide, , version 0.4, March 1995, and Linux source 2.5.56. 215. Linux man page: shm_open, . 216. Linux source, <lxr.linux.no/source/Documentation/file systems/tmpfs.txt?v=2.5.56> and <lxr.linux.no/source/mm/shmem.c?v=2.5.56> 217. Linux source, <lxr.linux.no/source/Documentation/filesystems/tmpfs.txt?v=2.5.56> and <lxr.linux.no/source/mm/shmem.c?v=2.5.56>. 218. Aivazian, T, "Linux Kernel 2.4 Internals," August 23, 2001, . 219. Linux kernel source code, version 2.5.75, . 220. Aivazian, T, "Linux Kernel 2.4 Internals," August 23, 2001, . 221. Cox, A., "Network Buffers," . 222. Linux kernel source code, version 2.5.75, . 223. Welte, H., "The Journey of a Packet Through the Linux 2.4 Network Stack," October 14, 2000, . 224. Linux kernel source code, version 2.5.75, . 225. Dobbelaere, J., "Linux Kernel Internals: IP Network Layer," 2001,. 226. Welte, H., "The Netfilter Framework in Linux 2.4," September 24, 2000, . 227. Schmidt, J., "Symmetrical Multiprocessing with Linux," 1999, . 228. Tumenbayer, E., et al., "Linux SMP HOWTO," July 9, 2002, . 229. Intel Corporation, "Linux Scalability: The Enterprise Question," 2000, . 230. Bergmann, K., "Linux for z/Series Performance Overview," April 3, 2002, . 231. McVoy, L., "SMP Scaling Considered Harmful," July 22, 1999, . 232. McVoy, L., "SMP Scaling Considered Harmful," July 22, 1999, . 233. Merkey, P., "Beowulf History," , viewed July 21,2003. 234. Linux kernel source code, version 2.5.75, . 235. Dobson, M.; P. Gaughen; M. Hohnbaum; and E. Focht, "Linux Support for NUMA Hardware," Proceedings of the Linux Symposium, 2003, pp. 181-195. 236. Dobson, M.; P. Gaughen; M. Hohnbaum; and E. Focht, "Linux Support for NUMA Hardware," Proceedings of the Linux Symposium, 2003, pp. 181-195. 237. Linux kernel source code, version 2.6.0-test7, <lxr.linux.no/source/include/linux/threads.h?v=2.6.0-test7#L33>. 238. Pranevich, J., "The Wonderful World of Linux 2.6," July 13,2003, . 239. Linux kernel source code, version 2.5.75, . 240. Corbet, J., "Driver Porting: Timekeeping Changes," February 2003, . 241. Pranevich, J., "The Wonderful World of Linux 2.6," July 13,2003, . 242. Linux source kernel version 2.5.75 . 243. J. Pranevich, "The Wonderful World of Linux 2.6," July 13, 2003, . 244. Linux kernel source code, version 2.5.75, . 245. "The Embedded Linux 'Cool Devices' Quick Reference Guide," modified March 21, 2002, . 246. "MontaVista Linux—Real-time Performance," May 2002, MontaVista Software, . 247. Lehrbaum, R., "Using Linux in Embedded Systems and Smart Devices," viewed July 21,2003, . 1030 Case Study: Linux 248. Hatch, B., and J. Lee, Hacking Linux Exposed, McGraw-Hill: Osborne, 2003, pp. 384-386. 249. Toxen, B., Real World Linux Security, 2nd ed., Prentice Hall PTR, 2002. 256. Jaeger, T.; D. Safford; and H. Franke, "Security Requirements for the Deployment of the Linux Kernel in Enterprise Systems," . 258. Wright, C, et al., "Linux Security Modules: General Security Support for the Linux Kernel," 2002, . 259. Bryson, D., "The Linux CryptoAPI: A User's Perspective," May 31,2002, . 260. Bryson, D., "Using CryptoAPI," May 31, 2002, . 261. Bovet, D., and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001. 262. Rubini, A., and J. Corbet, Linux Device Drivers, O'Reilly, 2001. But, soft! what light through yonder window breaks? It is the east, and Juliet is the sun! —William Shakespeare— An actor entering through the door, you've got nothing. But if he enters through the window, you've got a situation. -Billy Wilder- Chapter 21 Case Study: Windms XP Objectives After reading this chapter, you should understand: • the history of DOS and Windows operating systems. • the Windows XP architecture. • the various Windows XP subsystems. • asynchronous and deferred procedure calls. • how user processes, the executive and the kernel interact. • how Windows XP performs process, thread, memory and file management. • the Windows XP I/O subsystem. • how Windows XP performs interprocess communication. • networking and multiprocessing in Windows XP. • the Windows XP security model. Chapter Outline 21.1 Introduction 21.2 History Biographical Note: Bill Gates Biographical Note: David Cutler Mini Case Study: OS/2 21.3 Design Goals 21.6 Process and Thread Management 21.6.1 Process and Thread Organization 21.6.2 Thread Scheduling 21.6.3 Thread Synchronization 21.7 Memory Management 21.7.1 Memory Organization 21.7.2 Memory Allocation 21.7.3 Page Replacement 21.4 System Architecture 21.5 System Management Mechanisms 21.5.1 Registry 21.5.2 Object Manager 21.5.3 Interrupt Request levels (IRQs) 21.5.4 Asynchronous Procedure Calls (APCs) 21.5.5 Deferred Procedure Calls (DPCs) 21.5.6 System Threads 1034 21.8 File Systems Management 21.8.1 File System Drivers 21.8.2 NTFS 21.9 Input/Output Management 21.9.1 Device Drivers 21.9.2 Input/Output Processing 21.9.3 Interrupt Handling 21.9.4 File Cache Management 21.10 Interprocess Communication 21.10.1 Pipes 21.10.2 Mailslots 21.10.3 Shared Memory 21.10.4 Local and Remote Procedure Calls 21.10.5 Component Object Models (COM) 21.10.6 Drag-and-Drop and Compound Documents 21.11 Networking 21.11.1 Network Input/Output 21.11.2 Network Driver Architecture 21.11.3 Network Protocols 21.11.4 Network Services 21.11.5 .NET 21.12 Scalability 21.12.1 Symmetric Multiprocessing (SMP) 21.12.2 Windows XP Embedded 21.13 Securty 21.13.1 Authentication 21.13.2Authorization 21.13.3 Internet Connection Firewall 21.13.4 Other Features Web Resources | Key Terms | Exercises | Recommended Reading | Works Cited 1035 1036 Case Study: Windows XP 21.1 Introduction Windows XP, released by the Microsoft Corporation in 2001, combines Microsoft's corporate and consumer operating system lines. By early 2003, over one-third of all Internet users ran Windows XP, making it the most widely used operating system.1 Windows XP ships in five editions. Windows XP Home Edition is the desktop edition, and the other editions provide additional features. Windows XP Professional includes extra security and privacy features, more data recovery support and broader networking capabilities. Windows XP Tablet PC Edition is built for notebooks and laptops that need enhanced support for wireless networking and digital pens. Windows XP Media Center Edition provides enhanced multimedia support.2 Windows XP 64-Bit Edition is designed for applications that manipulate large amounts of data, such as programs that perform scientific computing or render 3D graphics. At the time of this writing, Microsoft plans to release a sixth edition, Windows XP 64-Bit Edition for 64-Bit Extended Systems, which is designed specifically to support AMD Opteron and Athlon 64 processors.3 Windows XP 64-Bit Edition currently executes on Intel Itanium II processors. Despite their differences, all Windows XP editions are built on the same core architecture, and the following case study applies to all editions (except when we explicitly differentiate between editions). This case study investigates how a popular operating system implements the components and strategies we have discussed throughout the book. 21.2 History In 1975, a Harvard University junior and a young programmer at Honeywell showed up at Micro Instrumentation and Telemetry Systems (MITS) headquarters with a BASIC compiler. The two men had written the compiler in just eight weeks and had never tested it on the computer for which it was intended. It worked on the first run. This auspicious beginning inspired the student, Bill Gates, and the programmer, Paul Allen, to drop everything they were doing and move to Albuquerque, New Mexico, to found Microsoft (see the Biographical Note, Bill Gates).4-5 By 2003, the two-man company had become a global corporation, employing over 50,000 people and grossing a yearly revenue of over $28 billion.6 Microsoft is the second most valuable and the sixth most profitable company in the world. Forbes Magazine in 2003 listed Bill Gates as the richest man on Earth, worth over $40.7 billion, and Paul Allen as the fourth richest man, worth $20.1 billion.7 Microsoft released the Microsoft Disk Operating System—MS-DOS 1.0—in 1981. MS-DOS 1.0 was a 16-bit operating system that supported 1MB of main memory (1MB was enormous by the standards of the time). The system ran one process at a time in response to user input from a command line. All programs executed in real mode, which provided them direct access to all of main memory including the portion storing the operating system.8 MS-DOS 2.0, released two years later, supported a 10MB hard drive and 360KB floppy disks.9 Subsequent 21.2 History 1037 Microsoft operating systems continued the trend of increasing disk space and supporting an ever-growing number of peripheral devices. Microsoft released Windows 1.0, its graphical-user-interface (GUI)-based operating system, in 1985. The GUI foundation had been developed by Xerox in the 1970s and popularized by Apple's Macintosh computers. The Windows GUI Biographical Note Bill Gates William H. Gates III was born in Seattle, Washington, in 1955.10 His first experience with computers was in high school. 11 ' 12 He befriended Paul Allen, a fellow computer enthusiast, and with two other students formed the Lakeside Programming Group. The group obtained computer time from nearby companies by arranging deals to find bugs in the companies' systems. They also earned money by forming a small and short-lived company named Traf-O-Data, writing software to process traffic flow data. 13, 14 Gates went on to Harvard University, but in 1974 Intel announced the 8080 chip, and Gates and Paul Allen realized that the future of computing was microprocessing. 15, 16, 17 When the MITS Altair 8800 was announced only months later—the world's first microcomputer—Gates and Allen quickly contacted MITS to say they had a implementation of BASIC (Beginner's All-purpose Symbolic Instruction Code) for the Altair. 18, 19 In fact they had no such thing, but Altair was interested; Gates took a leave of absence from Harvard (he never returned) and he and Allen ported BASIC to the Altair in just over one month, based only on Intel's manual for the 8080. 20, 21, 22 Early personal computers did not have operating systems, so all that was needed was a language interpreter built for the specific system to run programs. Gates' and Allen's implementation of BASIC was, in effect, the first operating system for the Altair. It was also the beginning of Microsoft. 23, 24 Over the next several years, Microsoft continued to develop their BASIC language and licensed it to other new microcomputer companies including Apple and Commodore. 25 Microsoft also implemented other languages for microcomputers and, because they had a jump-start on other microcomputer software compa- nies, began to take over the field. 26 In 1980, Gates hired his Harvard classmate, Steve Ballmer, to help with the business side of the growing company (Ballmer is the current CEO of Microsoft.) 27, 28 Only two years later, Microsoft developed MS-DOS based on Seattle Computing Product's 86DOS; it was offered as the cheapest operating system option for IBM's new line of personal computers and was a great success. In 1984 they broke into the office applications market with Word and Excel for the Apple Macintosh. The following year they released Windows 1.0, one of the first GUI operating systems for IBM-compatibles. 29 Microsoft's commercial success has continued to mount with progressive versions of Windows and their Microsoft Office software. Gates is currently the Chairman and Chief Software Architect of Microsoft. 30 He is also the richest person in the world. 31 1038 Case Study: Windows XP improved with each version. The window frames in Windows 1.0 could not overlap; Microsoft fixed this problem in Windows 2.0. The most important feature in Windows 2.0 was its support for protected mode for DOS programs. Although protected mode was slower than real mode, it prevented a program from overwriting the memory space of another program, including the operating system, which improved system stability. Even in protected mode, however, programs addressed memory directly. In 1990, Microsoft released Windows 3.0, quickly followed by Windows 3.1. This version of the operating system completely eliminated the unsafe real mode. Windows 3.1 introduced an enhanced mode, which allowed Windows applications to use more memory than DOS programs. Microsoft also released Windows for Workgroups 3.1, an upgrade to Windows 3.1 that included network support, especially for local area networks (LANs), which, at the time, were starting to become popular. Microsoft attempted to increase system stability in Windows by tightening parameter checking during API system calls; these changes broke many older programs that exploited "hidden" features in the API. 32, 33 Emboldened by the success of its home-user-oriented operating systems. Microsoft ventured into the business market. Microsoft hired Dave Cutler, an experienced operating systems designer who helped develop Digital Equipment Corporation's VMS operating system, to create an operating system specifically for the business environment (see the Biographical Note, David Cutler). Developing the new operating system took five years, but in 1993, Microsoft released Windows NT 3.1, a New Technology operating system. In so doing, Microsoft created its corporate line of operating systems, which was built from a separate code base than its consumer line. The two lines were not to merge until the release of Windows XP in 2001, although operating systems from one line used features and ideas from the other. For example, Windows NT 3.1 and Windows 95 included different implementations of the same API. In addition, the business and consumer GUI interfaces were always similar, although the business version tended to lag behind. 34 Because Windows NT 3.1 was developed for business users, its chief focus was security and stability. Windows NT introduced the New Technology File System (NTFS), which is more secure and more efficient than the then-popular Windows 3.1 FAT and IBM OS/2 HPFS file systems (see the Mini Case Study, OS/2). In addition, the 32-bit operating system protects its memory from direct access by user applications. The NT kernel runs in its own protected memory space. This extra layer of security came at a price; many existing graphics-intensive games could not run on Windows NT because they need to access memory directly to optimize performance. Windows NT 3.1 also lacks much of the multimedia support that made Windows 3.1 popular with home users.35 Microsoft released Windows NT 4.0 in 1996. The upgraded operating system provides additional security and networking support and includes the popular Windows 95 user-interface.36 Windows 2000 (whose kernel is NT 5.0), released in 2000. introduced the Active Directory. This database of users, computers and devices makes it easy for users to find information about resources spread across a network. 21.2 History 1039 In addition, system administrators can configure user accounts and computer settings remotely. Windows 2000 supports secure Kerberos single sign-on authentication and authorization (see Section 19.3.3, Kerberos). In general, Windows 2000 is more secure and has better networking support than previous Windows operating systems. It is Microsoft's last purely business-oriented desktop operating system.37 Microsoft continued developing consumer operating systems. It released Windows 95 in 1995 to replace Windows 3.1 for home users. Like Windows NT 3.1, Windows 95 makes the jump to 32-bit addressing, although it supports 16-bit applications for backward compatibility. Windows 95 permits applications to access Biographical Note David Cutler David Cutler graduated from Olivet College in Michigan in 196538 with a degree in Mathematics. 39 He first became interested in computers when he worked on a software simulation project at DuPont. 40, 41 Cutler moved to its engineering department, where he was involved in various programming projects, became interested in operating systems and worked on the development of several small, real-time operating systems.42 In 1971 Cutler moved to Digital Equipment Corporation (DEC),43, 44 where he led the team developing the RSX-11M operating system. This operating system was difficult to design for two reasons—there were severe memory limitations due to the PDP-11 's 16-bit addressing, and it had to be compatible with all of the popular PDP-11 models. A few years later. DEC began work on the VAX 32bit architecture, because increasing application sizes made the memory space accessible by a 16bit address insufficient. 45 Cutler managed what was probably DEC'S most important OS project ever—VMS for the VAX systems.46, 47 Because the PDP-11 was so widely used, VMS needed to be backward compatible with the PDP-11 system and it had to be scalable to VAX systems of varying power. 48, 49 Cutler also led the MicroVAX project, DEC's microprocessor version of the VAX. 50 In 1988, Cutler accepted an offer from Bill Gates to be one of the leaders on Microsoft's successor to OS/2—OS/2 New Technology, or OS/2 NT, which would use the OS/2 API.51, 52 cutler insisted on bringing 20 of his developers from DEC to form the core of this new team. Meanwhile, Microsoft released Windows 3.0, and its popularity led to a change in plans for Cutler's project. His operating system would now be Windows NT and would primarily support the Win32 interface. However, it needed to support parts of the DOS, POSIX and OS/2 APIs in addition to running Windows 3.0 programs. 53 Inspired by the Mach microkernel operating system. Cutler's team designed Windows NT to have a fairly small kernel with layers of separate modules to handle the various interfaces. Windows NT, originally intended for Microsoft's professional line of operating systems, eventually was used for the consumer line in version 5.1, more popularly known as Windows XP. Cutler remains a lead operating systems architect at Microsoft. 1040 Case Study: Windows XP main memory directly. However, Microsoft introduced DirectX, which simulates direct hardware access while providing a layer of protection. Microsoft hoped that DirectX, combined with faster hardware, would give game developers a viable alternative to accessing the hardware directly. Windows 95 introduced multithreading, an improvement on the multitasking support provided by Windows 3.1. Microsoft released Windows 98 in 1998, although most of the changes were minor (primarily increased multimedia and networking support). The biggest innovation was combining Internet Explorer (IE) with the operating system. Bundling IE made Windows easier to use when accessing the Internet and the Web. Windows Millennium Edition (Windows Me), released in 2000, is the last version in the consumer line. Windows Me has even more device drivers and multimedia and networking support than previous versions of Windows. It is also the first consumer version of Windows that cannot boot in DOS mode. 54 Mini Case Study OS/2 OS/2 was first released in 1987. The operating system was a joint project between Microsoft and IBM designed to run on IBM's new PS/2 line of personal computers.55 OS/2 was based on DOS, the most popular operating system at the time of OS/2's release, and was similar to DOS in many ways (e.g., they had similar commands and command prompts). OS/2 also made significant improvements over DOS. OS/2 was the first PC operating system to allow multitasking (however, only one application could be on the screen at a time). OS/2 also supported multithreading, interprocess communication (IPC), dynamic linking and many other features that were not supported by DOS.56 OS/2 1.1, released in 1988, was its first version with a graphical user interface. The interface was similar to early versions of Windows. However, unlike Windows and many of today's interfaces, the coordinate (0,0) referred to the lower-left corner of the screen as opposed to the upper-left comer. 57 This made transferring programs from OS/2 to other operating systems difficult. OS/2 version 2.0, released in 1992, was the first 32-bit operating system for personal computers. OS/ 2 2 contained the Integrating Platform, designed to run OS/2 2, OS/2 1, DOS and Windows applications. This benefitted OS/2, because there were few applications writ- ten for OS/2 2 at the time. As a result, many developers wrote programs specifically for Windows, since they would work on both platforms. 58 IBM and Microsoft disagreed on the progress and direction of OS/2 because Microsoft wanted to devote more resources to Windows than to OS/2, whereas IBM wished to further develop OS/2 1. In 1990, IBM and Microsoft agreed to work independently On upcoming OS/2 projects: IBM took control of all future versions of OS/2 1 and OS/2 2, while Microsoft developed OS/2 3. Soon after this agreement, Microsoft renamed OS/2 3 to Windows NT.59 21.3 Design Goals 1041 Microsoft released Windows XP in 2001 (whose kernel is NT 5.1). Windows XP unified the consumer and commercial code bases. Windows XP comes in several editions, each of which is built from the same code base but is tailored to fit the needs of a specific group of users. Every edition of Windows XP combines the stability, security and network support of the corporate line with the multimedia and device support of the consumer line. 21.3 Design Goals Microsoft's main design goal for Windows XP was reintegrating the code bases from its consumer and business operating system lines. In addition, Microsoft attempted to better meet the traditional goals of each individual line. For example, the corporate line has often focused on providing security and stability, whereas the consumer line provides enhanced multimedia support for games. When Microsoft created the NT line for business users, its chief focuses were stability, security and scalability.60 Businesses often employ computers to execute critical applications and store essential information. It is important for the system to be reliable and run continuously without error. The internal architecture described in this case study was developed over many years to enhance the stability of Microsoft operating systems. Corporate computers need to protect sensitive data while being connected to multiple networks and the Internet. For this reason, Microsoft devoted a great deal of effort to developing an all-encompassing security system. Windows XP uses modern security technologies such as Kerberos, access control lists and packet filtering firewalls to protect the system. The Windows XP security system is built on the codebase of Windows 2000, to which the government gave C2 level certification (the C2 certification applies only to Windows 2000 machines unconnected to a network). 61 At the time of this writing, Microsoft is in the process of gaining the same certification for Windows XP Professional. Computers must be scalable to meet the needs of different users. Microsoft developed Windows XP 64-Bit Edition to support high-performance desktop computers and Windows XP Embedded to support devices such as routers. Windows XP Professional supports symmetric multiprocessing (see Chapter 15, Multiprocessor Management). Windows XP incorporates many of the features that made the consumer line popular. The system contains an improved GUI and more multimedia support than previous versions of Windows. For the first time, both business and home users can take advantage of these innovations. Besides incorporating the strengths of its two lines of operating systems, Microsoft added another goal specifically for Windows XP. Customers had often complained that Windows started too slowly. As a result, Microsoft aimed at dramatically decreasing boot time. Windows XP must be usable in 30 seconds after a user turns on the power, 20 seconds after waking from hibernation (on a laptop) and 5 seconds 1042 Case Study: Windows XP after resuming from standby. Later in the chapter, we describe how Windows XP meets these goals by prefetching files necessary to load the operating system.62 21.4 System Architecture Windows XP is modeled on a microkernel architecture —often called a modified microkernel architecture. On the one hand, Windows XP is a modular operating system with a compact kernel that provides base services for other operating system components. However, unlike a pure microkernel operating system, these components (e.g., the file system and memory manager) execute in kernel mode rather than user mode. Windows XP is also a layered operating system, composed of discrete layers with lower layers providing functionality for higher layers.63 However, unlike a pure layered operating system, there are times when non-adjacent layers communicate. For a description of microkernel and layered operating system designs, see Section 1.13, Operating System Architectures. Figure 21.1 illustrates Windows XP's system architecture. The Hardware Abstraction Layer (HAL) interacts directly with the hardware, abstracting hardware specifics to the rest of the system. The HAL handles device components on the mainboard such as the cache and system timers. The HAL abstracts hardware specifics that might differ between systems of the same architecture, such as the exact layout of the mainboard. In many cases, kernel-mode components do not access the hardware directly, but instead call functions exposed by the HAL. Therefore, kernel-mode components can be constructed with little regard for architecture-specific details such as cache sizes and the number of processors included. The HAL interacts with device drivers to support access to peripheral devices.64 The HAL also interacts with the microkernel. The microkernel provides basic system mechanisms, such as thread scheduling, to support other kernel-mode components. [Note: The Windows XP microkernel should not be confused with the microkernel design philosophy discussed in Section 1.13, Operating System Architectures.] The microkernel also handles thread synchronization, dispatches interrupts and handles exceptions.65 Additionally, the microkernel abstracts architecture-specific features—features that differ between two different architectures, such as the number of interrupts. The microkernel and the HAL combine to make Windows XP portable, allowing it to run in many different hardware environments.66 The microkernel forms only a small part of kernel space. Kernel space describes an execution environment in which components can access all system memory and services. Components that reside in kernel space execute in kernel mode. The microkernel provides the base services for the other components residing in kernel space.67 Layered on top of the microkernel are the kernel-mode components responsible for administering the operating system subsystems (e.g., the I/O manager and virtual memory manager). Collectively, these components are known as the executive. Figure 21.1 displays the key executive components. The rest of this chapter is devoted to describing these subsystems and the executive components which man- 21.4 System Architecture 1043 Figure 21.1 | Windows XP system architecture. age the subsystems. The executive exposes services through an application programming interface (API) to user processes.68 This API is called the native API. However, most user processes do not call native API functions directly; rather, they call API functions exposed by user-mode system components called environment subsystems. Environment subsystems are user-mode processes interposed between the executive and the rest of user space that export an API for a specific computing environment. For example, the Win32 environment subsystem provides a typical 32-bit Windows environment. Win32 processes call functions defined in the Windows API; the Win32 subsystem translates these function calls to system calls in the native API (although, in some cases, the Win32 subsystem process itself can fulfill an API request without calling the native API). 69 Although Windows XP allows user-mode processes to call native API functions, Microsoft urges developers to use an environment subsystem's interface.70 By default, 32-bit Windows XP includes only the Win32 subsystem, but users can install a POSIX subsystem. Microsoft replaced the POSIX subsystem in older versions of Windows with a more robust UNIX environment, called Service For UNIX (SFU) 3.0. Running SFU 3.0, users can port code written for a UNIX environment to Windows XP. To run 16-bit DOS applications, Windows XP provides a Virtual DOS Machine (VDM), which is a Win32 process that provides a DOS environment for DOS applications.71 1044 Case Study: Windows XP On Windows XP 64-Bit Edition, the default subsystem supports 64-bit applications. However, Windows XP 64-Bit Edition provides a subsystem called the Windows on Windows 64 (WOW64) subsystem, which allows users to execute 32-bit Windows applications. Windows XP 64-Bit Edition does not support 16-bit applications.72 The top layer of the Windows XP architecture consists of user-mode processes. These are typical applications (e.g., word processors, computer games and Web browsers) as well as dynamic link libraries (DLLs). DLLs are modules that provide data and functions to processes. As shown in Fig. 21.1, the environment subsystems' APIs are simply DLLs that processes dynamically link when calling a function in a subsystem's API. Because DLLs are dynamically linked, applications can be more modular. If the implementation of a DLL function changes, only the DLL must be recompiled and not the applications that use it.73 Windows XP also includes special user-mode processes called system services (or Win32 services), which are similar to Linux daemons (see Section 20.3, Linux Overview). These processes usually execute in the background whether or not a user is logged onto the computer and typically execute the server side of client/server applications.74 Examples of system services are the Task Scheduler (which allows users to schedule tasks to be executed at specific times), IPSec (which manages Internet security usually during data transfer operations) and Computer Browser (which maintains a list of computers connected on the local network). 75 [Note: Sometimes, authors also use the term "system services" to refer to native API functions.] 21.5 System Management Mechanisms Before we investigate each component in the Windows XP operating system, it is important to understand the environment in which these components execute. This section describes how components and user processes store and access configuration data in the system. We also consider how Windows XP implements objects, how these objects are managed and how user- and kernel-mode threads can manipulate these objects. Finally, we describe how Windows XP prioritizes interrupts and how the system employs software interrupts to dispatch processing. 21.5.1 Registry The registry is a database, accessible to all processes and kernel-mode components that stores configuration information76 specific to users, applications and hardware. For example, user information might include desktop settings and printer settings. Application configuration includes such settings as recently used menus, user preferences and application defaults. Hardware information includes which devices are currently connected to the computer, which drivers serve each device and which resources (e.g., hardware ports) are assigned to each device. The registry also stores system information such as file associations (e.g., .doc files are associated with 21.5 System Management Mechanisms 1045 Microsoft Word). Applications, device drivers and the Windows XP system use the registry to store and access data in a centralized manner. For example, when a user installs a hardware device, device drivers place configuration information into the registry. When Windows XP allocates resources to various devices, it accesses the registry to determine what devices are installed in the system and distributes resources accordingly.77 Windows XP organizes the registry as a tree, and each node in the tree represents a registry key. A key stores subkeys and values; values consist of a value name and value data. 78 The system provides several predefined keys. For example, HKEY_LOCAL_MACHINE is a predefined key which forms the root of all configuration data for the local computer. Values in the HKEY_LOCAL_MACHINE key or its subkeys store such information as the hardware devices connected to the computer, system memory and network information (e.g., server names). Predefined keys act as roots for a single type of data, such as application-specific data or user preferences. 79 Threads can access subkeys by navigating the tree structure. Beginning with any key that is open, a thread can enumerate that key's subkeys and open any of them. The system's predefined keys remain open at all times. Once a key is open, values and subkeys can be read, modified, added or deleted (assuming the thread has access rights for the key to accomplish all these actions). 80, 81 Windows XP stores the registry in a collection of hives, which are portions of the registry tree. Most hives are stored as files and flushed to disk periodically to protect the data against crashes, whereas other hives remain solely in memory (e.g., HKEY_LOCAL_MACHINE\HARDWARE, which stores hardware information that the system collects at boot time and updates as it runs). 82 The configuration manager is the executive component responsible for managing the registry. Device drivers, system components and user applications communicate with the configuration manager to modify or obtain data from the registry. The configuration manager is also responsible for managing registry storage in system memory and in the hives.83 21.5.2 Object Manager Windows XP represents physical resources (e.g., peripheral devices) and logical resources (e.g., processes and threads) with objects. Note that these objects do not refer to objects in any object-oriented programming language, but rather constructs that Windows XP uses to represent resources. Windows XP represents objects with a data structure in memory; this data structure stores the object's attributes and procedures. 84 Each object in Windows XP is defined by an object type; the object type is also stored as a data structure. Object types are created by executive components; for example, the I/O manager creates the file object type. The object type specifies the types of attributes an instance of this type possesses and the object's standard procedures. Examples of standard procedures include the open procedure, close procedure and delete procedure. 85 Object types in Windows XP include files, devices, processes, threads, pipes, semaphores and many others. Each Windows XP object is an instance of an object type.86 1046 Case Study: Windows XP User-mode processes and kernel components can access objects through object handles. An object handle is a data structure that allows processes to manipulate the object. Using an object handle, a thread can call one of the object's procedures or manipulate the object's attributes. A kernel-mode component or usermode process with sufficient access rights can obtain a handle to an object by specifying the object's name in a call to the object's open method or by receiving a handle from another process.87 Once a process obtains a handle to an object, this process can duplicate the handle; this can be used by a process that wishes to pass a handle to another process and also retain a handle to the object.88 Processes can use a handle in similar way as pointers, but a handle gives the system extra control over the actions of a user process (e.g., by restricting what the process can do with the handle, such as not allowing the process to duplicate it). Kernel-mode components can use handles when interacting with user-mode processes, but otherwise can access objects directly through pointers. In addition, kernel-mode components can create kernel handles, which are handles accessible from any process's address space, but only from kernel mode. In this way, a device driver can create a kernel handle to an object and be guaranteed access to that object, regardless of the context in which the driver executes.89 Objects can be named or unnamed, although only kernel-mode components can open a handle to an unnamed object. A user-mode thread can obtain a handle to an unnamed object only by receiving it from a kernel-mode component. 90 The object manager is the executive component that manages objects for a Windows XP system. The object manager is responsible for creating and deleting objects and maintaining information about each object type, such as how many objects of a certain type exist.91 The object manager organizes all named objects into a hierarchal directory structure. For example, all device objects are stored in the \Device directory or one of its subdirectories. This directory structure composes the object manager namespace (i.e., group of object names in which each name is unique in the group). 92 To create an object, a process

passes a request to the object manager; the object manager initializes the object and returns a handle (or pointer if a kernelmode component called the appropriate function) to it. If the object is named, the object manager enters it into the object manager namespace. 93 For each object, the object manager maintains a count of how many existing handles refer to it and a count of how many existing references (both handles and pointers) refer to it. When the number of handles reaches zero, the object manager removes the object from the object manager namespace. When the number of references to the object reaches zero, the object manager deletes the object.94 21.5.3 Interrupt Reqiest levels (IRQLs) Interrupt definition and handling are crucial to Windows XP. Because some interrupts, such as hardware failure interrupts, are more important than others, Windows XP defines interrupt request levels (IRQLs). IRQLs are a measure of 21.5 System Management Mechanisms 1047 interrupt priority; a processor always executes at some IRQL, and different processors in a multiprocessor system can be executing at different IRQLs. An interrupt that executes at an IRQL higher than the current one can interrupt the current execution and obtain the processor. The system masks (i.e., delays processing) interrupts that execute at an IRQL equal to or lower than the current one.95 Windows XP defines several IRQLs (Fig. 21.2). User- and kernel-mode threads normally execute at the passive IRQL, which is the lowest-priority IRQL. Unless user-mode threads are executing asynchronous procedure calls (APCs), which we describe in Section 21.5.4, Asynchronous Procedure Calls (APCs), these threads always execute at the passive IRQL. In some instances, kernel-mode threads execute at a higher IRQL to mask certain interrupts, but typically, the system executes threads at the passive level so that incoming interrupts can be serviced promptly. Executing at the passive IRQL implies that no interrupt is currently being processed and no interrupts are masked. APCs, which are procedure calls that threads or the system can queue for execution by a specific thread, execute at the APC IRQL. Above the APC level is the DPC/dispatch IRQL at which deferred procedure calls (i.e., software interrupts that can be executed in any thread's context) and various other important kernel functions execute, including thread scheduling.96 We describe DPCs in Section 21.5.5, Deferred Procedure Calls (DPCs). All software interrupts execute at the APC level or DPC/dispatch level.97 Hardware interrupts execute at the remaining higher IRQLs. Windows XP provides multiple device IRQLs (DIRQLs) at which interrupts from devices execute.98 The number of DIRQLs varies, depending on how many interrupts a given architecture supports, but the assignment of these levels to devices is arbitrary; it is dependent on the specific interrupt request line at which a device interrupts. Therefore, Windows XP maps device interrupts to DIRQLs without priority; still, if a processor is executing at a certain DIRQL, interrupts that occur at DIRQLs less than or equal to the current one are masked. 99 The system reserves the highest five IRQLs for critical system interrupts. The first level above DIRQL is the profile IRQL, which is used when kernel profiling is enabled (see Section 14.5.1, Tracing and Profiling). The system clock, which issues interrupts at periodic intervals, interrupts at the clock IRQL. Interprocessor requests (i.e., interrupts sent by one processor to another such as when shutting down the system) execute at the request IRQL. The power IRQL is reserved for power failure notification interrupts, although this level has never been used in any NT-based operating system. Interrupts issued at the high IRQL (which is the highest IRQL) include hardware failure interrupts and bus errors. 100 For each processor, the HAL masks all interrupts with IRQLs less than or equal to the IRQL at which that processor is currently executing. This allows the system to prioritize interrupts and execute critical interrupts as quickly as possible. For example, executing at the DPC/dispatch level masks all other DPC/dispatch level interrupts and APC level interrupts. Interrupts that execute at an IRQL that is currently masked are not discarded (i.e., removed from the system). When the 1048 Case Study: Windows XP Figure 21.2 | Interrupt request levels (IRQLs) IRQL drops low enough, the processor processes the waiting interrupts. Kernelmode threads can raise the IRQL, and a thread that does so is responsible for subsequently lowering the IRQL to its previous level; user-mode threads cannot change the IRQL. Windows XP attempts to do as much processing as possible at the passive level, at which no interrupts are masked, so that all incoming interrupts can be serviced immediately.101 21.5.4 Asynchronous Procedure Calls (APCs) Asynchronous procedure calls (APCs) are procedure calls that threads or the system can queue for execution by a specific thread; the system uses APCs to complete vari- 21.5 System Management Mechanisms 1049 ous functions that must be done in a specific thread's context. For example, APCs facilitate asynchronous I/O. After initiating an I/O request, a thread can perform other work. When the I/O request completes, the system sends an APC to that thread, which the thread executes to complete the I/O processing. Windows XP uses APCs to accomplish a variety of system operations. For example, APCs are used by the SFU to send signals to POSIX threads and by the executive to notify a thread of an I/O completion or direct a thread to perform an operation. 102 Each thread maintains two APC queues: one for kernel-mode APCs and one for user-mode APCs.103 Kernel-mode APCs are software interrupts that are generated by kernel-mode components, and a thread must process all pending kernelmode APCs as soon as it obtains a processor. A user-mode thread can create a usermode APC and request to queue this APC to another thread; the system queues the APC to the target thread if the queuing thread has sufficient access rights. A thread executes user-mode APCs when the thread enters an alertable wait state.104 In an alertable wait state, a thread awakens either when it has a pending APC or when the object or objects on which the thread is waiting become available. A thread can specify that it enters an alertable wait state when it waits on an object, or a thread can simply enter an alertable wait state without waiting on any object.105 Windows XP also differentiates between normal kernel APCs and special kernel APCs. A thread executes kernel APCs in FIFO order, except that the thread executes all special kernel APCs before all normal kernel APCs. 106 When the system queues a kernel APC, the target thread does not execute the APC until the thread gains the processor. However, once the thread begins executing at the passive level, it begins executing kernel APCs until it drains (i.e., empties) its queue. Unlike kernel APCs, threads can choose when to execute user-mode APCs. A thread drains its user-mode APC queue when the thread enters an alertable wait state. If the thread never enters such a state, its pending user-mode APCs are discarded when it terminates. 107 A thread executing an APC is not restricted, except that it cannot receive new APCs. It can be preempted by another higher-priority thread. Because scheduling is not masked at the APC level (this occurs at the DPC/dispatch level), a thread executing an APC can block or access pageable data (which might cause a page fault and hence cause Windows XP to execute scheduling code at the DPC/dispatch IRQL). APCs can also cause the currently executing thread to be preempted. If a thread queues an APC to a thread of higher priority that is in an alertable wait state, the waiting thread awakens, preempts the queuing thread and executes the APC. 108 21.5.5 Defered Procedure Calls (DPCs) Most of Windows XP's hardware interrupt processing occurs in deferred procedure calls (DPCs). DPCs are software interrupts that execute at the DPC/dispatch IRQL109 and which run in the context of the currently executing thread. In other words, the routine that creates the DPC does not know the context in which the DPC will execute. Therefore, DPCs are meant for system processing, such as interrupt pro- 1050 Case Study: Windows XP cessing, that does not need a specific thread context in which to execute.110 These procedure calls are called "deferred" because Windows XP processes most hardware interrupts by calling the associated interrupt service routine, which is responsible for quickly acknowledging the interrupt and queuing a DPC for later execution. This strategy increases the system's responsiveness to incoming device interrupts. Windows XP attempts to minimize the time it spends at DIRQLs to maintain high responsiveness to all device interrupts. Recall that the system maps device interrupts arbitrarily to DIRQLs. Still, device interrupts mapped to a high DIRQL can interrupt the processing of a device interrupt mapped to a lower DIRQL. By using DPCs, the system returns to an IRQL below all DIRQLs and therefore can minimize this difference in the level of interrupt servicing.111 Device interrupt handlers can schedule DPCs, and Windows XP executes some kernel functions, such as thread scheduling, at the DPC/dispatch IRQL. 112 When an event that prompts scheduling occurs—such as when a thread enters the ready state, when a thread enters the wait state, when a thread's quantum expires or when a thread's priority changes—the system executes scheduling code at the DPC/ dispatch IRQL. 113 As we noted above, DPCs are restricted because the routine that creates a DPC does not specify the context in which the DPC executes (this context is simply the currently executing thread). DPCs are further restricted because they mask thread scheduling. Therefore, a DPC cannot cause the thread within whose context it is executing to block, because this would cause the system to attempt to execute scheduling code. In this case no thread would execute because the executing thread blocks, but the scheduler does not schedule a new thread because scheduling is masked (in reality, Windows XP halts the system and displays an error message).114 As a result, DPCs cannot access pageable data because if a page fault occurs, the executing thread blocks to wait for the completion of the data access. DPCs, however, can be preempted by hardware interrupts. 115, 116 When a device driver generates a DPC, the system places the DPC in a queue associated with a specific processor. The driver routine can request a processor to which the DPC should be queued (e.g., to ensure that the DPC routine is executed on the processor with relevant cached data); if no processor is specified, the processor queues the DPC to itself.117 The routine can also specify the D P C s priority. Low- and medium-priority DPCs are placed at the end of the processor's DPC queue; high-priority DPCs enter at the front of the queue. In most cases, when the IRQL drops to the DPC/dispatch level, the processor drains the DPC queue. However, when a low-priority DPC has been waiting for only a short time and the DPC queue is short, the processor does not immediately execute the DPC. The next time the processor drains the DPC queue, it processes this DPC. This policy allows the system to return to the passive level faster to unmask all interrupts and resume normal thread processing.118 21.6 Process and Thread Management 1051 21.5.6 System Threads When a kernel-mode component, such as a device driver or executive component, performs some processing on behalf of a user thread, the component's functions often execute in the context of the requesting thread. However, in some cases, kernel-mode components must accomplish a task that is not in response to a particular thread request. For example, the cache manager must periodically flush dirty cache pages to disk. Also, a device driver servicing an interrupt might not be able to accomplish all necessary processing at an elevated IRQL (i.e., the device's DIRQL when it interrupts or the DPC/dispatch level when executing a DPC). For example, the driver might need to accomplish some work at the passive IRQL, because it must access pageable data.119 Developers of kernel-mode components have two options to perform this processing. A component can create a kernel thread, which is a thread that executes in kernel mode and typically belongs to the System process—a kernel-mode process to which most kernel-mode threads belong. Aside from executing in kernel mode, kernel threads behave and are scheduled the same as user threads.120 Kernel threads normally execute at the passive IRQL, but can raise and lower the IRQL.121 A second option is to use a system worker thread—a thread, which is created at system initialization time or dynamically in response to a high volume of requests, that sleeps until a work item is received. A work item consists of a function to execute and a parameter describing the context in which to execute that function. System worker threads also belong to the System process. Windows XP includes three types of worker threads: delayed, critical and hypercritical. The distinction arises from the scheduling priority given to each type. Delayed threads have the lowest priority of the three types; critical threads have a relatively high priority; and the hypercritical thread (there is only one), which is reserved for system use (e.g., device drivers cannot queue work items to this thread), has the highest priority of the three types.122, 123 21.6 Process and Thread Management In Windows XP, a process consists of program code, an execution context, resources (e.g., object handles) and one or more associated threads. The execution context includes such items as the process's virtual address space and various attributes (e.g., security attributes and working set size limitations). Threads are the actual unit of execution; threads execute a piece of a process's code in the process's context, using the process's resources. A thread also contains its own execution context which includes its runtime stack, the state of the machine registers and several attributes (e.g., scheduling priority). 124 Both processes and threads are objects, so other processes can obtain handles for processes and threads, and other threads can wait on process and thread events just as with other object types.125, 126 1052 Case Study: Windows XP 21.6.1 Process and Thread Organization Windows XP stores process and thread context information in several data structures. An executive process block (EPROCESS block) is the main data structure that describes a process. The EPROCESS block stores information that executive components use when manipulating a process object; this information includes the process's ID, a pointer to the process's handle table, a pointer to the process's access token and working set information (e.g., the process's minimum and maximum working set size, page fault history and the current working set). The system stores EPROCESS blocks in a linked list.127.128 Each EPROCESS block also contains a kernel process block (KPROCESS block). A KPROCESS block stores process information used by the microkernel. Recall that the microkernel manages thread scheduling and thread synchronization. Therefore, the KPROCESS block stores such information as the process's base priority class, the default quantum for each of its threads and its spin lock.129, 130 The EPROCESS and KPROCESS blocks exist in kernel space and store information for kernel-mode components to access while manipulating a process. The system also stores process information in a process environment block (PEB), which is stored in the process's address space. A process's EPROCESS block points to the process's PEB. The PEB stores information useful for user processes, such as a list of DLLs linked to the process and information about the process's heap.131-132 Windows XP stores thread data in a similar manner. An executive thread block (ETHREAD block) is the main data structure describing a thread. It stores information that executive components use when manipulating a thread object. This information includes the ID of the thread's process, its start address, its access token and a list of its pending I/O requests. The ETHREAD block for a thread also points to the EPROCESS block of its process. 133 ' 134 Each ETHREAD block stores a kernel thread block (KTHREAD block). The microkernel uses information in the KTHREAD block for thread scheduling and synchronization. For example, the KTHREAD block stores information such as the thread's base and current priority (later in this section we see how a thread's priority can change), its current state (e.g., ready, waiting) and any synchronization objects for which it is waiting.135, 136 The ETHREAD and KTHREAD blocks exist in kernel space and therefore are not accessible to users. A thread environment block (TEB) stores information about a thread in the thread's process's address space. Each KTHREAD block points to a TEB. A thread's TEB stores information such as the critical sections owned by the thread, its ID and information about its stack. A TEB also points to the PEB of the thread's process. 137, 138 All threads belonging to the same process share a virtual address space. Although most of this memory is global, threads can maintain their own data in thread local storage (TLS). A thread within a process can allocate a TLS index for the process; the thread stores a pointer to local data in a specified location (called a TLS slot) in this index. Each thread using the index receives one TLS slot in the 21.6 Process and Thred Management 1053 index in which it can store a data item. A common use for a TLS index is to store data associated with a DLL that a process links. Processes often contain many TLS indexes to accommodate multiple purposes, such as for DLL and environment subsystem data. When threads no longer need a TLS index (e.g., the DLL completes execution), the process can discard the index.139 A thread also possesses its own runtime stack on which it can also store local data. Creating and Terminating Processes A process can create another process using API functions. For Windows processes, the parent (i.e., creating) process and child (i.e., created) process are completely independent. For example, the child process receives a completely new address space.140 This differs from the fork command in Linux, in which the child receives a copy of its parent's address space. The parent process can specify certain attributes that the child process inherits (i.e., the child acquires a duplicate from the parent), such as most types of handles, environment variables—i.e., variables that define an aspect of the current settings, such as the operating system version number—and the current directory.141 When the system initializes a process, the process creates a primary thread. The primary thread acts as any other thread, and any thread can create other threads belonging to that thread's process.142 A process can terminate execution for a variety of reasons. If all of its threads terminate, a process terminates, and any of a process's threads can explicitly terminate the process at any time. Additionally, when a user logs off, all the processes running in the user's context are terminated. Because parent and child processes are independent of one another, terminating a parent process has no affect on its children and vice versa.143 Jobs Sometimes, it is preferable to group several processes together into a unit, called a job. A job object allows developers to define rules and set limits on a number of processes. For example, a single application might consist of a group of processes. A developer might want to restrict the number of processor cycles and amount of memory that the application consumes (e.g., so that an application executing on behalf of a client does not monopolize all of a server's resources). Also, the developer might want to terminate all the processes of a group at once, which is difficult because all processes are normally independent of one another. 144 A process can be a member of at most one job object. The job object specifies such attributes as a base priority class, a security context, a working set minimum and maximum size, a virtual memory size limit and both a per-process and jobwide processor time limit. Just as a thread inherits these attributes from the process to which it belongs, a process inherits these attributes from its associated job (if the process belongs to a job). 145, 146 The system can terminate all processes in a job by terminating the job. 147 Systems executing batch processes, such as data mining, benefit from jobs. Developers can limit the amount of processor time and memory a computationally intensive job consumes to free more resources for interactive processes. Sometimes 1054 Case Study: Windows XP it is useful to create a job for a single process, because a job allows the developer to specify tighter restrictions than a process allows. For example, with a job object, a developer can limit the process's total processor time; a developer cannot use the process object to do this.148 Fibers Threads can create fibers, which are similar to threads, except that a fiber is scheduled for execution by the thread that creates it, rather than the microkernel. Windows XP includes fibers to permit the porting of code written for other operating systems to Windows XP. Some operating systems, such as many UNIX varieties, schedule processes and require them to schedule their own threads. Developers can use fibers to port code to and from these operating systems. A fiber executes in the context of the thread that creates the fiber.149 Creating fibers generates additional units of execution within a single thread. Each fiber must maintain state information, such as the next instruction to execute and the values in a processor's registers. The thread stores this state information for each fiber. The thread itself is also a unit of execution, and thus must convert itself into a fiber to separate its own state information from other fibers executing in its context. In fact, the Windows API forces a thread to convert itself into a fiber before creating or scheduling other fibers. Although the thread becomes a fiber, the thread's context remains, and all fibers associated with that thread execute in that context.150 Whenever the kernel schedules a thread that has been converted to a fiber for execution, the converted fiber or another fiber belonging to that thread runs. Once a fiber obtains the processor, it executes until the thread in whose context the fiber executes is preempted, or the fiber switches execution to another fiber (either within the same thread or a fiber created by a separate thread). Just as threads possess their own thread local storage (TLS), fibers possess their own fiber local storage (FLS), which functions for fibers exactly as TLS functions for a thread. A fiber can also access its thread's TLS. If a fiber deletes itself (i.e., terminates), its thread terminates. 151 Fibers permit Windows XP applications to write code executed using the many-to-many mapping; typical Windows XP threads are implemented with a oneto-one mapping (see Section 4.6.2, Kernel-Level Threads). Fibers are user-level units of execution and invisible to the kernel. This makes context switching between fibers of the same thread fast because it is done in user mode. Therefore, a singlethreaded process with many fibers can simulate a multithreaded process. However, the Windows XP microkernel only recognizes one thread, and because Windows XP schedules threads, not processes, the single-threaded process receives less execution time (all else being equal). 152, 153 Thread Pools Windows XP also provides each process with a thread pool that consists of a number of worker threads, which execute functions queued by user threads. The worker 21.6 Process and Thread Management threads sleep until a request is queued to the pool. 154 The thread that queues the request must specify the function to execute and must provide context information.155 When a process is created, it receives an empty thread pool; the system allocates space for worker threads when the process queues its first request. 156 Thread pools have many purposes. Developers can use them to handle client requests. Instead of incurring the costly overhead of creating and destroying a thread for each request, the developer simply queues the request to the thread pool. Also, several threads that spend most of their time sleeping (e.g., waiting for events to occur) can be replaced by a single worker thread that awakens each time one of these events occurs. Furthermore, applications can use the thread pool to accomplish asynchronous I/O by queuing a worker thread to execute the I/O completion routines. Using thread pools can make an application more efficient and simpler because developers do not have to create and delete as many threads. However, thread pools transfer some control from the programmer to the system, which can introduce inefficiency. For example, the system grows and shrinks the size of a process's thread pool in response to request volume; in some cases, the programmer can better guess how many threads are needed. Thread pools also require memory overhead, because the system grows a process's (initially empty) thread pool as the process's threads queue work items.157 21.6.2 Thread Scheduling Windows XP does not contain a specific "thread scheduler" module —the scheduling code is dispersed throughout the microkernel. The scheduling code collectively is referred to as the dispatcher.158 Windows XP supports preemptive scheduling among multiple threads. The dispatcher schedules each thread without regard to the process to which the thread belongs. This means that, all else being equal, the same process implemented with more threads will get more execution time. 159 The scheduling algorithm used by Windows XP is based on a thread's priority. Before describing the scheduling algorithm in detail, we investigate the lifecycle of a Windows XP thread. Thread States In Windows XP, threads can be in any one of eight states (Fig. 21.3). A thread begins in the initialized state during thread creation. Once initialization concludes, the thread enters the ready state. Threads in the ready state are waiting to use a processor. A thread that the dispatcher has decided will execute on a particular processor next enters the standby state as it awaits its turn for that processor. A thread is in the standby state, for example, during the context switch from the previously executing thread to that thread. Once the thread obtains a processor, it enters the running state. A thread transitions out of the running state if it terminates execution, exhausts its quantum, is preempted, is suspended or waits on an object. If a thread terminates, it enters the terminated state. The system does not necessarily delete a terminated thread; the object manager deletes a thread only when the thread object's 1055 1056 Case Study: Windows XP Figure 21.3 | Thread state-transition diagram. reference count becomes zero. If a running thread is preempted or exhausts its quantum, it returns to the ready state. If a running thread begins to wait on an object handle, it enters the waiting state. Also, another thread (with sufficient access rights) or the system can suspend a thread, forcing it into the waiting state until the thread is resumed. When the thread completes its wait, it either returns to the ready state or enters the transition state. A thread in the transition state has had its kernel stack paged out of memory (e.g., because it has not executed in a while and the system needed the memory for other purposes), but the thread is otherwise ready to execute. The thread enters the ready state once the system pages the thread's kernel stack back into memory. The system places a thread in the unknown state when the system is unsure of the thread's state (usually because of an error).160, 161 Thread Scheduling Algorithm The dispatcher schedules threads based on priority—the next subsection describes how thread priorities are determined. When a thread enters the ready state, the kernel places it into the ready queue that corresponds to its priority. Windows XP has 32 priority levels, denoted by integers from 0 through 31, where 31 is the highest priority and 0 is the lowest. The dispatcher begins with the highest-priority ready queue and schedules the threads in this queue in a round-robin fashion. A thread 21.6 Process and Thread Management 1057 remains in a ready queue as long as the thread is in either the ready state or the running state. Once the queue empties, the dispatcher proceeds to the next queue; it continues in this manner until either all queues are empty or a thread with a higher priority than the currently executing thread enters its ready state. In the latter case, the dispatcher preempts execution of the lower-priority thread and executes the new thread. 162 The dispatcher then executes the first thread in the highest-priority nonempty ready queue and resumes its normal scheduling procedure. 163 Each thread executes for at most one quantum. In the case of preemption, real-time threads have their quantum reset, whereas all other threads finish their quantum when they reacquire a processor. Giving real-time threads fresh quanta after preemption allows Windows XP to favor real-time threads, which require a high level of responsiveness. The system returns preempted threads to the front of the appropriate ready queue. 164 Note that user-mode threads can preempt kernelmode threads in many cases. However, kernel-mode threads can prevent this by masking certain interrupts. Events such as a thread entering the ready state, a thread exiting the running state or a change in a thread's priority trigger the system to execute dispatcher routines—routines that execute at DPC/dispatch level. By raising the IRQL to the DPC/dispatch IRQL, kernel-mode threads can mask scheduling and avoid being preempted. However, user-mode threads can still block the execution of system threads if they have a higher priority than the system threads.165 Determining Thread Priority Windows XP divides its 32 priority levels (0-31) into two categories. Real-time threads (i.e., threads that must maintain a high level of responsiveness to user requests) occupy the upper 16 priority levels (16-31), and dynamic threads occupy the lower 16 priority levels (0-15). Only the zero-page thread has priority level 0. This thread uses spare processor cycles to zero free memory pages so they are ready for use. Each thread has a base priority, which defines the lower limit that its actual priority may occupy. A user-mode thread's base priority is determined by its process's base priority class and the thread's priority level. A process's base priority class specifies a narrow range that the base priority of each of a process's threads can have. There are six base priority classes: idle, below normal, normal, above normal, high and real-time. The first five of these priority classes (called the dynamic priority classes) encompass priority levels 0 through 15; these are called dynamic priority classes, because threads belonging to processes in these classes can have their priorities dynamically altered by the operating system. The threads belonging to processes in the real-time priority class have a priority between 16 and 31; realtime threads have a static priority. Within each priority class, there are several base priority levels: idle, lowest, below normal, normal, above normal, highest and time critical. Each combination of base priority class and base priority level maps to a specific base priority (e.g., a thread with a normal base priority level and a normal priority class has a base priority of 7). 166 1058 Case Study: Windows XP A dynamic-priority thread's priority can change. A thread receives a priority boost when the thread exits the waiting state, such as after the completion of an I/O or after the thread gains a handle to a resource for which the thread is waiting. Similarly, a window that receives input (such as from the keyboard, mouse or timer) gets a priority boost. The system also can reduce a thread's priority. When a thread executes until its quantum expires, its priority is reduced one unit. However, a thread's dynamic priority can never dip below its base priority or rise into the realtime range. 167 Finally, to reduce the likelihood of threads being indefinitely postponed, the dispatcher periodically (every few seconds) scans the lists of ready threads and boosts the priority of dynamic threads that have been waiting for a long time. This scheme also helps solve the problem of priority inversion. Priority inversion occurs when a high-priority thread is prevented from gaining the processor by a lower-priority thread. For example, a high-priority thread might wait for a resource held by a low-priority thread. A third, medium-priority thread obtains the processor, preventing the low-priority thread from running. In this way, the medium-priority thread is also preventing the high-priority thread from executing—hence, priority inversion. The dispatcher will eventually boost the priority of the low-priority thread so it can execute and, hopefully, finish using the resource. 168 Scheduling on Multiprocessors Multiprocessor thread scheduling extends the preceding uniprocessor scheduling algorithm. All editions of Windows XP, except the Home Edition, provide support for multiple processors. However, even Home Edition provides support for symmetric multiprocessing (SMP) using Intel's hyper-threading (HT) technology. Recall from Section 15.2.1 that HT technology allows the operating system to view one physical processor as two virtual processors. Scheduling in an SMP system is similar to that in a uniprocessor system. Generally, when a processor becomes available, the dispatcher tries to schedule the thread at the front of the highest-priority nonempty ready queue. However, the system also attempts to keep threads on the same processors to maximize the amount of relevant data stored in L1 caches (see Section 15.4, Memory Access Architectures). Processes and threads can specify the processors on which they prefer to execute. A process can specify an affinity mask, which is a set of processors on which its threads are allowed to execute. A thread can also specify an affinity mask, which must be a subset of its process's affinity mask.169 Similarly, a job also can have an affinity mask, which each of its associated processes must set as its own affinity mask. One use of affinity masks is restricting computationally intensive processes to a few processors so these processes do not interfere with more time-critical interactive processes. In addition to affinity masks, each thread stores its ideal processor and last processor. Manipulating the value of a thread's ideal processor allows developers to influence whether related threads should execute in parallel (by setting related 21.6 Process and Thread Management 1059 threads' ideal processors to different processors) or on the same processor to share cached data (by setting the threads' ideal processors to the same processor). By default, Windows XP attempts to assign threads of the same process different ideal processors. The dispatcher uses the last processor in an effort to schedule a thread on the same processor on which the thread last executed. This strategy increases the likelihood that data cached by the thread during an execution can be accessed during the next execution of that thread. 170, 171 When a processor becomes available, the dispatcher schedules threads by considering each thread's priority, ideal processor, last processor and how long a thread has been waiting. Consequently, the thread at the front of the highest-priority nonempty ready queue might not be scheduled to the next available processor (even if the processor is in the thread's affinity mask). If this thread has not been waiting long and the available processor is not its ideal or last processor, the dispatcher might choose another thread from that ready queue which meets one or more of these other criteria.172 21.6.3 Thread Synchronization Windows XP provides a rich variety of synchronization objects and techniques, designed for various purposes. For example, some synchronization objects execute at the DPC/dispatch level. These objects provide certain guarantees —a thread holding one of these objects will not be preempted—but restrict functionality (e.g., a thread holding one of these objects cannot access pageable data). Some synchronization objects are designed specifically for kernel-mode threads, whereas others are designed for all threads. In the following subsections, we introduce some synchronization mechanisms provided in Windows XP, how they are used and their benefits and drawbacks. Dispatcher Objects Windows XP provides a number of dispatcher objects that kernel- and user-mode threads can use for synchronization purposes. Dispatcher objects provide synchronization for resources such as shared data structures or files. These objects include many familiar synchronization constructs such as mutexes, semaphores, events and timers. Kernel threads use kernel dispatcher objects; synchronization objects available to user processes encapsulate these kernel dispatcher objects (i.e., synchronization objects translate many API calls made by user-mode threads into function calls for kernel dispatcher objects). A thread holding a dispatcher object executes at the passive IRQL. 173 A developer can use these objects to synchronize the actions of threads of the same process or different processes.174 Dispatcher objects can be in either a signaled state or unsignaled state (some dispatcher objects also have states relating to error conditions). The object remains in the unsignaled state while the resource for which the dispatcher object provides synchronization is unavailable. Once the resource becomes available, the object enters its signaled state. If a thread wishes to access a protected resource, the thread can call a wait function to wait for one or more dispatcher objects to enter a sig- 1060 Case Study: Windows XP naled state. When calling a wait function for a single object, the thread specifies the object on which it is waiting (by passing the object's handle) and a maximum wait time for the object. Multiple-object wait functions can be used when a thread must wait on more than one object. The thread can specify whether it is waiting for all the objects or any one of them. Windows XP supplies numerous variations to these generic wait functions; e.g., a thread can specify that it enters an alertable wait state, allowing it to process any APCs queued to it while waiting. After calling a wait function, a thread blocks and enters its wait state. When the required object enters its signaled state, one or more threads can access the resource (later in this section, we describe how a dispatcher object enters its signaled state). 175 In many cases, the kernel maintains FIFO queues for waiting resources, but kernel APCs can disrupt this ordering. Threads process kernel APCs immediately, and when a thread resumes its wait, it is placed at the end of the queue. 176 Event Objects Threads often synchronize with an event, such as user input or I/O completion, by using an event object. When the event occurs, the object manager sets the event object to the signaled state. The event object returns to the unsignaled state by one of two methods (specified by the object's creator). In the first method, the object manager sets the event object to the unsignaled state when one thread is released from waiting. This method can be used when only one thread should awaken (e.g.. to process the completion of an I/O operation). In the other method, the event object remains in the signaled state until a thread specifically sets the event to its unsignaled state; this allows multiple threads to awaken. This option can be used, for example, when multiple threads are waiting to read data. When a writing thread completes its operation, the writer can use an event object to signal all the waiting threads to awaken. The next time a thread begins writing, the thread can reset the event object to the unsignaled state. 177 Mutex Objects Mutex objects provide mutual exclusion for shared resources; they are essentially binary semaphores (see Section 5.6, Semaphores). Only one thread can own a mutex at a time, and only that thread can access the resource associated with the mutex. When a thread finishes using the resource protected by the mutex, it must release the mutex to transition it into its signaled state. If a thread terminates before releasing a mutex object, the mutex is considered abandoned. In this case the mutex enters an abandoned state. A waiting thread can acquire this abandoned mutex. However, the system cannot guarantee that the resource protected by the mutex is in a consistent state. To be safe, a thread should assume an error has occurred if it acquires an abandoned mutex. 178 Semaphore Objects Semaphore objects extend the functionality of mutex objects; they allow the creating thread to specify the maximum number of threads that can access a shared 21.6 Process and Thread Management 1061 resource.179 For example, a process might have a small number of preallocated buffers; the process can use a semaphore to synchronize access to these buffers.180 Semaphore objects in Windows XP are counting semaphores (see Section 5.6.3, Counting Semaphores). A semaphore object maintains a count, which is initialized to the maximum number of threads that can simultaneously access the pool of resources. The count is decremented every time a thread acquires access to the pool of resources protected by the semaphore and incremented when a thread releases the semaphore. The semaphore remains in its signaled state while its count is greater than zero; it enters the unsignaled state when the count drops to zero. A single thread can decrement the count by more than one by specifying the semaphore object in multiple wait functions. For example, a thread might use several of the process's buffers in separate asynchronous I/O operations. Each time the thread issues a buffered I/O request, the thread reacquires the semaphore (i.e., by specifying the semaphore object in a wait function), which decrements the semaphore's count. The thread must release the semaphore, which increments the semaphore's count, each time a request completes.181 Waitable Timer Objects Threads might need to perform operations at regular intervals (e.g., autosave a user's document) or at specific times (e.g., display a calendar item). For this type of synchronization, Windows XP provides waitable timer objects. These objects become signaled after a specified amount of time elapses. Manual-reset timer objects remain signaled until a thread specifically resets the timer. Auto-reset timer objects remain signaled only until one thread finishes waiting on the object. Waitable timer objects can be single use, after which they become deactivated, or they can be periodic. In the latter case, the timer object reactivates after a specified interval and becomes signaled once the specified time expires. Figure 21.4 lists Windows XP's dispatcher objects.182 Several other Windows XP objects, such as console input, jobs and processes, can function as dispatcher objects as well. For example, jobs, processes and threads are set to the signaled state when they terminate, and a console input object is set to the signaled state when the console's input buffer contains unread data. In these cases, threads use wait functions and treat these objects just like other dispatcher objects.183 Dispatcher Object Transitions from Unsignaled to Signaled State When Event Associated event occurs. Mutex Owner of the mutex releases the mutex. Semaphore Semaphore's count rises above zero. Waitable timer Specified amount of time elapses. Figure 21.4 | Dispatcher objects in Windows XP. 1062 Case Study: Windows XP Kernel Mode Locks Windows XP provides several mechanisms for synchronizing access to kernel data structures. If a thread is interrupted while accessing a shared data structure, it might leave the data structure in an inconsistent state. Unsynchronized access to a data structure can result in erroneous results. If the thread belongs to a user process, an application might malfunction; if the thread belongs to a kernel process, the system might crash. In a uniprocessor system, one solution to the synchronization problem is to raise the IRQL level above the one at which any component that might preempt the thread executes and access the data structure. The thread lowers the IRQL level when it completes executing the critical code.184 Raising and lowering the IRQL level is inadequate in a multiprocessor system—two threads, executing concurrently on separate processors, can attempt to access the same data structure simultaneously. Windows XP provides spin locks to address this problem. Threads holding spin locks execute at the DPC/dispatch level or DIRQL, ensuring that the holder of the spin lock is not preempted by another thread. Threads should execute the fewest instructions possible while holding a spin lock to reduce wasteful spinning by other threads. When a thread attempts to access a resource protected by a spin lock, it requests ownership of the associated spin lock. If the resource is available, the thread acquires the lock, accesses the resource, then releases the spin lock. If the resource is not available, the thread keeps trying to acquire the spin lock until successful. Because threads holding spin locks execute at the DPC/dispatch level or DIRQL, developers should restrict the actions these threads perform. Threads holding spin locks should never access pageable memory (because a page fault might occur), cause a hardware or software exception, attempt any action that could cause deadlock (e.g., attempt to reacquire that spin lock or attempt to acquire another spin lock unless the thread knows that this will not cause deadlock) or call any function that performs any of these actions.185 Windows XP provides generic spin locks and queued spin locks. A queued spin lock enforces FIFO ordering of requests and is more efficient than a normal spin lock,186 because it reduces the processor-to-memory bus traffic associated with the spin lock. With a normal spin lock, a thread continually attempts to acquire the lock by executing a test-and-set instruction (see Section 5.5.2, Test-and-Set Instruction) on the memory line associated with the spin lock. This creates bus traffic on the processor-to-memory bus. With a queued spin lock, a thread that releases the lock notifies the next thread waiting for it. Threads waiting for queued spin locks do not create unnecessary bus traffic by repeating test-and-set. Queued spin locks also increase fairness by guaranteeing FIFO ordering of requests, because the releasing thread uses a queue associated with the spin lock to notify the appropriate waiting thread. 187 Windows XP supports both spin locks (for legacy compatibility) and queued spin locks, but Microsoft recommends the use of the latter.188 Alternatively, kernel-mode threads can use fast mutexes, a more efficient variant of mutexes that operate at the APC level. Although they require less overhead 21.6 Process and Thread Management 1063 than mutexes, fast mutexes somewhat restrict the actions of the thread owning the lock. For example, threads cannot specify a maximum wait time to wait for a fast mutex—they can either wait indefinitely or not wait at all for an unavailable fast mutex. Also, if a thread attempts to acquire a fast mutex that it already owns, it creates a deadlock (dispatcher mutexes allow a thread to reacquire a mutex the thread already owns as long as it releases the mutex for each acquisition of the mutex).189 Fast mutexes operate at the APC IRQL, which masks APCs that might require the thread to attempt to acquire a fast mutex it already owns.190 However, some threads might need to receive APCs. For example, a thread might need to process the result of an asynchronous I/O operation; the system often uses APCs to notify threads that an asynchronous I/O request has been processed. For this purpose, Windows XP provides a variant of fast mutexes that execute at the passive IRQL.191 Another synchronization resource available only to kernel-mode threads is the executive resource lock. Executive resource locks have two modes: shared and exclusive. Any number of threads can simultaneously hold an executive resource lock in shared mode, or one thread can hold an executive resource lock in exclusive mode. This lock is useful for solving the readers-and-writers problem (see Section 6.2.4, Monitor Example: Readers and Writers). A reading thread can gain shared access to a resource if no other thread currently holds or is waiting for exclusive (i.e., write) access to the resource. A writing thread can gain exclusive access to a resource as long as no other thread holds either shared (i.e., read) or exclusive (i.e., write) access to it.192 Other Available Synchronization Tools Aside from dispatcher objects and kernel-mode locks, Windows XP provides to threads several other synchronization tools. Kernel-mode locks are not applicable in all situations—they cannot be used by user-mode threads, and many of them operate at an elevated IRQL, restricting what threads can do while holding them. Dispatcher objects facilitate general synchronization but are not optimized for specific cases. For example, dispatcher objects can be used by threads of different processes but are not optimized for use among threads of the same process. Critical section objects provide services similar to mutex objects but can be employed only by threads within a single process. Moreover, critical section objects do not allow threads to specify a maximum wait time (it is possible to wait indefinitely for a critical section object), and there is no way to determine if a critical section object is abandoned. However, critical section objects are more efficient than mutex objects because critical section objects do not transition to kernel mode if there is no contention. This optimization requires that critical section objects be employed only to synchronize threads of the same process. Also, critical section objects are implemented with a processor-specific test-and-set instruction, further increasing efficiency.193'194 A timer-queue timer is an alternate method for using waitable timer objects that allows threads to wait on timer events while executing other pieces of code. When a thread wishes to wait on a timer, it can place a timer in a timer queue and 1064 Case Study: Windows XP specify a function to perform when the timer becomes signaled. At that time, a worker thread from a thread pool performs the specified function. In cases where the thread waits on both a timer and another object, the thread should use a waitable timer object with a wait function.195 For variables shared among multiple threads, Windows XP provides interlocked variable access through certain functions. These functions provide atomic read and write access to variables, but they do not ensure the order in which accesses are made. For example, the I n t e r l o c k e d l n c r e m e n t function combines incrementing a variable and returning the result into a single atomic operation. This can be useful, for example, when providing unique ids. Each time a thread needs to obtain a unique id, it can call I n t e r l o c k e d l n c r e m e n t . This avoids a situation in which one thread increments the value of the id variable but is interrupted before checking the result. The interrupting thread might then access the id variable, causing two items to have the same id. Because I n t e r l o c k e d l n c r e m e n t both increments the variable and returns the result in an atomic operation, this situation will not occur.196 Windows XP also provides interlocked singly linked lists (SLists). which provide atomic insertion and deletion operations. 197 This section provided a broad survey of Windows XP's synchronization tools. However, Windows XP provides other tools; readers interested in more information should visit the MSDN Library at m s d n . m i c r o s o f t . c o m / l i b r a r y . In addition to these synchronization tools, threads can synchronize by communicating through various IPC mechanisms (described in Section 21.10, Interprocess Communication) or by queuing APCs to other threads. 21.7 Memory Management The Windows XP virtual memory manager (VMM) creates the illusion that each process has a 4GB contiguous memory space. Because the system allocates more virtual memory to processes than can fit into main memory, the VMM stores some of the data on disk in files called pagefiles. Windows XP divides virtual memory into fixed-size pages which it stores either in page frames in main memory or files on disk. The VMM uses a two-level hierarchical addressing system. Windows XP has two strategies for optimizing main memory usage and reducing disk I/O. The VMM uses copy-on-write pages (see Section 10.4.6, Sharing in a Paging System) and employs lazy allocation, which is a policy of postponing allocating pages and page table entries in main memory until absolutely necessary. However, when the VMM is forced to perform disk I/O, it prefetches pages from disk and places pages into main memory before they are needed (see Section 11.4. Anticipatory Paging). Heuristics ensure that the gain in the rate of disk I/O outweighs the cost of filling main memory with potentially unused pages. When main memory fills, Windows XP performs a modified version of the LRU page-replacement algorithm. The following sections describe the internals of the Windows XP memory management system. 21.7 21.7.1 Memory Organization Windows XP systems provide either a 32-bit or a 64-bit address space, depending on the processor and the edition of Windows XP. We limit our discussion to memory operations using the Intel IA-32 architecture (e.g., Intel Pentium and AMD Athlon systems), because the vast majority of today's Windows systems are built for that platform. See the MSDN Library for more information on Windows XP 64Bit Edition. 198 Windows XP allocates a unique 4GB virtual address space to each process. By default, a process can access only the first 2GB of its virtual address space. The system reserves the other two gigabytes for kernel-mode components—this space is called system space. 199 Physical memory is divided into fixed-size page frames, which are 4KB on a 32-bit system (except in the case when the system uses large pages as described later in this section). The VMM maintains a two-level memory map in the system portion of each process's virtual address space that stores the location of pages in main memory and on disk.200 The VMM assigns each process one page directory table. When a processor switches contexts, it loads the location of the new process's page directory table into the page directory register. The page directory table is composed of page directory entries (PDEs); each PDE points to a page table. Page tables contain page table entries (PTEs); each PTE points to a page frame in main memory or a location on disk. The VMM uses the virtual address in conjunction with the memory map to translate virtual addresses into physical addresses. A virtual address is composed of three portions; the offset in a page directory table, the offset in a page table and the offset on a page in physical memory 201 Windows XP translates a virtual address in three stages, as shown in Fig. 21.5. In the first stage of address translation, the system calculates the sum a + d, which is the value in the page directory register plus the first portion of the virtual address, to determine the location of the page directory entry (PDE) in the page directory table. In the second stage, the system calculates the sum b + t, which is the value in the PDE plus the second portion of the virtual address, to find the page table entry (PTE) in the page table. This entry contains c, the page frame number corresponding to the virtual page's location in main memory. Finally, in the last stage, the system concatenates c and the page offset, o, to form the physical address (see Section 10.4.4, Multilevel Page Tables). The system performs all of these operations quickly; the delay occurs when the system must read the PDE and PTE from main memory.202 Address translation is often accelerated by the Translation Look-aside Buffer (TLB) as discussed in Section 10.4.3, Paging Address Translation with Direct/Associative Mapping. The PTE is structured differently depending on whether it points to a page in memory or a page in a pagefile. Five of the PTE's 32 bits are for protection—they indicate whether a process may read the page, write to the page and execute code on the page. These protection bits also tell the system whether the page is a copyon-write page and whether the system should raise an exception when a process Memory Management 1065 1066 Case Study: Windows XP Figure 21.5 | Virtual address translation. accesses the page. 203 Twenty bits index a frame in memory or offset in a pagefile on disk; this is enough to address 1,048,576 virtual pages. If the page is stored on disk, a designated four bits indicate in which of 16 possible pagefiles the page is located. If the page is in memory, a different set of three bits indicate the state of the page. Figure 21.6 lists these three bits and their meaning. Windows XP allows applications to allocate large pages. A large page is a set of contiguous pages that the operating system treats as one page. Large pages are useful when the system knows it will repeatedly need the same large chunk of code or data. The system needs to store only one entry in the TLB. Because the page is bigger, it is more likely to be accessed, and less likely to be deleted from the TLB. Page access is accelerated because the VMM can look up address translations in the TLB rather than consulting page tables.204 Windows XP stores information that cannot be swapped to disk (i.e., the nonpaged pool) and memory maps on large pages.205 Page state bit Definition Valid PTE is valid—it points to a page of data. Modified Page in memory is no longer consistent with the version on disk. Transition VMM is in the process of moving the page to or from disk. Pages in transition are always invalid. Figure 21.6 | Page states.206 21.7 Memory Management 1067 Windows XP imposes several restrictions on the use of large pages: • Each type of processor has a minimum size for large pages, usually 2MB or greater. The size of every large page must be a multiple of that minimum large page size.207 • Large pages always allow read and write access. This means that read-only data, such as program code and system DLLs, cannot be stored in large pages.208 • The pages that compose a large page must be contiguous in both virtual and physical memory.209, 210 Windows XP uses copy-on-write pages (see Section 10.4.6, Sharing in a Paging System) as one of its lazy allocation mechanisms. The system manages copy-on-write pages using prototype page tables. The system also uses prototype page tables to manage file mappings. A file mapping is a portion of main memory that multiple processes may access simultaneously to communicate with one another (this is discussed further in Section 21.10.3, Shared Memory). The PTE of a copy-on-write page does not point directly to the frame in which the page is stored. Instead, it points to a Prototype Page Table Entry (PPTE), a 32-bit record that points to the location of the shared page. When a process modifies a page whose PTE protection bits indicate that it is a copy-on-write page, the VMM copies the page to a new frame and sets the process's PPTE to reference the new location. All of the other processes's PPTE's continue pointing to the original frame. Using copy-on-write pages conserves memory, because processes share main memory page frames. As a result, each process can store more of its pages in main memory. This causes fewer page faults, because there is more space in main memory, and once the VMM fetches a page from disk, it is less likely to move the page back to disk. However, PPTEs add a level of indirection; it takes four memory references instead of three to translate an address. Therefore, translating an address that is not in the TLB is more expensive for copyon-write pages than for normal pages.211 21.7.2 Memory Allocation Windows XP performs memory allocation in three stages (Fig. 21.7). A process must first reserve space in its virtual address space. Processes can either specify what virtual addresses to reserve or allow the VMM to decide. A process may not access space it has reserved until it commits the space. When a process commits a page, the VMM creates a PTE and ensures that there is enough space in either main memory or a pagefile for the page. A process usually commits a page only when it is ready to write to the page; however, the Windows API supports single-step reserve and commit. Separating memory allocation into two steps allows a process to reserve a large contiguous virtual memory region and sparsely commit portions of it.212. 213 Finally, when a process is ready to use its committed memory, it accesses the committed virtual memory. At that point, the VMM writes the data to a zeroed 1068 Case Study: Windows XP Figure 21.7 | Memory allocation stages. page in main memory. This three-stage process ensures that processes use only as much space in main memory as they need, rather than the amount they reserve.214 A system rarely has enough main memory to satisfy all processes. Previous versions of Windows allowed an application that required more main memory to 21.7 Memory Management 1069 function properly to issue must-succeed requests. Device drivers often issued these requests, and the VMM always fulfilled a must-succeed request. This led to system crashes when the VMM was forced to allocate main memory when none was available. Windows XP does not suffer from this problem; it denies all must-succeed requests. The system expects components to handle a denied memory allocation request without crashing.215, 216 Windows XP has a special mechanism for handling low-memory conditions when all or most of main memory is allocated. Under normal circumstances, Windows XP optimizes performance by handling multiple memory allocation requests simultaneously. When there are few page frames in main memory to allocate, the system runs into the same scarce-resource problem faced by all multiprogramming systems. Windows XP solves this problem via a process called I/O throttling; when the system detects that it has only a few available page frames, it begins to manage memory one page at a time. I/O throttling slows the system, because the VMM ceases to manage multiple requests in parallel but instead retrieves pages from disk only one at a time. However, it does make the system more robust and helps prevent crashes.217 To track main memory, Windows XP uses a page frame database, which lists the state of each frame, ordered by page frame number. There are eight possible states, as shown in Fig. 21.8. The system tracks page frames by state. It has a singly linked list, called a page list, for each of the eight states. A page list is referred to by the state of its pages; for example, the Free Page List contains all free pages, the Zeroed Page List contains all zeroed pages and so on. The page lists permit the VMM to swap pages and allocate memory quickly. For example, to allocate two new pages, the VMM Frame State Definition Valid Page belongs to a process's working set and its PTE is set to valid. Transition Page is in the process of being transferred to or from disk. Standby Page has just been removed from a process's working set; its PTE is set to invalid and in transition. Modified Page has just been removed from a process's working set; it is not consistent with the on-disk version. The VMM must write this page to disk before freeing this page. The PTE of this page is set to invalid and in transition. Modified No-Write Page has just been removed from a process's working set; it is not consistent with the on-disk version. The VMM must write an entry to the log file before freeing this page. The PTE of this page is set to invalid and in transition. Free Page frame does not contain a valid page; however it might contain an invalid page that has no PTE and is not part of any working set. Figure 21.8 | Page frame states.218(Part 1 of 2.) 1070 Case Study: Windows XP Frame State Definition Zeroed Page frame is not part of any working set and all of its bits have been set to zero. For security reasons, only zeroed page frames are allocated to processes. Bad Page frame has generated a hardware error and should not be used. Figure 21.8 | Page frame states 218 (Part 2 of 2.) needs to access only the first two frames in the Zeroed Page List. If the Zeroed Page List is empty, the VMM takes a page from another list, using the algorithm described in Section 21.7.3, Page Replacement. 219 The system uses Virtual Address Descriptors (VADs) to manage the virtual address space of each process. Each VAD describes a range of virtual addresses allocated to the process.220 Windows XP attempts to anticipate requests for pages on disk and move pages to main memory to prevent page faults. A process can invoke API functions to tell the system the process's future memory requirements. 221 In general, the system employs a policy of demand paging, loading pages into memory only when a process requests them. It also loads several nearby pages—spatial locality implies. that these pages are likely to be referenced soon. The Windows file systems divide disk space into clusters of bytes; pages in the same cluster are, by definition, part of the same file. Windows XP takes advantage of spatial locality by loading all pages in the same cluster at once, a paging policy referred to as clustered demand paging. These two mechanisms reduce disk seek time because only one disk seek is needed to fetch the entire cluster. Prefetching, however, potentially places some unneeded pages into main memory, leaving less space for needed pages and increasing the number of page faults. It might also decrease efficiency during periods of heavy disk I/O by forcing the system to page out pages that are needed right away for pages that are not needed yet.222 Windows XP reduces the time needed to load (i.e., start) applications, including the operating system itself, by prefetching files. The system records what pages are accessed and in what order during the last eight application loads. Before loading an application, the system asks for all of the pages it will need in one asynchronous request, thus decreasing disk seek time.223, 224 The system decreases the time it takes to load Windows by simultaneously prefetching pages and initializing devices. When Windows XP starts, it must direct numerous peripheral hardware devices to initialize—the system cannot continue loading until they are done. This is the ideal time to perform prefetching, because the system is otherwise idle.225 Windows XP uses the Logical Prefetcher to perform prefetching. The user can set a special registry key to tell Windows XP for which scenarios, such as all application loads or Windows start-up, the user wants the system to run the Logical 21.7 Memory Management 1071 Prefetches 226 When an application loads, Windows XP stores a trace of memory accesses in a scenario file. Each application, including the Windows operating system, has its own scenario file. The system updates scenario files ten seconds after a successful application load. Storing scenario files has a price: recording each memory access uses processor cycles and occupies space on the disk. However, the reduced disk I/O offsets the extra time it takes to store the memory trace, and most users are willing to give up some disk space in return for significantly faster load time for all applications, including for Windows itself.227 The Logical Prefetcher accesses pages based on their location on disk rather than waiting for a process to explicitly request them. As a result, Windows XP can load applications faster and more efficiently. However, the pages and files indicated in scenario files could be scattered throughout the disk. To further optimize application load time, the Logical Prefetcher periodically reorganizes portions of the hard drive to ensure that all files that the system has previously prefetched are contiguous on disk. Every few days, when the system is idle, the Logical Prefetcher updates a layout file which contains the ideal disk layout and uses this layout to rearrange pages on disk.228 21.7.3 Page Restatement Windows XP bases its page-replacement policy on the working set model. Recall from Section 11.7, Working Set Model, that a process's working set is the set of pages that a process is currently using. Because this is difficult to determine, Windows XP simply considers a process's working set to be all of its pages in main memory Pages not in the process's working set, but mapped to its virtual address space, are stored on disks in pagefiles.229, 230 Storing pagefiles on different physical disks accelerates swapping because it enables the system to read and write multiple pages concurrently 231 When memory becomes scarce, the balance set manager moves parts of different processes' working sets to pagefiles. Windows XP employs a localized leastrecently used (LRU) policy to determine which pages to move to disk. The policy is localized by process. When the system needs to free a frame to meet a process's request, it removes a page belonging to the process which the process has not recently accessed. The VMM assigns each process a working set maximum and a working set minimum that denote an acceptable range for the size of the process's working set. Whenever a process whose working set size is equal to its working set maximum requests an additional page, the balance set manager moves one of the process's pages to secondary storage, then fulfills the request. A process using less than its working set minimum is not in danger of losing pages unless available memory drops below a certain threshold. The balance set manager checks the amount of free space in memory once per second and adjusts the working set maximums accordingly. If the number of available bytes exceeds a certain threshold, the working set maximums of processes working at their working set maximum shift upward; if it drops below a certain threshold, all working set maximums shift downward.232 1072 Case Study: Windows XP Windows XP does not wait until main memory is full to move pages to disk. It regularly tests each process to ensure that only the pages the process needs are in main memory. Windows XP uses page trimming to remove pages from memory. Periodically, the balance set manager moves all of a process's pages above the working set minimum from the Valid Page List to the Standby Page List, Modified Page List, or Modified No-Write Page List, as appropriate. Collectively, these are called the transitional page list, although there is no such actual list in memory.233 The VMM chooses which pages to trim, using the localized LRU policy, as illustrated in Fig. 21.9. The system sets the status bits of the PTEs of trimmed pages to invalid, meaning the PTEs no longer point to a valid frame, but the system does not modify the PTEs in any other way. The state in the page frame database of trimmed pages is set to transition, standby, modified or modified no-write.234 If the process requests a page that is in one of these transitional page lists, the MMU issues a transitional fault. In this situation, the VMM removes the page from the transitional list and puts it back on the Valid Page List. Otherwise, the page is reclaimed. A system thread writes pages in the Modified Page List to disk. Components, such as file systems, use the Modified No-Write Page List to ensure that the VMM does not write a dirty page to disk without first making an entry in a log file. The application notifies the VMM when it is safe to write the page.235 Once a modi- Figure 21.9 | Page-replacement process. 21.8 File Systems Management fied or modified no-write page becomes consistent with the data on disk, the VMM moves the page to the Standby Page List. After a waiting period, the VMM moves pages on the Standby Page List to the Free Page List. Later, another low-priority thread zeroes the bits of pages on the Free Page List and moves the pages to the Zeroed Page List. Windows XP zeroes pages to prevent a process that is allocated a page from reading the (potentially sensitive) information that another process previously stored on that page. When the system needs to allocate a new page to a process, it claims a page from the Zeroed Page List.236 The localized LRU algorithm requires certain hardware features not available on all systems. Because portability between different platforms is an important design criterion for all Windows operating systems, many early Windows systems avoided using hardware-specific features to ensure that the same operating system could be used on multiple platforms. Operating systems commonly simulate LRU page replacement with the clock algorithm (see Section 11.6.7, Modifications to FIFO: Second-Chance and Clock Page Replacement). Doing so requires an accessed bit which the system sets on for each page that has been accessed within a certain time interval. Many early computers did not have an accessed bit. As a result, early versions of Windows NT used the FIFO page-replacement policy (see Section 11.6.2, First-In-First-Out (FIFO) Page Replacement) or even an effectively random page replacement. Microsoft later developed versions of

Windows specifically for Intel IA-32 and other platforms that support the accessed bit, allowing Windows XP to employ the more effective LRU algorithm.237 Some data cannot be paged out of memory. For example, code that handles interrupts must stay in main memory, because processing a page fault during an interrupt would cause an unacceptable delay and might even crash the system. Paging passwords presents a security risk if the system crashes, because an unencrypted copy of secret data would be stored on disk. All such data is stored in a designated area of system space called the nonpaged pool. Data that can be sent to disk is stored in the paged pool.238 Portions of device drivers and VMM code are stored in the nonpaged pool. 239 A device driver developer must consider several trade-offs when deciding what portions of the code to place in the nonpaged pool. Space in the nonpaged pool is limited. In addition, code in the nonpaged pool may not access paged pool code or data because the VMM might be forced to access the disk, which defeats the purpose of nonpaged pool code.240 21.8 File Systems Management Windows XP file systems consist of three layers of drivers. At the very bottom are various volume drivers that control a specific hardware device, such as the hard disk. File system drivers, which compose the next level, implement a particular file system format, such as New Technology File System (NTFS) or File Allocation Table (FAT). These drivers implement what a typical user views as the file system: a 1073 1074 Case Study: Windows XP hierarchical organization of files and the related functions to manipulate these files. Finally, file system filter drivers accomplish high-level tasks such as virus protection, compression and encryption.241242 NTFS is the native Windows XP file system (and the one described in depth in this case study), but FAT16 and FAT32 are also supported (see Section 13.6.3, Tabular Noncontiguous File Allocation) and typically used for floppy disks. Windows XP also supports Compact Disc File System (CDFS) for CDs and Universal Disk Format (UDF) for CDs and DVDs. The following subsections introduce Windows XP file system drivers and describe the NTFS file system.243 21.8.1 File System Drivers File system drivers cooperate with the I/O manager to fulfill file I/O requests. File system drivers provide a link between the logical representation of the file system exposed to applications and its corresponding physical representation on a storage volume. The I/O manager interface enables Windows XP to support multiple file systems. Windows XP divides each data storage device into one or more volumes, and associates each volume with a file system. To understand how file system drivers and the I/O manger interact, consider the flow of a typical file I/O request. A user-mode thread that wishes to read data from a file sends an I/O request via a subsystem API call. The I/O manager translates the thread's file handle into a pointer to the file object and passes the pointer to the appropriate file system driver. The file system driver uses the pointer to locate the file object and determine the location of the file on disk. Next, the file system driver passes the read request through the layers of drivers and eventually the request reaches the disk. The disk processes the read request and returns the requested data. 244 A file system driver can be either a local file system driver or a remote file system driver. Local file system drivers process I/O for hardware devices such as hard disk drives or DVD drives. The preceding paragraph described the role of a local file system driver in fulfilling a disk I/O request. Remote file system drivers transfer files to and from remote file servers via network protocols. Remote file system drivers cooperate with the I/O manager, but instead of interacting with volume drivers, remote file system drivers interact with remote file system drivers on other computers. 245 In general, a file system driver can act as a black box; it provides support for a particular file system, such as NTFS, independent of the underlying storage volume on which the files reside. File system drivers provide the link between the user's view of a file system and the data's actual representation on a storage volume. 21.8.2 NTFS NTFS is the default file system for Windows XP. When creating the NT line of operating systems, Microsoft decided to construct a new file system to address the limitations of the FAT file system used in DOS and older versions of Windows. FAT. 21.8 File Systems Management which uses a tabular noncontiguous allocation scheme (see Section 13.6.3, Tabular Noncontiguous File Allocation), does not scale well to large disk drives. For example, a file allocation table for FAT32 (the file system included in Windows ME) on a 32GB hard drive using 2KB clusters consumes 64MB of memory. Furthermore, FAT32 can address no more than 2 32 data blocks. FAT's addressing limitation was a problem for FAT12 (12-bit FAT) and FAT16 (16-bit FAT) and undoubtedly will be a problem in the future for FAT32. To avoid these shortcomings, NTFS uses an indexed approach (see Section 13.6.4, Indexed Noncontiguous File Allocation) with 64-bit pointers. This allows NTFS to address up to 16 exabytes (i.e., 16 billion gigabytes) of storage. Furthermore, NTFS includes additional features that make a file system more robust. These features include file compression, file encryption, support for multiple data streams and user-level enhancements (e.g., support for hard links and easy file system and directory browsing). Master File Table The most important file on an NTFS volume is the Master File Table (MFT). The MFT stores information about all files on the volume, including file metadata (e.g., time of creation, the filename and whether the file is read-only, archive, etc.). The MFT is divided into fixed-size records, usually 1KB long. Each file has an entry in the MFT, which consists of at least one record, plus additional ones if necessary.246 NTFS stores all information about a file in attributes, each consisting of a header and data. The header contains the attribute's type (e.g., f i 1 e_name), name (necessary for files with more than one attribute of the same type) and flags. If any of a file's attributes do not fit into the file's first MFT record, NTFS creates a special attribute that stores pointers to the headers of all attributes located on different records. An attribute's data can be of variable length. Attribute data that fits in an MFT entry is stored in the entry for quick access; attributes whose data reside in the MFT entry are called resident attributes. Because the actual file data is an attribute, the system stores small files entirely within their MFT entry. NTFS stores nonresident attributes, the attributes whose data does not fit inside the MFT entry, elsewhere on disk.247 The system records the location of nonresident attributes using three numbers. The logical cluster number (LCN) tells the system on which cluster on disk the attribute is located. The run length indicates how many clusters the attribute spans. Because a file attribute might be split into multiple fragments, the virtual cluster number (VCN) tells the system to which cluster in the attribute the LCN points. The first VCN is zero. 248 Figure 21.10 shows a sample multirecord MFT entry that contains both resident and nonresident attributes. Attribute 3 is a nonresident attribute. The data portion of attribute 3 contains a list of pointers to the file on disk. The first 10 clusters are stored on disk clusters (i.e., LCNs) 1023-1032. The eleventh cluster, VCN 10, is stored on disk cluster 624. Attributes 1 and 2 are resident attributes, so they stored on the first MFT record; attributes 3 and 4 are nonresident attributes stored on the second MFT record. 1075 1076 Case Study: Windows XP Figure 21.10 | Master File Table (MFT) entry for a sample file. NTFS stores directories as files. Each directory file contains an attribute called index that stores the list of files inside the directory. Each file's directory entry contains the file's name and standard file information, such as the time of the most recent modification to the file and the file's access (e.g., read-only). If the index attribute is too large to be a resident attribute, NTFS creates index buffers to contain the additional entries. An additional attribute, the index_root, describes where on disk any index buffers are stored. For easy directory searching, NTFS sorts directory entries alphabetically. NTFS structures a large directory (one that is not a resident attribute) in a two-level B-tree. Figure 21.11 shows a sample directory. Suppose the system directs NTFS to find the file e.txt. NTFS begins searching in the i ndex attribute, where it finds a.txt. The file a . t x t comes before e.txt, so NTFS scans to the next file, j . txt. Because j . t x t comes after e.txt, NTFS proceeds to the appropriate child node of a.txt. NTFS searches in the first index buffer, obtaining its location on the volume from information in the i ndex_root attribute. This index buffer begins with b.txt; NTFS scans the first index buffer until it finds e.txt. 249 Windows XP allows multiple directory entries (either in the same directory or in different directories) to point to the same physical file. The system can create a new path (i.e., a new directory entry) to an already existing file, i.e., a hard link (see Section 13.4.2, Metadata). It first updates the existing file's MFT entry by adding a new file_name attribute and incrementing the value of the hard_link attribute. Next, it creates the new directory entry to point to the MFT entry of the existing file. NTFS treats each newly created hard link to a file exactly as it treats the original file name. If a user "moves" the file, the value of a file_name attribute is changed from the old to the new path name. NTFS does not delete the file until all hard links to the file are removed. It does not permit hard links to directory files. 21.8 File Systems Management 1077 Figure 21.11 | Directory contents are stored in B-trees. and because hard links point directly to MFT entries, a hard link must point to a file residing on the same physical volume.250, 251 Data Streams The contents (i.e., the actual file data) of an NTFS file are stored in one or more data streams. A data stream is simply an attribute in an NTFS file that contains file data. Unlike many other file systems, including FAT, NTFS allows users to place multiple data streams into one file. The unnamed default data stream is what most people consider to be the "contents" of a file. NTFS files can have multiple alternate data streams. These streams can store metadata about a file, such as author, summary or version number. They also can be a form of supplemental data, such as a small preview image associated with a large bitmap or a backup of a text file. NTFS considers each data stream of a file to be an attribute of type Data. Recall that header information for an attribute stores both the attribute type (in this case Data) and the attribute's name. NTFS differentiates between different alternate data streams using the attribute's name; the default data stream is unnamed. Using separate attributes for each data stream enables the system to access any data stream without scanning through the rest of the file.252 File Compression NTFS permits users to compress files and folders through the GUI interface or via system calls. When a user compresses a folder, the system compresses any file or 1078 Case Study: Windows XP folder that is added to the compressed folder. Compression is transparent; applications can open, manipulate and save compressed files using the same API calls as they would with uncompressed files. This capability exists because NTFS decompresses and recompresses files at the system level; i.e., the system reads the compressed file and decompresses it in memory, and then recompresses when the file is saved.253 NTFS uses the Lempel-Ziv algorithm to compress files.254 Lempel-Ziv is one of the most commonly used compression algorithms. For example, the Windows application WinZip and the application gzip, which is popular on UNIX-compatible systems, both use Lempel-Ziv compression.255 NTFS uses segmented compression, dividing files into compression units and compressing one unit at a time. A compression unit consists of 16 clusters; on most systems the cluster size is 2KB or 4KB, but clusters can range anywhere from 512 bytes to 64KB. Sometimes compressing a portion of the file does not significantly reduce its size. In that case, compression and decompression add unnecessary overhead. If the compressed version of a compression unit still occupies 16 clusters, NTFS stores the compression unit in its uncompressed state. 256 Although larger files (or sections of files) tend to compress better than small files, this segmented approach decreases file access time. Segmented compression enables applications to perform random file I/O without decompressing the entire file. Segmented compression also enables NTFS to compress a file while an application modifies it. Modified portions of the file are cached in memory. Because a portion of the file that has been modified once is likely to be modified again, constantly recompressing the file while it is open is inefficient. A low-priority lazy-writer thread is responsible for compressing the modified data and writing it to disk.257 File Encryption NTFS performs file encryption, like file compression, by dividing the file into units of 16 clusters and encrypting each unit separately. Applications can specify that a file should be encrypted on disk when created, or if the user has sufficient access rights, an application executing on the user's behalf can encrypt an existing file. The system will encrypt any file or folder for a user-mode thread, except system files, system folders and root directories. Because these files and folders are shared among all users, no user-mode thread can encrypt them (and therefore not allow other users access to them).258 NTFS uses a public/private key pair (see Section 19.2, Cryptography) to encrypt files. NTFS creates recovery keys that administrators can use to decrypt any file. This prevents files from being permanently lost when users forget their private keys or change jobs.259 NTFS stores encryption keys in the nonpaged pool in system memory. This is done for security reasons to ensure that the encryption keys are never written to disk. Storing keys on disk would compromise the security of the storage volume, because malicious users could first access the encryption keys, then use them to 21.8 File Systems Management 1079 decrypt the data.260 Instead of storing the actual keys, NTFS encrypts private keys using a randomly generated "master key" for each user; the master key is a symmetric key (i.e., it does both encryption and decryption). The private key is stored on disk in this encrypted form. The master key itself is encrypted by a key generated from the user's password and stored on disk. The administrator's recovery key is stored on disk in a similar manner.261 When NTFS encrypts a file, it encrypts all data streams of that file. NTFS's encryption and decryption of files is transparent to applications. Note that NTFS encryption ensures that files residing on a secondary storage volume are stored in an encrypted form. If data is stored on a remote server, the user must take extra precautions to protect it during network transfer, such as using a Secure Sockets Layer (SSL) connection.262 Sparse Files A file in which most of the data is unused (i.e., set to zero), such as an image file that is mostly white space, is referred to as a sparse file. Sparse files can also appear in databases and scientific data with sparse matrices. A 4MB sparse file might contain only 1MB of nonzero data; storing the 3MB of zeroes on disk is wasteful. Compression, although an option, degrades performance, owing to the need to decompress and compress the files when they are used. Therefore, NTFS provides support for sparse files.263 A thread explicitly specifies that a file is sparse. If a thread converts a normal file to a sparse file, it is responsible for indicating areas of the file that contain large runs of zeroes. NTFS does not store these zeroes on disk but keeps track of them in a zero block list for the file. Each entry in the list contains the start and end position of a run of zeroes in the file. The used portion of the file is stored in the usual manner on disk. Applications can access just the used file segments or the entire file, in which case NTFS generates streams of zeroes where necessary.264 When an application writes new data to a sparse file, the writing thread is responsible for indicating areas of the data that should not be stored on disk but rather in the zero block list. If a write contains an entire compression unit of zeroes (i.e., 16 clusters of zeroes), NTFS recognizes this and records the empty block in the zero block list instead of writing the compression unit full of zeroes to disk. (The write operation must be done using a special API call that notifies NTFS that the written data consists exclusively of zeroes.) As a result, while an application can specify exact ranges of zeroes for better memory management, the system also performs some memory optimizations on its own.265 Resparse Points and Mounted Volumes Recall our earlier discussion of NTFS hard links from the Master File Table subsection. We noted that they are limited because they cannot point to directories, and the data to which they point must reside on the same volume as the hard link. NTFS includes a file attribute, called a reparse point, to address these limitations. A rep- 1080 Case Study: Windows XP arse point contains a 32-bit tag and can contain up to 16KB of attribute data. When the system accesses a file with a reparse point, the system first processes the information in the reparse point. NTFS uses the tag to determine which file system filter driver should handle the data in the reparse point. The appropriate filter driver reads the attribute data and performs some function, such as scanning for viruses, decrypting a file or locating the file's data.266 Reparse points permit applications to establish links to files on another volume. For example, little-used hard disk data is sometimes moved to a tertiary storage device such as a tape drive. Although NTFS might delete the file's data after it is copied to the tertiary storage (to save space on the hard disk), it retains the file's MFT record. NTFS also adds a reparse point specifying from where to retrieve the data. When an application accesses the data, NTFS encounters the reparse point and uses its tag to call the appropriate file system driver, which retrieves the data from tertian storage and copies it back to disk. This operation is transparent to the application.267 Additionally, reparse points are used to mount volumes. The reparse point data specifies the root directory of the volume to mount and how to find the volume, and the file system driver uses this information to mount that volume. In this way, a user can browse a single directory structure that includes multiple volumes. A reparse point can associate any directory within an NTFS volume with the root directory of any volume; this directory is called a mounting directory. NTFS redirects all access to the mounting directory to the mounted volume.268 A reparse point can be used to create a directory junction—a directory referring to another directory, similar to a symbolic directory link in Linux. The reparse point specifies the pathname of the directory to which the directory junction refers. This is similar to mounting volumes, except that both directories must be within the same volume. The referring directory must be empty; a user must remove the directory junction before inserting files or folders into the referring directory.269 21.9 Input/Output Management Managing input/output (I/O) in Windows XP involves many operating system components (Fig. 21.12). User-mode processes interact with an environment subsystem (such as the Win32 subsystem) and not directly with kernel-mode components. The environment subsystems pass I/O requests to the I/O manager, which interacts with device drivers to handle such requests. Often, several device drivers, organized into a driver stack, cooperate to fulfill an I/O request.270 The Plug and Play (PnP) manager dynamically recognizes when new devices are added to the system (as long as these devices support PnP) and allocates and deallocates resources, such as I/O ports or DMA channels, to them. Most recently developed devices support PnP. The power manager administers the operating system's power management policy. The power policy determines whether to power down devices to conserve energy or keep them fully powered for high responsiveness.271 We describe the PnP manager and power manager in more detail later in this section. First, we describe how these components and device drivers cooperate to manage I/O in Windows XP. 21.9 Input/Output Management 1081 21.9.1 Device Drivers Windows XP stores information about each device in one or more device objects. A device object stores device-specific information (e.g., the type of device and the current I/O request being processed) and is used for processing the device's I/O requests. Often, a particular device has several device objects associated with it, because several drivers, organized into a driver stack, handle I/O requests for that device. Each driver creates a device object for that device.272 The driver stack consists of several drivers that perform different tasks for I/O management. A low-level driver—a driver that interacts most closely with the HAL—controls a peripheral device and does not depend on any lower-level drivers; a bus driver is a low-level driver. A high-level driver abstracts hardware specifics and passes I/O requests to low-level drivers. An NTFS driver is a high-level driver that abstracts how data is stored on disk. An intermediate driver can be interposed between high- and low-level drivers to filter or process I/O requests and 1082 Case Study: Windows XP export an interface for a specific device; a class driver (i.e., a driver that implements services common to a class of devices) is an intermediate driver. For example, a driver stack that handles mouse input might include a PCI bus driver, a USB controller driver, a USB hub driver, a driver that performs mouse acceleration and a driver that provides the user interface and processes incoming mouse data. The drivers cooperate to process mouse input. The drivers for the various hardware buses (i.e., the PCI bus, USB controller and USB hub) process interrupts and handle PnP and power management for the mouse; the intermediate driver manipulates the data, providing mouse acceleration; and the high-level driver translates the incoming data and sends it to a user-level GUI application.273 These driver types all refer to kernel-mode drivers. Other drivers, such as some printer drivers, can be user-mode drivers; these drivers execute in user space and are specific to an environment subsystem.274 Kernel-mode drivers operate in the thread context of the currently executing thread, implying that developers cannot assume that a driver will execute in the context of the same thread each time driver code executes. However, drivers need to store context information such as device state information, driver data and handles to kernel objects. Drivers store this information in a device extension. Windows XP allocates a portion of the nonpaged memory for drivers to store device extensions. Each device object points to a device extension, and a driver uses the device extension for each device it serves to store information and objects needed to process I/O for that device.275 The system represents a device driver with a driver object which stores the device objects for the devices the driver services. The driver object also stores pointers to standard driver routines.276 Many of the standard driver routines must be implemented by all kernel-mode drivers. These include functions for adding a device, unloading the driver and certain routines for processing I/O requests (such as read and write). Other standard driver routines are implemented only by certain classes of drivers. For example, only low-level drivers need to implement the standard routine for handling interrupts. In this way, all drivers expose a uniform interface to the system by implementing a subset of the standard driver routines.277 Plug and Play (PnP) Plug and Play (PnP), introduced in Section 2.4.4, describes the ability of a system to dynamically add or remove hardware components and redistribute resources (e.g.. I/O ports or DMA channels) appropriately. A combination of hardware and software support enables this functionality. Hardware support involves recognizing when a user adds or removes components from a system and easily identifying these components. Both the operating system and third-party drivers must cooperate to provide software support for PnP. Windows XP support for PnP involves utilizing information provided by hardware devices, the ability to dynamically allocate resources to devices and a programming interface for device drivers to interact with the PnP system.278 21.9 Input/Output Management 1083 Windows XP implements PnP with the PnP manager, which is divided into two components, one in user space and the other in kernel space. The kernel-mode PnP manager configures and manages devices and allocates system resources. The user-mode PnP manager interacts with device setup programs and notifies usermode processes when some device event has occurred for which the process has registered a listener. The PnP manager automates most device maintenance tasks and adapts to hardware and resource changes dynamically.279 Driver manufacturers must adhere to certain guidelines to support PnP for Windows XP. The PnP manager collects information from drivers by sending driver queries. These queries are called PnP I/O requests, because they are sent in the form of I/O request packets (IRPs). IRPs are described in Section 21.9.2, Input/ Output Processing. PnP drivers must implement functions that respond to these PnP I/O requests. The PnP manager uses information from the requests to allocate resources to hardware devices. PnP drivers must also refrain from searching for hardware or allocating resources, because the PnP manager handles these tasks.280 Microsoft urges all driver vendors to support PnP, but Windows XP supports nonPnP drivers for legacy compatibility.281 Power Maanagement The power manager is the executive component that administers the system's power policy.282 The power policy dictates how the power manager administers system and device power consumption. For example, when the power policy emphasizes conservation, the power manager attempts to shut down devices that are not in use. When the power policy emphasizes performance, the power manager leaves these devices on for quick response times.283 Drivers that support power management must be able to respond to queries or directives made by the power manager.284 The power manager queries a driver to determine whether a change in the power state of a device is feasible or will disrupt the device's work. The power manager also can direct a driver to change the power state of a device.285 Devices have power states DO, D1, D2 and D3. A device in state DO is fully powered, and a device in D3 is off.286 In states Dl and D2, the device is in a sleep (i.e., low-powered) state, and when the device returns to the DO state, the system will need to reinitialize some of its context. A device in Dl retains more of its context and is in a higher-powered state than a device in D2.287 In addition to controlling devices' power states, the power manager controls the overall system's power state, which is denoted by values S0-S5. State SO is the fully powered working state, and S5 denotes that the computer is off. States S1-S4 represent states in which the computer is on, but in a sleeping state; a system in S1 is closest to fully powered, and in S4 is closest to off.288 Windows Driver Model (WDM) Microsoft has defined a standard driver model—the Windows Driver Model (WDM)—to promote source-code compatibility across all Windows platforms. 1084 Case Study: Windows XP Windows XP supports non-WDM drivers to maintain compatibility with legacy drivers, but Microsoft recommends that all new drivers be WDM drivers. This section briefly outlines the WDM guidelines.289 WDM defines three types of device drivers. Bus drivers interface with a hardware bus, such as a SCSI or PCI bus, provide some generic functions for the devices on the bus, enumerate these devices and handle PnP I/O requests. Each bus must have a bus driver, and Microsoft provides bus drivers for most buses.290 Filter drivers are optional and serve a variety of purposes. They can modify the behavior of hardware (e.g., provide mouse acceleration or enable a joystick to emulate a mouse) or add additional features to a device (e.g., implement security checks or merge audio data from two different applications for simultaneous playback). Additionally, filter drivers can sort I/O requests among several devices. For example, a filter driver might serve multiple storage devices and sort read and write requests between the devices. Filter drivers can be placed in numerous locations in the device stack—a filter driver that modifies hardware behavior is placed near the bus driver, whereas one that provides some high-level feature is placed near the user interface.291 A function driver implements a device's main function. A device's function driver does most of the I/O processing and provides the device's interface. Windows XP groups devices that perform similar functions into a device class (such as the printer device class).292 A function driver can be implemented as a class/miniclass driver pair.293 A class driver provides generic processing for a particular device class (e.g., a printer); a miniclass driver provides the functionality specific to a particular device (e.g., a particular printer model). Typically, Microsoft provides class drivers, and software vendors provide the miniclass drivers.294 All WDM drivers must be designed as either a bus driver, filter driver or function driver.295 Additionally, WDM drivers must support PnP, power management and Windows Management Instrumentation (WMI).296 Drivers that support WMI provide users with measurement and instrumentation data. This data includes configuration data, diagnostic data and custom data. Also, WMI drivers permit user applications to register for WMI driver-defined events. The main purpose of WMI is to provide hardware and system information to user processes and allow user processes greater control in configuring devices.297 21.9.2 Input/Output Processing Windows XP describes I/O requests with I/O request packets (IRPs). IRPs contain all the information necessary to process an I/O request. An IRP (Fig. 21.13) consists of a header block and an I/O stack. Most header block information does not change during request processing. The header block records such information as the requestor's mode (either kernel or user), flags (e.g., whether to use caching) and information about the requestor's data buffer.298 The header block also contains the I/O status block, which indicates whether an I/O request completed successfully or. if not, the request's error code.299 21.9 Input/Output Management 1085 Figure 21.13 | I/O request packet (IRP). The I/O stack portion of an IRP contains at least one different stack location for each driver in the device stack of the target device. Each I/O stack location contains specific information that each driver needs in order to process the IRP, in particular the IRP's major function code and the minor function code. Figure 21.14 lists several major function codes. The major function code specifies the general IRP function such as read (IRP_MJ_READ) or write (IRP_MJ_WRITE). In some cases, it designates a class of IRPs, such as IRP_MJ_PNP, which specifies that the IRP is a PnP Major function code Typical reason to send an IRP with this major function code IRP_MJ_READ User-mode process requests to read from a file. IRP_MJ_WRITE User-mode process requests to write to a file. IRP_MJ_CREATE User-mode process requests a handle to a file object. IRP_MJ_CLOSE All handles to a file object have been released and all outstanding I/O requests have been completed. IRP_MJ_POWER Power manager queries a driver or directs a driver to change the power state of a device. Figure 21.14 | Major function code examples in Windows XP. (Part 1 of 2.) 1086 Case Study: Windows XP Major function code Typical reason to send an IRP with this major function code IRP_MJ_PNP PnP manager queries a driver, allocates resources to a device or directs a driver to perform some operation. IRP_MJ_DEVICE_CONTROL User-mode process calls a device I/O control function to retrieve information about a device or direct a device to perform some operation (e.g., format a disk). Figure 21.14 | Major function code examples in Windows XP. (Part 2 of 2.) I/O request. The minor function code then indicates the particular I/O function to perform.300 For example, the PnP manager starts a device by specifying the major function code IRP_MD_PNP and the minor function code IRP_MN_START_DEVICE. Each I/O stack location contains several other fields, including a pointer to the target device object and some driver-specific parameters.301 Each driver accesses one location in the stack, and before passing the IRP to the next driver, the currently executing driver initializes the stack location of the next driver by assigning each field in the stack a value.302 Processing an I/O Request The I/O manager and device drivers cooperate to fulfill an I/O request. Figure 21.15 describes the path an IRP travels as the system fulfills a read I/O request from a usermode process. First, a thread passes the I/O request to the thread's associated environment subsystem (1). The environment subsystem passes this request to the I/O manager (2). The I/O manager interprets the request and builds an IRP (3). Each driver in the stack accesses its I/O stack location, processes the IRP, initializes the stack location for the next driver and passes the IRP to the next driver in the stack (4 and 5). Additionally, a high-level or intermediate driver can divide a single I/O request into smaller requests by creating more IRPs. For example, in the case of a read to a RAID subsystem, the file system driver might construct several IRPs, enabling different parts of the data transfer to proceed in parallel.303 Any driver, except a low-level driver, can register an I/O completion routine with the I/O manager. Low-level drivers complete the I/O request and, therefore, do not need I/O completion routines. When processing of an IRP completes, the I/O manager proceeds up the stack, calling all registered I/O completion. A driver can specify that its completion routine be invoked if an IRP completes successfully, produces an error and/or is canceled. An I/O completion routine can be used to retry an IRP that fails.304 Also, drivers that create new IRPs are responsible for disposing of them when processing completes; they accomplish task by using an I/O completion routine.305 Once the IRP reaches the lowest-level driver, this driver checks that the input parameters are valid, then notifies the I/O manager that an IRP is pending for a particular device (6). The I/O manager determines whether the target device is available.306 If it is, the I/O manager calls the driver routine that handles the I/O operation (7). The driver routine cooperates with the HAL to direct the device to 21.9 Input/Output Management 1087 Figure 21.15 | Servicing an IRP. perform an action on behalf of the requesting thread (8). Finally, the data is sent to the calling process's address space (9). The different methods for transferring I/O are described later in this section. If the device is not available, the I/O manager queues the IRP for later processing.307 Section 21.9.3, Interrupt Handling, describes how Windows XP completes I/O processing. Synchronous and Asynchronous I/O Windows XP supports both synchronous and asynchronous I/O requests. Microsoft uses the term overlapped I/O to mean asynchronous I/O. If a thread issues a synchronous I/O request, the thread enters its wait state. Once the device services the I/O request, the thread enters its ready state, and when it obtains the processor, it 1088 Case Study: Windows XP completes I/O processing. However, a thread that issues an asynchronous I/O request can continue executing other tasks while the target device services the request.308 Windows XP provides several options for managing asynchronous I/O requests. A thread can poll the device to determine whether an I/O has completed. Polling can be useful when a thread is executing a loop. In this case, the thread can perform useful work while waiting for the I/O to complete and process the completion of the I/O as soon as it occurs. Alternatively, a thread can employ an event object and wait for this object to enter its signaled state.309 This method is similar to synchronous I/O, except the thread does not enter the wait state immediately, but instead might perform some processing. After completing this processing, the thread waits for the completion of the I/O just as with synchronous I/O. As a third option, a thread can perform alertable I/O. When the system completes servicing an alertable I/O request, it queues an APC to the requesting thread. The next time this thread enters an alertable wait state, it processes this APC.310 Additionally, threads can accomplish asynchronous I/O using an I/O completion port. Several file handles can be associated with an I/O completion port. When I/O processing completes for a file associated with one of these handles, the I/O manager queues an I/O completion packet to the I/O completion port. A number of threads register with the port and block, waiting for these packets. When a packet arrives, the system awakens one of these threads to complete the I/O processing.311 Data Transfer Techniques A driver can choose from three methods for transferring data between a device and a process's memory space by setting flags in the device object for that device.312 If the driver chooses buffered I/O, the I/O manager allocates system pages to form a buffer equal to the size of the requesting thread's buffer. On read requests, the driver reads the data into the system buffer. The I/O manager transfers the data to the thread's buffer the next time the requesting thread gains the processor.313 For writes, the I/O manager transfers the data from the thread's buffer to the newly allocated system buffer before passing the IRP to the top-level driver.314 The system buffer does not act as a cache; it is reclaimed after I/O processing completes. Buffered I/O is useful for small transfers, such as mouse and keyboard input, because the system does not need to lock physical memory pages due to the fact that the system transfers the data first between the device and nonpaged pool. When the VMM locks a memory page, the page cannot be paged out of memory. However, for large transfers (i.e., more than one memory page), the overhead of first copying the data into the system buffer and then making a second copy to the process buffer reduces I/O performance. Also, because system memory becomes fragmented as the system executes, the buffer for a large transfer likely will not be contiguous, further degrading performance.315 As an alternative, drivers can employ direct I/O, with which the device transfers data directly to the process's buffer. Before passing the IRP to the first driver, the I/O manager creates a memory descriptor list (MDL), which maps the applica- 21.9 Input/Output Management 1089 ble range of the requesting process's virtual addresses to physical memory addresses. The VMM then locks the physical pages listed in the MDL and constructs the IRP with a pointer to the MDL. Once the device completes transferring data (using the MDL to access the process's buffer), the I/O manager unlocks the memory pages.316 Drivers can choose not to use buffered I/O or direct I/O and, instead, use a technique called neither I/O. In this technique, the I/O manager passes IRPs that describe the data's destination (for a read) or origin (for a write) using the virtual addresses of the calling thread. Because a driver performing neither I/O needs access to the calling thread's virtual address space, the driver must execute in the calling thread's context. For each IRP, the driver can decide to set up a buffered I/O transfer by creating the two buffers and passing a buffered I/O IRP to the next lowest driver (recall that the I/O manager handles this when the buffered I/O flag is set). The driver can also choose to create an MDL and set up a direct I/O IRP. Alternatively, the driver can perform all the necessary transfer operations in the context of the calling thread. In any of these cases, the driver must handle all exceptions that might occur and ensure that the calling thread has sufficient access rights. The I/O manager handles these issues when direct I/O or buffered I/O is used.317 Because a driver employing neither I/O must execute in the context of the calling thread, only high-level drivers can use neither I/O. High-level drivers can guarantee that they will be entered in the context of the calling thread; on the other hand, an intermediate or low-level driver is called by another driver and cannot assume that the other driver will execute in the context of the calling thread.318 All drivers in a device stack must use the same transfer technique for that device—otherwise the transfer would fail due to device drivers' executing conflicting operations—except that the highest-level driver can use neither I/O instead.319 21.9.3 Interrupt Handling Once a device completes processing an I/O request, it notifies the system with an interrupt. The interrupt handler calls the interrupt service routine (ISR) associated with that device's interrupt. 320 An ISR is a standard driver routine used to process interrupts. It returns false if the device with which it is associated is not interrupting; this can occur because more than one device can interrupt at the same DIRQL. If the associated device is interrupting, the ISR processes the interrupt and returns true. 321 When a device that generates interrupts is installed on a Windows XP system, a driver for that device must register an ISR with the I/O manager. Typically, low-level drivers are responsible for providing ISRs for the devices they serve. When a driver registers an ISR, the system creates an interrupt object.322 An interrupt object stores interrupt-related information, such as the specific DIRQL at which the interrupt executes, the interrupt vector of the interrupt and the address of the ISR.323 The I/O manager uses the interrupt object to associate an interrupt vector in the kernel's Interrupt Dispatch Table (IDT) with the ISR. The IDT maps hardware interrupts to interrupt vectors.324 1090 Case Study: Windows XP When an interrupt occurs, the kernel maps a hardware-specific interrupt request into an entry in its IDT to access the correct interrupt vector.325 Often, there are more devices than entries in the IDT, so the processor must determine which ISR to use. It does this by executing, in succession, each ISR associated with the applicable vector. When an ISR returns false, the processor calls the next ISR.326 An ISR for a Windows XP system should execute as quickly as possible, then queue a DPC to finish processing the interrupt. While a processor is executing an ISR, it executes at the device's DIRQL. This masks interrupts from devices with lower DIRQLs. Quickly returning from an ISR, therefore, increases a system's responsiveness to all device interrupts. Also, executing at a DIRQL restricts the support routines that can be called (because some of these routines must run at some IRQL below DIRQL). Additionally, executing an ISR can prevent other processors from executing the ISR—and some other pieces of the driver's code—because the kernel interrupt handler holds the driver's spin lock while processing the ISR. Therefore, an ISR should determine if the device with which it is associated is interrupting. If it is not interrupting, the ISR should return false immediately; otherwise, it should clear the interrupt, gather the requisite information for interrupt processing, queue a DPC with this information and return true. The driver can use the device extension to store the information needed for the DPC. Typically, the DPC will be processed the next time the IRQL drops below DPC/dispatch level.327 21.9.4 File Cache Management Not all I/O operations require the process outlined in the previous sections. When requested data resides in the system's file cache, Windows XP can fulfill an I/O request without a costly device access and without executing device driver code. The cache manager is the executive component that manages Windows XP's file cache, which consists of main memory that caches file data.328 Windows XP does not implement separate caches for each mounted file system, but instead maintains a single systemwide cache.329 Windows XP does not reserve a specific portion of RAM to act as the file cache, but instead allows the file cache to grow or shrink dynamically, depending on system needs. If the system is running many I/O-intensive routines, the file cache grows to facilitate fast data transfer. If the system is executing large programs that consume lots of system memory, the file cache shrinks to reduce page faults. Furthermore, the cache manager never knows how much cached data currently resides in physical memory. This is because the cache manager caches by mapping files into the system's virtual address space, rather than managing the cache from physical memory. The memory manager is responsible for paging the data in these views into or out of physical memory. This caching method allows easy integration of Windows XP's two data access methods: memory-mapped files (see Section 13.9, Data Access Techniques) and traditional read/write access. The same file can be accessed via both methods (i.e., as a memory-mapped file through the memory manager and as a traditional file through the file system), because the cache manager maps file data into 21.10 Interprocess Communication 1091 virtual memory. The cache manager bridges the gap between these two file representations, ensuring consistency in file data between the two views.330 When a file is read from a disk, its contents are stored in the system cache, which is a portion of the system address space. Then, a user-mode process copies the data from the cache into its own address space. A thread can set a flag that overrides this default behavior; in this case, the transfer occurs directly between the disk and the process's address space. Similarly, instead of writing directly to the disk, a process writes new data to the cache entry (unless a flag is set).331 Fulfilling an I/O request without generating an IRP or accessing a device is called fast I/O.332 Dirty cache entries can be written to disk in several ways. The memory manager might need to page a dirty cache page out of memory to make room for another memory page. When this occurs, the contents of the cache page are queued to be written to disk. If a dirty cache page is not paged out of memory by the memory manager, the cache manager must flush the page back to disk. It accomplishes this by using a lazy writer thread. The lazy writer is responsible for, once per second, choosing one-eighth of the dirty pages to be flushed to disk. It constantly reevaluates how often dirty pages are being created and adjusts the amount of pages it queues to be flushed accordingly. The lazy writer chooses pages based on how long a page has been dirty and the last time the data on that page was accessed for a read operation. Additionally, threads can force a specific file to be flushed. Threads can also specify write-through caching (see Section 12.8, Caching and Buffering) by setting a flag when creating the file object.333 21.10 Interprocess Communication Windows XP provides many interprocess communication (IPC) mechanisms to allow processes to exchange data and cooperate to complete tasks. It implements many traditional UNIX IPC mechanisms, such as pipes, message queues (called mailslots by Windows XP)334 and shared memory. In addition to these "data-oriented" IPCs, Windows XP allows processes to communicate through "procedureoriented" or "object-oriented" techniques, using such tools as remote procedure calls or Microsoft's Component Object Model. Users on a Windows XP system also can initiate IPC with familiar features such as the clipboard and drag-and-drop capabilities. In each of Windows XP's IPC mechanisms, a server process makes some communication object available. A client process can contact the server process via this communication object to place a request. The request might be for the server to execute a function or to return data or objects. The remainder of this section describes these IPC tools. Section 21.11, Networking, introduces techniques and tools, such as sockets, that processes can use to communicate across a network. 21.10.1 Pipes Windows XP provides pipes for direct communication between two processes.335 If communicating processes have access to the same memory bank, pipes use shared 1092 Case Study: Windows XP memory. Thread can manipulate pipes using standard file system routines (e.g., read, write and open). A process that creates the pipe is called the pipe server; processes that connect to a pipe are referred to as pipe clients.336 Each pipe client communicates solely with the pipe server (i.e., not with other clients using the same pipe) because the pipe server uses a unique instance of the pipe for each client process.337 The pipe server specifies a mode (read, write or duplex) when the pipe is created. In read mode, the pipe server receives data from clients; in write mode, it sends data to pipe clients; in duplex mode, it can both send and receive data via the pipe.338 Windows XP provides two types of pipes: anonymous pipes and named pipes. Anonymous Pipes Anonymous pipes are used for unidirectional communication (i.e., duplex mode is not allowed) and can be used only among local processes. Local processes are processes that can communicate with each other without sending messages over a network.339 A process that creates an anonymous pipe receives both a read handle and a write handle for the pipe. A process can read from the pipe by passing the read handle as an argument in a read function call, and a process can write to the pipe by passing the write handle as an argument in a write function call (for more information on handles see Section 21.5.2, Object Manager). To communicate with another process, the pipe server must pass one of these handles to another process. Typically, this is accomplished through inheritance (i.e., the parent process allows the child process to inherit one of the pipe handles). Alternatively, the pipe server can send a handle for the pipe to another, unrelated process via some IPC mechanism.340 Anonymous pipes support only synchronous communication. When a thread attempts to read from the pipe, it waits until the requested amount of data has been read from the pipe. If there is no data in the pipe, the thread waits until the writing process places the data in the pipe. Similarly, if a thread attempts to write to a pipe containing a full buffer (the buffer size is specified at creation time), it waits until the reading process removes enough data from the pipe to make room for the new data. A thread that is waiting to read data via a pipe can stop waiting if all write handles to the pipe are closed or if some error occurs. Similarly, a thread waiting to write data to a pipe can stop waiting if all read handles to the pipe are closed or if some error occurs.341 Named Pipes Named pipes support several features absent in anonymous pipes. Named pipes can be bidirectional, be shared between remote processes342 and support asynchronous communication. However, named pipes introduce additional overhead. When the pipe server creates a pipe, the pipe server specifies its mode, its name and the maximum number of instances of the pipe. A pipe client can obtain a handle for the pipe by specifying the pipe's name when attempting to connect to the pipe. If the number of pipe instances is less than the maximum, the client process connects to a new instance of the pipe. If there are no available instances, the pipe client can wait for an instance to become available.343 21.10 Interprocess Communication 1093 Named pipes permit two pipe writing formats. The system can transmit data either as a stream of bytes or as a series of messages. All instances of a pipe must use the same write format (recall that a single pipe server can communicate with multiple clients by creating a unique instance of the pipe for each client). Although sending data as a series of bytes is faster, messages are simpler for the receiver to process because they package related data. 344 Windows XP permits a process to enable write-through mode. When the writethrough flag is set, write operations do not complete until the data reaches the receiving process's buffer. In the default mode, once the data enters the writer's buffer, the write method returns. Write-through mode increases fault tolerance and permits greater synchronization between communicating processes because the sender knows whether a message successfully reaches its destination. However, write-through mode degrades performance, because the writing process must wait for the data to be transferred across a network from the writer's end of the pipe to the reader's end.345 The system always performs writes using the write-through method when a process is using messages to ensure that each message remains a discrete unit.346 Asynchronous I/O capability increases named pipes' flexibility. Threads can accomplish asynchronous communication via named pipes in several ways. A thread can use an event object to perform an asynchronous read or write request. In this case, when the function completes, the event object enters its signaled state (see Section 21.6.3, Thread Synchronization). The thread can continue processing, then wait for one or more of these event objects to enter signaled states to complete a communication request. Alternatively, a thread can specify an I/O completion routine. The system queues this routine as an APC when the function completes, and the thread executes the routine the next time the thread enters an alertable wait state. 347 In addition, a pipe can be associated with an I/O completion port because pipes are manipulated like files. 21.10.2 Mailslots Windows XP provides mailslots for unidirectional communication between a server and clients.348 The process that creates the mailslot is the mailslot server, whereas the processes that send messages to the mailslot are mailslot clients. Mailslots act as repositories for messages sent by mailslot clients to a mailslot server. Mailslot clients can send messages to either local or remote mailslots; in either case, there is no confirmation of receipt. Windows XP implements mailslots as files in the sense that a mailslot resides in memory and can be manipulated with standard file functions (e.g., read, write and open). Mailslots are temporary files (the Windows XP documentation refers to them as "pseudofiles"); the object manager deletes a mailslot when no process holds a handle to it.349 Mailslot messages can be transmitted in two forms. Small messages are sent as datagrams, which are discussed in Section 16.6.2, User Datagram Protocol (UDP). Datagram messages can be broadcast to several mailslots within a particular domain. A domain is a set of computers that share common resources such as print- 1094 Case Study: Windows XP ers.350, 351 Larger messages are sent via a Server Message Block (SMB) connection.352 SMB is a network file sharing protocol used in Windows operating systems.353 Both SMB and datagram messages can be sent from a mailslot client to a mailslot server by specifying the server name in a write operation.354 21.10.3 Shared Memory Processes can communicate by mapping the same file to their respective address spaces, a technique known as shared memory. The Windows XP documentation calls this mechanism file mapping. A process can create a file mapping object that maps any file, including a pagefile, to memory The process passes a handle for the file mapping object to other processes, either by name or through inheritance. The process communicates by writing data into these shared memory regions, which other processes can access. A process can use the handle to create a file view that maps all or part of the file to the process's virtual address space. A process may own multiple file views of the same file. This allows a process to access the beginning and end of a large file without mapping all of the file to memory, which would reduce its available virtual address space. Because identical file views map to the same memory frames, all processes see a consistent view of a memory-mapped file. If one process writes to its file view, the change will be reflected in all file views of that portion of the file.355, 356 File mapping objects accelerate data access. The system maps file views independently to each process's virtual address space, but the data is located in the same frames in main memory. This minimizes the necessity of reading data from the disk. The VMM treats file mappings the same way it treats copy-on-write pages (see Section 21.7.1, Memory Organization). Because the VMM has been optimized to deal with paging, file access is efficient. Due to its efficient implementation, file mapping has uses beyond communication. Processes can use file mapping to perform random and sequential I/O quickly. File mapping is also helpful for databases that need to overwrite small sections of large files.357, 358 File mapping objects do not provide synchronization mechanisms to protect the files they map. Two processes can overwrite the same portion of the same file simultaneously. To prevent race conditions, processes must use synchronization mechanisms, such as mutexes and semaphores, that are provided by Windows XP and described in Section 21.6.3, Thread Synchronization.359 21.10.4 Local and Remote Procedure Calls Many IPC mechanisms facilitate data exchange between two processes. Local procedure calls (LPCs) and remote procedure calls (RPCs), which we described in Section 17.3.2, provide a more procedural form of IPC and largely hide the underlying network programming. LPCs and RPCs allow a process to communicate with another process by calling functions executed by the other process.360 The process that calls the function is the client process, and the one that executes the function is the server process. 21. 10 Interprocess Communication 1095 When client and server processes reside on the same physical machine, processes execute LPCs; when they reside on separate machines, processes execute RPCs.361A designer employs LPCs and RPCs the same as procedure calls within a process. However, in some cases, the programmer takes a few extra steps to establish a connection (these are described shortly). Note that Microsoft does not publicly document the LPC interface, and usermode threads cannot expose LPCs. The system reserves LPCs for use by the Windows XP kernel. For example, when a user thread calls a function in an environment subsystem's API, the kernel might convert the call into an LPC. Also, Windows XP converts RPCs between two processes on the same machine (called local remote procedure calls or LRPCs) into LPCs, because LPCs are more efficient.362 LPCs have less overhead than RPCs, because all data transfers using LPCs involve direct memory copying rather than network transport.363 Although the name "local remote procedure call" might sound like an oxymoron, it is simply a descriptive name for a procedure call made with the RPC interface between local processes. Figure 21.16 describes the path through which data travels during an LPC or RPC call. In both an RPC and an LPC, the client process calls a procedure that maps to a stub (1). This stub marshals the necessary function arguments (i.e., gathers the function arguments into a message to send to the server) and converts them to Network Data Representation (NDR), which is a standard network data format, described in The Open Group's Distributed Computing Environment (DCE) standard (2).364, 365 Next, the stub calls the appropriate functions from a runtime library to send the request to the server process. The request traverses the network (in the case of an RPC) to reach the server-side runtime library (3), which passes the request to Figure 21.16 | Flow of an LPC or RPC call. 1096 Case Study: Windows XP the server stub (4). The server stub unmarshals the data (i.e., converts the client's message into a call to the intended procedure) for the server function (5). The server process executes the requested function and sends any return values to the client via the reverse path (6 through 10).366 Developers can employ RPCs using many different transport and network protocols such as TCP/IP and Novell Netware's Internetworking Packet eXchange/Sequenced Packet eXchange (IPX/SPX).367 Both client processes and server processes can execute RPCs either synchronously or asynchronously, and the communicating processes need not use the same method.368 Cliend and Server Communication in an RPC To expose a procedure as an RPC, the server process must create an Interface Definition Language (IDL) file. This file specifies the interfaces that the RPC server presents to other processes.369 The developer writes the interface in Microsoft IDL (MIDL), which is Microsoft's extension of IDL—The Open Group's Distributed Computing Environment (DCE) standard for RPC interoperability. 370 The IDL file consists of a header and an interface body. The header describes information global to all interfaces defined in the body such as the universally unique identifier (UUID) for the interface and an RPC version number. 371 The body contains all variable and function prototype declarations. The MIDL compiler builds the client and server stubs from the IDL file.372 The server and client also can create application configuration files (ACFs), which specify platform-specific attributes, such as the manner in which data should be marshalled or unmarshalled. 373 The physical communication in an RPC is accomplished through an endpoint that specifies the network-specific address of the server process and is typically a hardware port or named pipe. The server creates an endpoint before exposing the RPC to other processes.374 Client-side runtime library functions are responsible for establishing a binding to this endpoint, so that a client can send a request to the server. A binding is a connection between the client and the server. To create a binding, the client process must obtain a binding handle, which is a data structure that stores the connection information.375 The binding handle stores such information as the name of the server, address of the endpoint and the protocol sequence.376 The protocol sequence specifies the RPC protocol (e.g., connection-oriented, LRPC), the transport protocol (e.g., TCP) and the network protocol (e.g., IP). 377 Windows XP supports three types of binding handles: automatic, implicit and explicit handles. Using an automatic handle, the client process calls the remote function, and the stub manages all the communication tasks. Implicit handles permit the client process to specify the particular server to use with an RPC, but once the client process passes the handle to the runtime library functions, it no longer needs to manage the handle. With explicit handles, the client process must specify the binding information as well as create and manage the handle. 378 This extra control permits client processes using explicit handles to connect to more than one server process and simultaneously execute multiple RPCs. 379 Communication via an LPC is similar to an RPC, although some steps are omitted. The server must still create an IDL file, which is compiled by a MIDL 21.10 Interprocess Communication 1097 compiler. The client process must include the client-side runtime library with the file that calls the LPC. The stub handles the communication, and the client and server processes communicate over a port for procedure calls that involve small data transfers. Procedure calls in which processes transfer a large amount of data must use shared memory. In this case, the sender and receiver place message data into a section of shared memory.380 21.10.5 Component Object Model (COM) Microsoft's Component Object Model (COM) provides a software architecture allowing interoperability between diverse software components. In the COM architecture, the relative location of two communicating components is transparent to the programmer and can be in process, cross process or cross network. COM is not a programming language, but a standard designed to promote interoperability between components written in different programming languages. COM also is implemented on some flavors of UNIX and Apple Macintosh operating systems. Developers use COM to facilitate cooperation and communication between separate components in large applications.381 COM objects (a COM object is the same as a COM component) interact indirectly and exclusively through interfaces.382 A COM interface, which is written in MIDL, is similar to a Java interface; it contains function prototypes that describe function arguments and return values, but does not include the actual implementation of the functions. As with Java, a COM object must implement all of the methods described in the interface and can implement other methods as well.383 Once created, a COM interface is immutable (i.e., it cannot change). A COM object can have more than one interface; developers augment COM objects by adding a new interface to the object. This permits clients dependent on the old features of a COM object to continue functioning smoothly when the object is upgraded. Each interface and object class possesses a globally unique ID (GUID), which is a 128-bit integer that is, for all practical purposes, guaranteed to be unique in the world. Interface IDs (IIDs) are GUIDs for interfaces, and class IDs (CLSIDs) are GUIDs for object classes. Clients refer to interfaces using the IID, and because the interface is immutable and the IID is globally unique, the client is guaranteed to access the same interface each time.384 COM promotes interoperability between components written in diverse languages by specifying a binary standard (i.e., a standard representation of the object after it has been translated into machine code) for calling functions. Specifically, COM defines a standard format for storing pointers to functions and a standard method for accessing the functions using these pointers. 385 Windows XP, and other operating systems with COM support, provide APIs for allocating and deallocating memory using this standard. The COM library in Windows XP provides a rich variety of support functions that hide most of the COM implementations. 386 COM servers can register their objects with the Windows XP registry. Clients can query the registry using the COM object's CLSID to obtain a pointer to the 1098 Case Study: Windows XP COM object.387 A client can also obtain a pointer to a COM object's interface from another COM object or by creating the object. A client can use a pointer to any interface for a COM object to find a pointer to any other interface that the COM object exposes.388 Once the client obtains a pointer to the desired interface, the client can execute a function call to any procedure exposed by this interface (assuming the client has appropriate access rights). For in-process procedure calls, COM executes the function directly, adding essentially no overhead. For cross-process procedure calls in which both processes reside on the same computer, COM uses LRPCs (see Section 21.10.4, Local and Remote Procedure Calls), and Distributed COM (DCOM) supports cross-network function calls.389 COM marshals and unmarshals all data, creates the stubs and proxies (proxies are client-side stubs) and transports the data (via direct execution, LRPCs or DCOM).390 COM supports several threading models a process can employ to maintain its COM objects. In the apartment model, only one thread acts as a server for each COM object. In the free thread model, many threads can act as the server for a single COM object (each thread operates on one or more instances of the object). A process can also use the mixed model in which some of its COM objects reside in single apartments and others can be accessed by free threads. In an apartment model, COM provides synchronization by placing function calls in the thread's window message queue. COM objects that can be accessed by free threads must maintain their own synchronization (see Section 21.6.3, Thread Synchronization, for information on thread synchronization).391 Microsoft has built several technologies on top of COM to further facilitate cooperation in component software design. COM+ extends COM to handle advanced resource management tasks; e.g., it provides support for transaction processing and uses thread pools and object pools, which move some of the responsibility for resource management from the component developer to the system.392 COM+ also adds support for Web services and optimizes COM scalability.393 Distributed COM (DCOM) provides a transparent extension to basic COM services for cross-network interactions and includes protocols for finding DCOM objects on remote services.394 Object Linking and Embedding (OLE), discussed in the next section, builds on COM to provide standard interfaces for applications to share data and objects. ActiveX Controls are self-registering COM objects (i.e., they insert entries in the registry upon creation). Typically, ActiveX Controls support many of the same embedding interfaces as OLE objects; however, ActiveX Controls do not need to support all of them. This makes ActiveX Controls ideal for embedding in Web pages because they have less overhead.395 21.10.6 Drag-and-Drop and Compound Documents Windows XP provides several techniques that enable users to initiate IPC. A familiar example is permitting users to select text from a Web page, copy this text and paste it into a text editor. Users can also embed one document inside another; e.g., a 21.11 Networking 1099 user could place a picture created in a graphical design program into a word processing document. In both of these cases, the two applications represent different processes that exchange information. Windows XP provides two primary techniques for this type of data transport: the clipboard and Object Linking and Embedding (OLE). The clipboard is a central repository of data, accessible to all processes.396 A process can add data to the clipboard when the user invokes either the copy or the cut command. The selected data is stored in the clipboard along with the data's format.397 Windows XP defines several standard data formats, including text, bitmap and wave. Processes can register new clipboard formats, create private clipboard formats and synthesize one or more of the existing formats.398 Any process can retrieve data from the clipboard when the user invokes the paste command.399 Conceptually, the clipboard acts as a small area of globally shared memory. OLE builds on COM technology by defining a standard method for processes to exchange data. All OLE objects implement several standard interfaces that describe how to store and retrieve data for an OLE object, how to access the object and how to manipulate the object. Users can create compound documents—documents with objects from more than one application—either by linking outside objects or embedding them. When a document links an object from another source, it does not maintain the object inside its document container (the document container provides storage for the document's objects and methods for manipulating and viewing these objects). Rather, the document maintains a reference to the original object. The linked object reflects updates made by the server process to the object. Embedding an object places it, along with the interfaces for manipulating the object, inside the client document. The linked or embedded object is a COM component complete with a set of interfaces that the client can use to manipulate the object.400 21.11 Networking Windows XP supports many networking protocols and services. Developers employ networking services to accomplish IPC with remote clients and make services and information available to users. Users exploit networks to retrieve information and access resources available on remote computers. This section investigates various aspects of Windows XP's networking model. We examine network I/O and the driver architecture employed by Windows XP. We consider the network, transport and application protocols supported by Windows XP. Finally, we describe the network services that Windows XP provides, such as the Active Directory and .NET. 21.11.1 Network Input/Output Windows XP provides a transparent I/O programming interface. In particular, programmers use the same functions regardless of where the data reside. However, the Windows XP system must handle network I/O differently than local I/O.401 1100 Case Study: Windows XP Figure 21.17 illustrates how Windows XP handles a network I/O request. The client contains a driver called a

redirector (or network redirector).402 A redirector is a file system driver that directs network I/O requests to appropriate devices over a network.403 First, the client process passes an I/O request to an environment subsystem (1). The environment subsystem then sends this request to the I/O manager (2), which packages the request as an IRP, specifying the file's location in Uniform Naming Convention (UNC) format. UNC format specifies the file's pathname, including on which server and in which directory on that server the file is located. The I/O manager sends the IRP to the Multiple UNC Provider (MUP), which is a file system driver that determines the appropriate redirector to which to send the request (3). Later in this section we see that Windows XP ships with two file sharing protocol, CIFS and WebDAV, which use different redirectors. After the redirector receives the IRP (4), the redirector sends the request over the network to the appropriate server file system driver (5). The server driver determines whether the client has sufficient access rights for the file, then communicates the request to the target device's driver stack via an IRP (6). The server driver receives the result of the I/O request (7). If there is an error processing the request, the server driver informs the client. Otherwise, the server driver passes the data back to the redirector (8), and this data is passed to the client process (9).404,405 The Common Internet File System (CIFS) is Windows XP's native file sharing protocol. CIFS is used for the application layer of the TCP/IP stack (see Section 16.4, Figure 21.17 | Handling & network I/O request. 21.11 Networking 1101 TCP/IP Protocol Stack). VMS and several varieties of UNIX also support CIFS. Developers often pair CIFS with NetBIOS (discussed later in this section) over TCP/ IP for the network and transport layers, but CIFS can use other network and transport protocols as well.406 CIFS is meant to complement HTTP and replace older file sharing protocols such as FTP.407 Windows XP also supports other network file sharing protocols such as Web-based Distributed Authoring and Versioning (WebDAV). WebDAV allows users to write data directly to HTTP servers and is designed to support collaborative authoring between groups in remote locations.408 To share files using CIFS, the client and server must first establish a connection. Before a client can access a file, the client and server connect, and the server authenticates the client (see Section 19.3, Authentication) by evaluating the username and password sent by the client. The redirectors interact to set up the session and transfer data.409 CIFS provides several types of opportunistic locks (oplocks) to enhance I/O performance. A client uses an oplock to secure exclusive access to a remote file. Without one of these oplocks, a client cannot cache network data locally, because other clients also might be accessing the data. If two clients both cache data locally and write to this data, their caches will not be coherent (see Section 15.5, Multiprocessor Memory Sharing). Therefore, a client obtains an oplock to ensure that it is the only client accessing the data.410 When a second client attempts to gain access to the locked file (by attempting to open it), the server may break (i.e., invalidate) the first client's oplock. The server allows the client enough time to flush its cache before granting access to the second client.411 21.11.2 Network Driver Architecture Windows XP uses a driver stack to communicate network requests and transmit network data. This driver stack is divided into several layers, which promotes modularity and facilitates support for multiple network protocols. This subsection describes the low-level driver architecture employed in Windows XP. Microsoft and 3Com developed the Network Driver Interface Specification (NDIS), which specifies a standard interface between lower-level drivers in the network driver stack. NDIS drivers provide the functionality of the link layer and some functionality of the network layer of the TCP/IP protocol stack (see Section 16.4, TCP/IP Protocol Stack). 412 NDIS drivers communicate through functions provided in Windows XP's NDIS library. The NDIS library functions translate one driver's function call into a call to a function exposed by another driver. This standard interface increases the portability of NDIS drivers.413 NDIS allows the same programming interface (e.g., Windows sockets, see Section 21.11.3, Network Protocols) to establish connections using different network protocols, such as IP and Internet Packet eXchange (IPX). 414 Figure 21.18 illustrates the architecture that Windows XP employs to process network data. Windows XP divides the network architecture into three layers that map to layers of the TCP/IP protocol stack. The physical network hardware, such as the network interface card (NIC) and the network cables, form the physical layer. 1102 Case Study: Windows XP Figure 21.18 | Networking driver architecture. NDIS drivers (described in the following paragraphs) manage the data link and network layers. Transport Driver Interface (TDI) drivers and TDI clients compose the top layer. The TDI drivers provide a transport interface between NDIS drivers and TDI clients. TDI clients are low-level application drivers and might interact with other application-layer drivers. Designers implement the NDIS layer with several drivers. An NDIS miniport driver manages the NIC and transmits data between the NIC and higher-level drivers. The miniport driver interfaces with the drivers above it in the driver stack to transmit outgoing network data to the NIC. An upper-level driver invokes an NDIS function that passes the packet to the miniport driver, and the miniport driver passes the data to the NIC. The miniport driver also processes the NIC's interrupts and passes data that the NIC receives up the driver stack.415 21.11 Networking 1103 NDIS intermediate drivers are optional drivers that reside between a miniport driver and a higher-level driver. Intermediate drivers translate packets between different communication media, filter packets or provide load balancing across several NICs.416 NDIS protocol drivers are the highest-level NDIS drivers. Protocol drivers place data into packets and pass these packets to lower-level drivers (such as intermediate drivers or miniport drivers). Protocol drivers provide an interface between the transport drivers and other NDIS drivers and can be used as the lowest layer in the implementation of a transport protocol stack (such as TCP/IP). 417 The NDIS protocol driver interacts with TDI drivers. A TDI driver exports a network interface to upper-level kernel-mode clients. The redirectors described in the previous subsection are TDI clients. Winsock applications (described in the next subsection) also are TDI clients. TDI drivers implement transport protocols such as UDP and TCP; NDIS protocol drivers often implement the network protocol (e.g., IP). Alternatively, a TDI transport driver can implement both the network and transport protocols. Upper-level kernel-mode drivers send network requests to TDI drivers.418 These drivers implement the application layer of the TCP/IP protocol stack. The next subsections describe the various protocols and network services available to Windows XP applications. 21.11.3 Network Protocols Windows XP supports several different protocols for the network, transport and application layers of the TCP/IP stack. Windows XP supports some of these protocols to provide compatibility with popular legacy programs or other network clients. Network and Transport Protocols IP is installed by default on Windows XP systems; IP provides routing for packets traversing a network and is the Internet's network protocol. Users can install other network protocols, such as the Internetwork Packet exchange (IPX) protocol, which is used by Novell's Netware. 419 IPX provides services similar to IP; IPX routes packets between different locations in a network, but is designed for LANs. 420 Because the NDIS library supports a standard interface between protocol drivers and NICs, Windows XP can use the same NIC to operate both protocols. 421 Windows XP also supports several transport protocols. TCP, by far the most commonly used transport protocol, is installed by default. TCP provides a connection-oriented transport; i.e., the participants of a network communication must create a session before communicating.422 For connectionless transport, Windows XP supports UDP. 423 Users can also install support for Sequenced Packet eXchange (SPX), which offers connection-oriented services for IPX packets. Windows XP provides SPX for interoperability with clients and servers using IPX/SPX protocols, such as Netware, 424 Before the explosion of interest in the Internet and World Wide Web, the DOS and Windows operating systems supported NetBIOS Extended User Inter- 1104 Case Study: Windows XP face (NetBEUI) as the native network and transport protocols. Network Basic Input/Output System (NetBIOS) is the programming API that supports NetBEUI. NetBEUI is a fairly primitive protocol most applicable for small networks. It has several limitations; for example, it does not provide for network connection routing outside a LAN. 425 Windows XP does not provide support for NetBEUI, although support can be manually installed by a user. However, NetBIOS over TCP/IP (NBT) is still supported by Windows XP to provide compatibility with old applications that employ the NetBIOS API. 426 NetBIOS layered over TCP and IP provides connection-oriented communication using NBT. Alternatively, developers can layer NetBIOS over UDP and IP for connectionless communication using NBT. CIFS is commonly used with NBT as the underlying network and transport protocols. However, in Windows XP, CIFS can execute over TCP/IP without NetBIOS encapsulation. 427 Application Layer Protocols At the application layer, Windows XP supports HTTP for Web browsing and CIFS for file and print sharing. For HTTP support, Windows XP provides WinHTTP and WinlNet. WinHTTP supports communication between clients and servers over an HTTP session. It provides SSL security and is integrated with Microsoft Passport (see Section 19.3.4, Single Sign-On). WinHTTP also provides other HTTP services, such as tracing and Kerberos authentication (see Section 19.3.3, Kerberos) and allows synchronous and asynchronous communication.428 WinlNet is an older API that allows applications to interact with the FTP, HTTP and Gopher protocols to access resources over the Internet.429 WinlNET is designed for client applications, whereas WinHTTP is ideal for servers.430 CIFS, described in the previous section, is an extension of the Server Message Block (SMB) file sharing protocol. SMB is the native file sharing protocol of older Windows operating systems such as Windows NT and Windows 95. SMB uses NetBIOS to communicate with other computers in a virtual LAN. A virtual LAN consists of computers that share the same NetBIOS namespace; the computers can be anywhere in the world. To share files with SMB, two computers on the same virtual LAN set up an SMB session; this includes acknowledging the connection, authenticating the identity of both parties and determining the access rights of the client. CIFS extends SMB, adding a suite of protocols that improve access control and handle announcing and naming of network resources. The familiar My Network Places program included with Windows XP allows users to share files through CIFS.431 Winsock Windows XP supports network communication through sockets. Windows sockets 2 (Winsock 2) is an application of BSD sockets (see Section 20.10.3, Sockets) to the Windows environment. Sockets create communication endpoints that a process can use to send and receive data over a network.432 Sockets are not a transport protocol; consequently, they need not be used at both ends of a communication connection. Developers can layer Winsock 2 on top of any transport and network protocols (e.g. 21.11 TCP/IP or IPX/SPX). Users can port code written for UNIX-style sockets to Winsock applications with minimal changes. However, Winsock 2 includes extended functionality that might make porting an application from Windows XP to a UNIXbased operating system implementing Berkeley sockets difficult.433, 434 The Winsock specification implements many of the same functions that Berkeley sockets implement to provide compatibility with Berkeley socket applications. As with Berkeley sockets, a Windows socket can be either a stream socket or a datagram socket. Recall that stream sockets provide reliability and packet ordering guarantees.435 Winsock adds extra functionality to sockets, such as asynchronous I/O, protocol transparency (i.e., developers can choose the network and transport protocols to use with each socket) and quality-of-service (QoS) capabilities.436 The latter provide to processes information about network conditions and the success or failure of socket communication attempts.437 21.11.4 Network Services Windows XP includes various network services that allow users to share information, resources and objects across a network. Active Directory provides directory services for shared objects (e.g., files, printers, services, users, etc.) on a network.438 Its features include location transparency (i.e., users do not know the address of an object), hierarchical storage of object data, rich and flexible security infrastructure and the ability to locate objects based on different object properties.439 Lightweight Directory Access Protocol (LDAP) is a protocol for accessing, searching and modifying Internet directories. An Internet directory contains a listing of computers or resources accessible over the Internet. LDAP employs a client/ server model.440 LDAP clients can access an Active Directory.441 Windows XP provides Remote Access Service (RAS), which allows users to remotely connect to a LAN. The user must be connected to the network through a WAN or a VPN. Once the user connects to the LAN through RAS, the user operates as though directly on the LAN.442 Support for distributed computing on Windows XP is provided through the Distributed Component Object Model (DCOM) and .NET. DCOM is an extension of COM, which we described in Section 21.10.5, Component Object Model (COM), for processes that share objects across a network. Constructing and accessing DCOM objects is done just as with COM objects, but the underlying communication protocols support network transport. For information on using DCOM for distributed computing see Section 17.3.5, DCOM (Distributed Component Object Model).443 The next section introduces .NET. 21.11.5 .NET Microsoft has developed a technology called .NET, which supplants DCOM for distributed computing in newer applications. .NET is aimed at transforming computing from an environment where users simply execute applications on a single computer to a fully distributed environment. In this computing model, local applications com- Nefworking 1105 1106 Case Study: Windows XP municate with remote applications to execute user requests. This contrasts with the traditional view of a computer as merely fulfilling a single user's requests. .NET is a middleware technology, which provides a platform- and language-neutral development environment for components that can easily interperate.444, 445 Web services are the foundation of .NET; they encompass a set of XML-based standards that define how data should be transported between applications over HTTP. Because of XML's large presence in this technology, .NET Web services are often called XML Web services. Developers use Web services to build applications that can be accessed over the Internet. Web service components can be exposed over the Internet and used in many different applications (i.e., Web services are modular). Because XML messages (which are sent and retrieved by Web services) are simply text messages, any application can retrieve and read XML messages. Web services make .NET interoperable and portable between different platforms and languages.446 The .NET programming model is called the .NET framework. This framework provides an API which expands on the Windows API to include support for Web services. The .NET framework supports many languages, including Visual Basic .NET, Visual C++ .NET and C#. The .NET framework facilitates development of .NET Web services.447 .NET server components include Web sites and .NET enterprise servers, such as Windows Server 2003. Many Web service components that home users access are exposed on these two platforms. Although desktop operating systems, such as Windows XP, or server systems running Windows XP can expose Web services, Windows XP systems are mainly consumers (i.e., clients) of .NET services.448-449 21.12 Scalability To increase Windows XP's flexibility, Microsoft developed several editions. For example, Windows XP Home Edition is tailored for home users, whereas Windows XP Professional is more applicable in a corporate environment. Microsoft has also produced editions of Windows XP to address scalability concerns. Although all editions of Windows XP support symmetric multiprocessing (SMP), Windows XP 64Bit Edition is specifically designed for high-performance desktop systems, which are often constructed as SMP systems (see Section 15.4.1, Uniform Memory Access). Additionally, Microsoft has developed an embedded operating system derived from Windows XP, called Windows XP Embedded. An embedded system consists of tightly coupled hardware and software designed for a specific electronic device, such as a cellular phone or an assembly-line machine. This section describes the features included in Windows XP to facilitate scaling up to SMP systems and down to embedded systems. 21.12.1 Symmetric Multiprocessing (SMP) Windows XP scales to symmetric multiprocessor (SMP) systems. Windows XP supports multiprocessor thread scheduling, locking the kernel, a 64-bit address space 21.12 Scalability 1107 (in Windows XP 64-Bit Edition) and API features that support programming server applications. Windows XP can schedule threads efficiently on multiprocessor systems, as described in Section 21.6.2, Thread Scheduling. Windows XP attempts to schedule a thread on the same processor on which it recently executed to exploit data cached on a processor's private cache. The system also uses the ideal processor attribute to increase the likelihood that a process's threads execute concurrently, thus increasing performance. Alternatively, developers can override this default and use the ideal processor attribute to increase the likelihood that a process's threads execute on the same processor to share cached data. 450 Windows XP implements spin locks (described in Section 21.6.3, Thread Synchronization), which are kernel locks designed for multiprocessor systems. Specifically, the queued spin lock decreases traffic on the processor-to-memory bus, increasing Windows XP's scalability.451 In addition, as the original NT kernel has evolved (recall that Windows XP is NT 5.1), designers have reduced the size of critical sections, which reduces the amount of time a processor is busy waiting to obtain a lock. Sometimes, a processor cannot perform other work when waiting for a spin lock, such as when a processor needs to execute thread scheduling code. Reducing the size of this critical section increases Windows XP's scalability.452 Additionally, Windows XP provides a number of other fine-grained locks (e.g., dispatcher objects and executive resource locks), so that developers need only lock specific sections of the kernel, allowing other components to execute without disruption. 453 Memory support for large SMP systems emanates from Windows XP 64-Bit Edition. 64-bit addressing provides 2 64 bytes (or 16 exabytes) of virtual address space; this contrasts with 2 32 bytes (or 4 gigabytes) for 32-bit versions of Windows XP. In practice, Windows XP 64-Bit Edition provides processes with 7152GB of virtual memory address range. 454 Windows XP 64-Bit Edition currently supports up to 1 terabyte of cache memory, 128 gigabytes of paged system memory and 128 gigabytes of nonpaged pool. Windows XP 64-Bit Edition has been ported to Intel Itanium II processors, which are described in Section 14.8.4, Explicitly Parallel Instruction Computing (EPIC) 455 . Soon (as of this writing), Microsoft plans to release Windows XP 64-Bit Edition for Extended Systems to support AMD Opteron and Athlon 64 processors.456 The 64-bit editions of Windows also offer greater performance and precision when manipulating floating point numbers, because these systems store floating point numbers using 64 bits.457 Finally, Windows XP has added several programming features tailored for high-performance systems, which is an environment in which many SMP systems are employed. The job object (described in Section 21.6.1, Process and Thread Organization) allows servers to handle resource allocation to a group of processes. In this way, a server can manage the resources it commits to a client request. 458 Also, I/O completion ports (described in Section 21.9.2, Input/Output Processing) facilitate the processing of large numbers of asynchronous I/O requests. In a large server program, many threads might issue asynchronous I/O requests; I/O completion ports provide a central area where dedicated threads can wait and process incoming I/O completion notifications.459 Additionally, thread pools (described in Section 21.6.1, Process and Thread Organization) permit server processes to handle incoming client requests without the costly overhead of creating and deleting threads for each new request. Instead, these requests can be queued to the thread pool.460 21.12.2 Windows XP Embedded Microsoft has created an operating system called Windows XP Embedded, which scales Windows XP for embedded devices. Microsoft divides Windows XP into clearly defined functional units called components. Components range from usermode applications such as Notepad to portions of kernel space such as the CDROM interface and power management tools. The designer of a Windows XP Embedded system chooses from over 10,000 components to customize a system for a particular device.461 Components for Windows XP Embedded contain the same code as Windows XP Professional, providing full compatibility with Windows XP. Windows XP Embedded also contains the core Windows XP microkernel.462 Windows XP Embedded is designed for more robust embedded devices such as printers, routers, point-of-sale systems and digital set-top boxes. Microsoft provides a different operating system, Windows CE, for lighter devices such as cellular phones and PDAs. The goal of Windows XP Embedded is to allow devices with limited resources to take advantage of the Windows XP computing environment.463 Devices such as medical devices, communication equipment and manufacturing equipment would benefit from Windows XP Embedded, but require hard realtime guarantees. Recall from Section 8.9, Real-Time Scheduling, the difference between soft real-time scheduling (like that provided by Windows XP) and hard real-time scheduling. Soft real-time scheduling assigns higher priority to real-time threads, whereas hard real-time scheduling provides guarantees that, given a start time, a thread will finish by a certain deadline. Venturcom464 has designed a RealTime Extension (RTX) for Windows XP Embedded and Windows XP. RTX includes a new threading model (implemented as a device driver), a collection of libraries and an extended HAL. Venturcom's RTX achieves hard real-time goals by masking all non-real-time interrupts when real-time threads execute, never masking real-time interrupts (recall that interrupt masking is accomplished through the HAL) and enforcing strict thread priorities. For example, threads waiting for synchronization objects, such as mutexes and semaphores, are placed in the ready state in priority order (as opposed to FIFO order). RTX implements 128 priority levels to increase scheduling flexibility. 21.13 Security Any operating system aimed at the corporate market must offer substantial security capabilities. Windows XP Professional provides a wide variety of security services to its users, whereas Windows XP Home Edition has a slightly more limited range and is targeted at consumers with simple security needs. The following subsections 21.13 describe security in Windows XP Professional. Much of the information, especially non-networking related material, applies to Windows XP Home Edition as well. Windows XP provides users with a range of authentication and authorization options that permit access a physical terminal, a local network or the Internet. A database of users' credentials helps provide users with single sign-on; once a user logs onto Windows XP, the system takes care of authenticating the user to other domains and networks without prompting for credentials. The operating system includes a built-in firewall (see Section 19.6.1, Firewalls) that protects the system from malicious access. 21.13.1 Authentication Windows XP has a flexible authentication policy. Users provide the system with credentials which consists of their identity (e.g. username) and proof of identity (e.g. password). For local access, the system authenticates a user's credentials against the Security Account Manager (SAM) database. A designated computer, called the domain controller, handles remote authentication. The native authentication system for network access is Kerberos V5 (see Section 19.3.3, Kerberos). Windows XP also supports the NT LanMan (NTLM) authentication protocol for backward compatibility with domain controllers running NT 4.O.465 Authentication begins at a logon screen; this can be either the standard logon program (called MSGINA) or a third-party Graphical Identification and Authentication (GINA) DLL. 466 GINA passes the user's username and password to the Winlogon program, which passes the credentials to the Local Security Authority (LSA). The LSA is responsible for handling all authentication and authorization for users physically accessing that particular machine.467 If the user is logging on only to the local machine, the LSA verifies the user's credentials with the local SAM database. Otherwise, the LSA passes the credentials over the network to the domain controller (the computer responsible for handling authentication) via the Security Support Provider Interface (SSPI). This is a standard interface for obtaining security services for authentication that is compatible with both Kerberos V5 and NTLM. The domain controller first attempts to authenticate using Kerberos, which verifies credentials using the Active Directory. If Kerberos fails, GINA resends the username and password. This time the domain controller attempts to authenticate via the NTLM protocol, which relies on a SAM database stored on the domain controller. If both Kerberos and NTLM fail, GINA prompts the user to reenter the username and password.468, 469 Figure 21.19 illustrates the authentication process. Computers, especially mobile ones such as laptops, often use different credentials to connect to different domains. This is done for security reasons; if one domain is compromised, accounts on the other domains remain secure.(Windows XP lets users store domain-specific credentials in Stored User Names and Passwords. Upon an authentication error, the LSA first attempts to authenticate using a saved credential for that domain. Upon failure, the LSA activates the GINA screen Security 1109 1110 Case Study: Windows XP Figure 21.19 | Authentication process in a Windows XP network. to prompt the user for a credential. The screen lets the user select a stored credential from a drop-down menu. The system records every credential that a user enters to a GINA (as long as the user selects a box giving the system permission to do so). In addition, users may use a special GUI to add, edit and remove credentials.470 21.13.2 Authorization Any user, group, service or computer that performs an action in a Windows XP system is considered a security principal. A user can set security permissions for any security principal.471 It is often advantageous to place users with the same security privileges into a security group, then set security permissions based on the security group. Security groups can be nested and grouped to form larger security groups. Using security groups simplifies managing the security policy for a large system.472 The system assigns each security principal a unique security identifier (SID). When a security principal logs on, the system uses the security principal's SID to assign an access token. The access token stores security information about the security principal, including its SID, the SID of all groups to which the security principal belongs and the default access control list (see Section 19.4.3, Access Control Mechanisms) to assign to all objects the security principal creates. The system attaches a copy of the access token to every process and thread that the security principal runs. The system uses a thread's access token to verify that the thread is authorized to perform a desired action.473 Windows XP uses access tokens to implement fast user switching—enabling a new user to log on without logging off the current user. (Fast user switching works only on computers that are not members of a domain). Often, especially in home environments, a user may start a long-running process, then leave the computer to 21.13 Security 1111 execute unattended. Fast user switching permits another person to log on, run programs, check e-mail, play a game, and so on. The first user's processes continue executing in the background; however, this is transparent to the new user. An access token stores a session ID that is unique to each logon session. The system uses the access token attached to each thread and process to determine to which user a process or thread belongs.474, 475 Every resource, such as a file, folder or device, has a security descriptor that contains the resource's security information. A descriptor contains the resource owner's SID and a discretionary access control list (DACL) that determines which security principals may access the resource. The DACL is an ordered list of access control entries (ACEs); each ACE contains the SID of a security principal and the type of access (if any) the security principal has to that particular resource. 476 When a thread attempts to access a resource, Windows XP uses the thread's access token and the resource's security descriptor to determine whether to permit access. The system traverses the DACL and stops when it finds an ACE with a SID that matches one of the SIDs in the access token. The thread is permitted to perform the requested operation only if the first matching ACE authorizes the action. If the security descriptor has no DACL, access is granted; if the DACL has no matching ACE, access is denied. 477 Windows XP offers users flexibility in specifying the security policy of their system. The main role of the operating system is to enforce the policy that its users set. However, with so many options to choose from, an administrator might inadvertently create an insecure security policy. To prevent this, Windows XP includes many useful features to help administrators maintain system security. Recall the notion of security groups mentioned at the start of the section. Windows XP defines three basic security groups: Everyone (all authenticated security principals), Anonymous (all unauthenticated security principals) and Guest (a general-purpose account). System administrators can use these three security groups as the building blocks of their security policy. In addition, Windows XP includes several default security policies that administrators can employ as a starting point for creating their own policy. In general, each default policy uses the highest level of security, and system administrators are responsible for increasing user privileges. For example, by default, users who log on from a remote machine are forced to use Guest privileges. Although this might be an annoyance, this policy is preferable to inadvertently overlooking a security hole.478 21.13.3 Internet Connection Firewall Windows XP has a built-in packet filtering firewall (see 18.6.1, Firewalls) called the Internet Connection Firewall (ICF). This firewall can be used to protect either an entire network or a single computer.479 ICF filters out all unsolicited inbound traffic, with the exception of packets sent to ports explicitly designated as open. The firewall stores information about all outgoing packets in a flow table. It permits an incoming packet to enter the system only if there is an entry in the flow table corre- 1112 Case Study: Windows XP sponding to an outgoing packet sent to the home address of the incoming packet. 480 ICF does not filter outgoing packets; it assumes that only legitimate processes would send an outgoing packet. (This can cause problems if a Trojan Horse program gains access to the system.)481 ICF supports common Internet communication protocols such as FTP and LDAP. To permit applications to communicate with a Windows XP system via these protocols, ICF leaves certain ports open for all incoming traffic. Also, many applications such as MSN Messenger and online games require an open port to run successfully. Users and applications with administrative privileges can designate a particular port open for all incoming traffic; this process is called port mapping.482, 483 21.13.4 Other Features Windows XP comes with many features that help keep the system secure. For example, users can encrypt files and folders using the Encrypting File System (see Section 21.8.2, NTFS, for more information). Users can control which Web sites may store cookies (small temporary files used by a server to store user-specific information) on their computer. Administrators can limit which applications may execute on the system; each application can be identified by its file hash, so that moving or renaming it would not bypass the restriction. Windows XP can also check whether the application is signed with a trusted certificate (i.e., a signature from an entity that the user specified as trustworthy) or comes from a trusted Internet Zone (i.e., an Internet domain that the user specified as safe).484 Users can use Automatic Update to get notification, or even installation, of the latest security patches.485 Web Resources msdn.microsoft.com/library The MSDN Library provides information about Microsoft products and services, including the most recent platform Software Development Kit (SDK) and Driver Development Kit (DDK). Go to the menu on the left side of the screen and click on Windows Development, followed by Windows Base Services, to research the Windows XP SDK. To research the DDK, click on Windows Development, followed by Driver Development Kit. Particularly useful information can be found by next clicking Kernel-Mode Driver Architecture, followed by Design Guide. The MSDN Library also provides networking information (click Networking and Directory Services) and COM information (click Component Development). www.microsoft.com/windowsxp/default.asp Microsoft's homepage for Windows XP. From here, the reader can find XP overviews, support information and gateways to other areas of Microsoft with more technical information on Windows XP. www.microsoft.com/technet/prodtechnol/winxppro Microsoft TechNet provides technical information on Microsoft products. This section is devoted to Windows XP Professional; particularly useful is the Resource Kit (click Resource Kits), www.winnetmag.com Publishes technical articles about Windows & .NET ranging from introductions for novice users to in-depth internals descriptions for advanced developers. www.osronline.com Features technical articles and tutorials about the internals of Windows operating systems. It offers online articles, as well as a free subscription to NT Insider, a technical magazine about the NT line of operating systems. www.extremetech.com/category2/0,3971,23445,00.asp ExtremeTech's page for operating systems. It includes news articles, some technical insight and a forum to discuss operating systems. Key www.theeldergeek.com/index.htm Describes Windows XP from the user's perspective, including excellent information about the registry, system services and networking. www.windowsitlibrary.com Provides free articles and sells books and articles about Windows operating systems. Some of these articles and books provide technical internals information. www.sysinternals.com Dedicated to providing internals information about Windows operating systems, especially the NT line. msdn.microsoft.com/msdnmag/default.aspx The MSDN Magazine (formerly know as Microsoft Systems Journal) contains articles about Microsoft's products, targeted for developers. Terms 1113 www.winsupersite.com Provides updated information about Windows products. It evaluates them from both technical and user perspectives. developer.intel.com Provides technical documentation regarding Intel products, including Pentium and Itanium processors. Their system programmer guides (e.g., for the Pentium 4 processor, developer.intel.com/design/Pentium4/manuals/) provide information about how Intel processors interact with the operating system, developer.amd.com Provides technical documentation regarding AMD products, including Athlon and Opteron processors. Their system programmer guides (e.g., for the Opteron processor, www.amd.com/us-en/Processors/DevelopWithAMD/ 0,,30_2252_739_7044,00.html) provide information about how AMD processors interact with the operating system. Key Terms .NET—Microsoft initiative aimed at transforming computing from an environment in which users simply execute applications on a single computer to a fully distributed environment; .NET provides a platform- and language-neutral development environment for components that can easily interoperate. .NET framework—Programming model for creating XMLbased Web services and applications. The .NET framework supports over 20 programming languages and facilitates application programming by providing libraries to perform common operations (e.g., input/output, string manipulation and network communication) and access data via multiple database interfaces (e.g., Oracle and SQL Server). abandoned mutex (Windows XP)—Mutex that was not released before the termination of the thread that held it. access control entry (ACE) (Windows XP)—Entry in a discretionary access control list (DACL) for a specific resource that contain the security identifier (SID) of a security principal and the type of access (if any) the security principal has to that particular resource. ActiveX Controls—Self-registering COM objects useful for embedding in Web pages. affinity mask (Windows XP)—List of processors on which a process's threads are allowed to execute. alertable I/O (Windows XP)—Type of asynchronous I/O in which the system notifies the requesting thread of the completion of the I/O with an APC. alertable wait state (Windows XP)—State in which a thread cannot execute until it is awakened either by an APC entering the thread's APC queue or by receiving a handle to the object (or objects) for which the thread is waiting. alternate data stream—File attribute that stores the contents of an NTFS file that are not in the default data stream. NTFS files may have multiple alternate data streams. anonymous pipe (Windows XP)—Unnamed pipe that can be used only for synchronous one-way communication between local processes. apartment model—COM object threading model in which only one thread acts as a server for each COM object. access token (Windows XP)—Data structure that stores security information, such as the security identifier (SID) and group memberships, about a security principal. APC IRQL (Windows X P) - I R Q L at which APCs execute and incoming asynchronous procedure calls (APCs) are masked. Active Directory (Windows XP)—Network service that provides directory services for shared objects (e.g., files, printers, services, etc.) on a network. application configuration file (ACF) (Windows XP)—File that specifies platform-specific attributes (e.g., how data should be formatted) for an RPC function call. 1114 Case Study: Windows XP asynchronous procedure call (APC) (Windows XP)—Procedure calls that threads or the system can queue for execution by a specific thread. automatic binding handle (Windows XP) —RPC binding handle in which the client process simply calls a remote function, and the stub manages all communication tasks. auto-reset timer object (Windows XP)—Timer object that remains signaled only until one thread finishes waiting on the object. Bad Page List (Windows XP)—List of page frames that have generated hardware errors; the system does not store pages on these page frames. balance set manager (Windows XP)—Executive component responsible for adjusting working set minimums and maximums and trimming working sets. base priority class (Windows XP)—Process attribute that determines a narrow range of base priorities the process's threads can have. base priority level (Windows XP)—Thread attribute that describes the thread's base priority within a base priority class. binding (Windows XP) — Connection between the client and the server used for network communication (e.g., for an RPC). binding handle (Windows XP)—Data structure that stores the connection information associated with a binding between a client and a server. bus driver—WDM (Windows Driver Model) driver that provides some generic functions for devices on a bus, enumerates those devices and handles Plug and Play I/O requests. cache manager (Windows XP)—Executive component that handles file cache management. class ID (CLSID)- Globally unique ID given to a COM object class. class/miniclass driver pair (Windows XP)—Pair of device drivers that act as a function driver for a device; the class driver provides generic processing for the device's particular device class, and the miniclass driver provides processing for the specific device. clipboard (Windows XP)—Central repository of data that is accessible to all processes, typically used with copy, cut and paste commands. clock IRQL (Windows XP)—IRQL at which clock interrupts occur and at which APC, DPC/dispatch, DIRQL and profile-level interrupts are masked. cluster (NTFS)—Basic unit of disk storage in an NTFS volume, consisting of a number of contiguous sectors; a system's cluster size can range from 512 bytes to 64KB, but is typically 2KB, 4KB or 8KB. COM+—Extension of COM (Microsoft's Common Object Model) that handles advanced resource management tasks, such as providing support for transaction processing and using thread and object pools. commit memory (Windows XP)—Necessary stage before a process can access memory. The VMM ensures that there is enough space in a pagefile for the memory and creates page table entries (PTEs) in main memory for the committed pages. Common Internet File System (CIFS)—Native file sharing protocol of Windows XP. Component Object Model (COM)—Microsoft-developed software architecture that allows interoperability between diverse components through the standard manipulation of object interfaces. component (Windows XP)—Functional unit of Windows XP. Components range from user-mode applications such as Notepad to portions of kernel space such as the CD-ROM interface and power management tools. Dividing Windows XP into components facilitates operating system development for embedded systems. compression unit (NTFS)—Sixteen clusters. Windows XP compresses and encrypts files one compression unit at a time. configuration manager (Windows XP)—Executive component that manages the registry. critical section object (Windows XP)—Synchronization object, which can be employed only by threads within a single process, that allows only one thread to own a resource at a time. data stream (NTFS)—File attribute that stores the file's content or metadata; a file can have multiple data streams. default data stream (NTFS)—File attribute that stores the primary contents of an NTFS file. When an NTFS file is copied to a file system that does not support multiple data streams, only the default data stream is preserved. deferred procedure call (DPC) (Windows XP)—Software interrupt that executes at the DPC/dispatch IRQL and executes in the context of the currently executing thread. device extension (Windows XP)—Portion of the nonpaged pool that stores information a driver needs to process I/O requests for a particular device. device IRQL (DIRQL) (Windows X P) - I R Q L at which devices interrupt and at which APC, DPC/dispatch and Key lower DIRQLs are masked; the number of DIRQLs is architecture dependent. device object (Windows XP)—Object that a device driver uses to store information about a physical or logical device. direct I/O —I/O transfer method whereby data is transferred between a device and a process's address space without using a system buffer. directory junction (Windows XP)—Directory that refers to another directory within the same volume, used to make navigating the file system easier. discretionary access control list (DACL) (Windows XP) — Ordered list that tells Windows XP which security principals may access a particular resource and what actions those principals may perform on the resource. dispatcher (Windows XP)—Thread scheduling code dispersed throughout the microkernel. dispatcher object (Windows XP)—Object, such as a mutex, semaphore, event or waitable timer, that kernel and user threads can use for synchronization purposes. domain (Windows XP) — Set of computers that share common resources. domain controller (Windows XP) — Computer responsible for security on a network. DPC/dispatch IRQL (Windows X P) - I R Q L at which DPCs and the thread scheduler execute and at which APC and incoming DPC/dispatch level interrupts are masked. driver object (Windows XP)—Object used to describe device drivers; it stores pointers to standard driver routines and to the device objects for the devices that the driver services. driver stack—Group of related device drivers that cooperate to handle the I/O requests for a particular device. dynamic-linked library (DLL) (Windows XP)—Module that provides data or functions to which processes or other DLLs link at execution time. dynamic priority class (Windows XP)—Priority class that encompasses the five base priority classes within which a thread's priority can change during system execution; these classes are idle, below normal, normal, above normal and high. environment subsystem (Windows XP)—User-mode component that provides a computing environment for other user-mode processes; in most cases, only environment subsystems interact directly with kernel-mode components in Windows XP. event object (Windows XP)—Synchronization object that becomes signaled when a particular event occurs. Terms 1115 executive (Windows XP)—Portion of the Windows XP operating system that is responsible for managing the operating system's subsystems (e.g., I/O subsystem, memory subsystem and file system). executive resource lock (Windows XP)—Synchronization lock available only to kernel-mode threads and that may be held either in shared mode by many threads or in exclusive mode by one thread. executive process (EPROCESS) block (Windows XP)—Executive data structure that stores information about a process, such as that process's object handles and the process's ID; an EPROCESS block also stores the process's KPROCESS block. executive thread (ETHREAD) block (Windows XP)—Executive data structure that stores information about a thread, such as the thread's pending I/O requests and the thread's start address; an ETHREAD block also stores the thread's KTHREAD block. explicit handle (Windows XP)—Binding handle in which the client must specify all binding information and create and manage the handle. fast mutex (Windows XP)—Efficient mutex variant that operates at the APC level with some restrictions (e.g., a thread cannot specify a maximum wait time to wait for a fast mutex). fast user switching (Windows XP)—Ability of a new user to logon to a Windows XP machine without logging off the previous user. fiber (Windows XP)—Unit of execution created by a thread and scheduled by that thread. fiber local storage (FLS) (Windows XP)—Area of a process's address space where a fiber can store data that only the fiber can access. file mapping (Windows XP)—Interprocess communication mechanism in which multiple processes access the same file by placing it in their virtual memory space. The different virtual addresses correspond to the same main memory addresses. file mapping object (Windows XP) —Object used by processes to map any file into memory. file view (Windows XP)—Portion of a file specified by a file mapping object that a process maps into its memory. filter driver—WDM (Windows Driver Model) driver that modifies the behavior of a hardware device (e.g., providing mouse acceleration) or adds some extra services (e.g., security checks). 1116 Case Study: Windows XP free model—COM (Microsoft's Component Object Model) object threading model in which many threads can act as the server for a COM object. Free Page List (Windows XP)—List of page frames that are available for reclaiming; although these page frames do not contain any valid data, they may not be used until the zero-page thread sets all of their bits to zero. function driver—WDM (Windows Driver Model) device driver that implements a device's main functions; it does most of the I/O processing and provides the device's interface. I/O manager (Windows XP)—Executive component that interacts with device drivers to handle I/O requests. I/O throttling (Windows XP)—Technique that increases stability when available memory is low; the VMM manages memory one page at a time during I/O throttling. I/O status block (Windows XP)—Field in an IRP that indicates whether an I/O request completed successfully or, if not, the request's error code. Interface Definition Language (IDL) file (Windows XP)—File that specifies the interfaces that an RPC server exposes. globally unique ID (GUID) —128-bit integer that is, for all practical purposes, guaranteed to be unique in the world. COM (Microsoft's Component Object Model) uses GUIDs to uniquely identify interfaces and object classes. interface ID (IID) — Globally unique ID for a COM interface. Graphical Identification and Authentication (GINA) (Windows XP) — Graphical user interface used that prompts users for credentials, usually in the form of a username and password. Windows XP ships with its own implementation of GINA called MSCINA, but it accepts third-party DLLs. interlocked variable access (Windows XP) —Method of accessing variables that ensures atomic reads and writes to shared variables. Hardware Abstraction Layer (HAL) (Windows XP)—Operating system component that interacts directly with the hardware and abstracts hardware details for other system components. high IRQL (Windows XP)—Highest-priority IRQL, at which machine check and bus error interrupts execute and all other interrupts are masked. high-level driver (Windows XP)—Device driver that abstracts hardware specifics and passes I/O requests to low-level drivers. ideal processor (Windows XP)—Thread attribute that specifies a processor on which the system should attempt to schedule the thread for execution. interlocked singly linked list (SList) (Windows XP) —Singly linked list in which insertions and deletions are performed as atomic operations. intermediate driver (Windows XP)—Device driver that can be interposed between high- and low-level drivers to filter or process I/O requests for a device. Internet Connection Firewall (ICF)—Windows XP's packet filtering firewall. Internetwork Packet eXchange (IPX)—Novell Netware's network protocol designed specifically for LANs. Interrupt Dispatch Table (IDT)—Kernel data structure that maps hardware interrupts to interrupt vectors. interrupt request level (IRQL) (Windows XP)—Measure of interrupt priority; an interrupt that occurs at an IRQL equal to or lower than the current IRQL is masked. interrupt service routine (ISR) (Windows XP)—Function,registered by a device driver, that processes interrupts issued by the device that the driver services. inheritance (Windows XP)—Technique by which a child process obtains attributes (e.g., most types of handles and the current directory) from its parent process upon creation. job object (Windows XP)—Object that groups several processes and allows developers to manipulate and set limits on these processes as a group. initialized state (Windows XP) —State in which a thread is created. kernel handle (Windows XP)—Object handle accessible from any process's address space, but only in kernel mode. I/O completion port (Windows XP)—Port at which threads register and block, waiting to be awakened when processing completes on an I/O request. kernel-mode APC (Windows XP)—APC generated by a kernel-mode thread and queued to a specified user-mode thread; the user-mode thread must process the APC as soon as the user-mode thread obtains the processor. I/O completion routine (Windows XP)—Function registered by a device driver with the I/O manager in reference to an IRP; the I/O manager calls this function when processing of the IRP completes. I/O request packet (IRP) (Windows XP)—Data structure that describes an I/O request. kernel-mode driver—Device driver that executes in kernel mode. kernel process (KPROCESS) block (Windows XP)—Kernel data structure that stores information about a process, such as that process's base priority class. Key Terms 1117 kernel thread (KTHREAD) block (Windows X P) - K e r n e l data structure that stores information about a thread, such as the objects on which that thread is waiting and the location in memory of the thread's kernel stack. memory descriptor list (MDL) (Windows XP)—Data structure used in an I/O transfer that maps a process's virtual addresses to be accessed in the transfer to physical memory addresses. large page (Windows XP) — Set of pages contiguous in memory that the VMM treats as a single page. memory map—Data structure that stores the correspondence between a process's virtual address space and main memory locations. last processor (Windows XP)—Thread attribute equal to the processor that most recently executed the thread. lazy allocation—Policy of waiting to allocate resources, such as pages in virtual memory and page frames in main memory, until absolutely necessary. Lempel-Ziv compression algorithm—Data compression algorithm that NTFS uses to compress files. Lightweight Directory Access Protocol (LDAP) (Windows XP)—Protocol for accessing, searching and modifying Internet directories (e.g., Active Directory). localized least-recently used page replacement (Windows XP)—Policy of moving to disk the least-recently used page of a process at its working set maximum when that process requests a page in main memory. The policy is localized by process, because Windows XP moves only the requesting process's pages; Windows XP approximates this policy using the clock algorithm. local procedure call (LPC) (Windows XP)—Procedure call made by a thread in one process to a procedure exposed by another process in the same domain (i.e., set of computers that share common resources); LPCs can be created only by system components. local remote procedure call (LRPCs) (Windows X P) - R P C s between two processes on the same machine. low-level driver (Windows XP)—Device driver that controls a peripheral device and does not depend on any lower-level drivers. mailslot (Windows XP)—Message queue, which a process can employ to receive messages from other processes. mailslot client (Windows XP)—Process that sends mailslot messages to mailslot servers. mailslot server (Windows XP)—Process that creates a mailslot and receives messages in it from mailslot clients. major function code (Windows XP)—Field in an IRP that describes the general function (e.g., read or write) that should be performed to fulfill an I/O request. manual-reset timer object (Windows XP)—Timer object that remains signaled until a thread specifically resets the timer. Master File Table (MFT) (NTFS)-File which is structured as a table in which NTFS stores information (e.g. name, time stamp, and location) about all files in the volume. Microsoft IDL (MIDL)—Microsoft's extension of Interface Definition Language (IDL)—The Open Group's Distributed Computing Environment (DCE) standard for RPC interoperability. minor function code (Windows XP)—Field in an IRP that, together with the major function code, describes the specific function that should be performed to fulfill an I/O request (e.g., to start a device, a PnP major function code is used with a start device minor function code). mixed model—COM threading model, in which some COM objects reside in single apartments and others can be accessed by free threads. Modified No-write Page List (Windows XP)—List of page frames for which the VMM must write an entry to a log file before freeing. Modified Page List (Windows XP)—List of page frames that the VMM must write to the pagefile before freeing. Multiple UNC Provider (MUP) (Windows X P) - F i l e system driver that determines the appropriate redirector to which to send a network I/O request. must-succeed request (Windows XP) —Request for space in main memory that a process issues when it requires more main memory to continue functioning properly. Windows XP always denies must-succeed requests; previous versions of Windows always fulfilled them. mutex object (Windows XP) — Synchronization object that allows at most one thread access to a protected resource at any time; it is essentially a binary semaphore. named pipe (Windows XP)—Type of pipe that can provide bidirectional communication between two processes on local or remote machines and that supports both synchronous and asynchronous communication. native API (Windows XP)—Programming interface exposed by the executive, which environment subsystems employ to make system calls. NDIS intermediate driver (Windows XP)—NDIS driver that can be interposed between a miniport driver and a higherlevel driver and adds extra functionality, such as translating packets between different communication media, filtering packets or providing load balancing across several NICs. 1118 Case Study: Windows XP NDIS miniport driver (Windows XP)—NDIS driver that manages a NIC and sends and receives data to and from the NIC. NDIS protocol driver (Windows XP)—NDIS driver that places data into packets and passes these packets to lower-level drivers. NDIS protocol drivers provide an interface between the transport drivers and other NDIS drivers and can be used as the lowest layer in the implementation of a transport protocol stack such as TCP/IP. neither I/O (Windows XP)—I/O transfer technique in which a high-level driver executes in the context of the calling thread to set up a buffered I/O transfer or direct I/O transfer or to directly perform the transfer in the process's address space. NetBIOS Extended User Interface (NetBEUI) (Windows XP)—Native network and transport protocol for DOS and older Windows operating systems. NetBIOS over TCP/IP (NBT) (Windows XP)-Replacement protocol for NetBEUI that provides backward compatibility with older applications (by keeping the NetBIOS interface) and takes advantage of TCP/IP protocols. Network Basic Input/Output System (NetBIOS) (Windows XP)—API used to support NetBEUI and now used with NBT. Network Data Representation (NDR)—Standard format for network data described in the The Open Group's Distributed Computing Environment (DCE) standard. Network Driver Interface Specification (NDIS) (Windows XP)—Specification that describes a standard interface between lower-level layered drivers in the network driver stack; NDIS drivers provide the functionality of the link layer in the TCP/IP stack and are used on Windows XP systems. network redirector—See redirect or (Windows XP). nonpaged pool—Area of main memory that stores pages that are never moved to disk. nonresident attribute—NTFS file attribute whose data does not fit inside the MFT entry and is stored elsewhere. object handle (Windows XP)—Data structure that allows threads to manipulate an object. Object Linking and Embedding (OLE)—Microsoft technology built on COM that defines a standard method for processes to exchange data by linking or embedding COM objects. object manager (Windows XP)—Executive component that manages objects. object manager namespace (Windows XP)—Group of object names in which each name is unique in the group. opportunistic lock (oplock) (Windows XP) —Lock used by a CIFS client to secure exclusive access to a remote file, ensuring that the client can cache data locally and keep its cache coherent. overlapped I/O (Windows XP)—Microsoft synonym for asynchronous I/O. page directory register—Hardware register that stores a pointer to the current process's page directory table. page directory table (Windows XP)—4KB page that contains 1024 entries that point to frames in memory. paged pool (Windows XP)—Pages in memory that may be moved to the pagefile on disk. pagefile (Windows XP)—File on disk that stores all pages that are mapped to a process's virtual address space but do not currently reside in main memory. page frame database (Windows XP)—Array that contains the state of each page frame in main memory. page list (Windows XP)—List of page frames that are in the same state. The eight state lists are: the Valid Page List, the Standby Page List, the Modified Page List, the Modified No-write Page List, the Transitional Page List, the Free Page List, the Zeroed Page List, and the Bad Page List. page trimming (Windows XP)—Technique in which Windows XP takes a page belonging to a process and sets the page's PTE to invalid to determine whether the process actually needs the page. If the process does not request the page within a certain time period, the system removes it from main memory. passive IRQL (Windows XP)—IRQL at which user- and kernel-mode normally execute and no interrupts are masked. Passive IRQL is the lowest-priority IRQL. periodic timer (Windows XP)—Waitable timer object that reactivates after a specified interval. pipe client (Windows XP)—Process that connects to an existing pipe to communicate with that pipe's server. pipe server (Windows XP)—Process that creates a pipe and communicates with pipe clients that connect to the pipe. PnP I/O requests (Windows X P) - I R P s generated by the PnP manager to query a driver for information about a device, to assign resources to a device or to direct a device to perform some action. Plug and Play (PnP) manager (Windows XP)—Executive component (which also exists partly in user space) that dynamically recognizes when new devices are added to Key Terms 1119 the system (as long as these devices support PnP), allocates and deallocates resources to devices and interacts with device setup programs. port mapping (Windows XP)—Process of explicitly allowing the Internet Connection Firewall (ICF) to accept all incoming packets to a particular port. power IRQL (Windows XP) —IRQL at which power failure interrupts execute and at which all interrupts except highIRQL interrupts are masked. power manager (Windows XP)—Executive component that administers the operating system's power management policy. power policy (Windows XP)—Policy of a system with regard to balancing power consumption and responsiveness of devices. primary thread (Windows XP)—Thread created when a process is created. priority inversion—Situation which occurs when a high-priority thread is waiting for a resource held by a low-priority thread, and the low-priority thread cannot obtain the processor because of a medium-priority thread; hence, the high-priority thread is blocked from execution by the medium-priority thread. process environment block (PEB) (Windows XP) — Userspace data structure that stores information about a process, such as a list of DLLs linked to the process and information about the process's heap. profile IRQL (Windows XP)—IRQL at which debugger interrupts execute and at which APC, DPC/dispatch, DIRQL and incoming debugger interrupts are masked. protocol sequence (Windows XP)—String used by a client in an RPC call that specifies the RPC protocol, transport protocol and network protocol for that RPC. Prototype Page Table Entry (PPTE) (Windows XP)-32-bit record that points to a frame in memory that contains either a copy-on-write page or a page that is part of a process's view of a mapped file. when a user has forgotten the private key needed to decrypt the file. redirector (Windows XP)—File system driver that interacts with a remote server driver to facilitate network I/O operations. registry (Windows XP)—Central database in which user, system and application configuration information is stored. Remote Access Service (RAS) (Windows XP)—Network service, which allows users to remotely connect to a LAN. reparse point (NTFS)—File attribute containing a tag and a block of up to 16KB of data that a user associates with a file or directory; when an application accesses a reparse point, the system executes the file system filter driver specified by the reparse point's tag. request IRQL (Windows XP) —IRQL at which interprocess interrupts execute and all interrupts except power-level and high-level interrupts are masked. reserve memory (Windows XP)—To indicate to the VMM that a process intends to use an area of virtual memory; the VMM allocates memory in the process's virtual address space but does not allocate any page frames in main memory. resident attribute (NTFS)—File attribute whose data is stored within the MFT entry. running state (Windows XP) —State in which a thread is currently in execution. Security Accounts Manager (SAM) (Windows XP)—Database that administers information about all security principals in the system. It provides services such as account creation, account modification and authentication. security descriptor (Windows XP)—Data structure that stores information about which security principals may access a resource and what actions they may perform on that resource. The most important element of a security descriptor is its discretionary access control list (DACL). security identifier (SID) (Windows XP)—Unique identification number assigned to each security principal in the system. queued spin lock (Windows XP)—Spin lock in which a thread releasing the lock notifies the next thread in the queue of waiting threads. security principal (Windows XP) —Any user, process, service or computer that can perform an action in a Windows XP system. ready state (Windows XP)—State in which a thread is waiting to use a processor. Security Support Provider Interface (SSPI) (Windows XP) — Microsoft's standardized protocol for authentication and authorization recognized by both the Kerberos and NTLM authentication services. real-time priority class (Windows XP)—Base priority class encompassing the upper 16 priority levels; threads of this class have static priorities. recovery key (NTFS)—Key that NTFS stores that can decrypt an encrypted file; administrators can use recovery keys semaphore object (Windows XP) — Synchronization object that allows a resource to be owned by up to a specified number of threads; it is essentially a counting semaphore. 1120 Case Study: Windows XP Sequenced Packet eXchange (SPX)—Novell Netware's transport protocol, which offers connection-oriented services for IPX packets. transition state (Windows XP)—Thread state denoting a thread that has completed a wait but is not yet ready to run because its kernel stack has been paged out of memory. Server Message Block (SMB)—Network file-sharing protocol used in Windows operating systems on top of which CIFS (Common Internet File System) is built. Transport Driver Interface (TDI) driver (Windows X P) - N e t work driver that exports a network interface to upperlevel kernel-mode clients and interacts with low-level NDIS drivers. single-use timer (Windows XP)—Waitable timer object that is used once and then discarded. signaled state (Windows XP)—State in which a synchronization object can be placed, allowing one or more threads waiting on this object to awaken. sparse file (NTFS)—File with large blocks of regions filled with zeroes that NTFS tracks using a list of empty regions rather than explicitly recording each zero bit. Standby Page List (Windows XP)—List of page frames that are consistent with their on-disk version and can be freed. standby state (Windows XP)—Thread state denoting a thread that has been selected for execution. system service (Windows XP)—Process that executes in the background whether or not a user is logged into the computer and typically executes the server side of a client/ server application; a system service for Windows XP is similar to a daemon for Linux. system worker thread (Windows XP)—Thread controlled by the system that sleeps until a kernel-mode component queues a work item for processing. terminated state (Windows XP)—Thread state that denotes a thread is no longer available for execution. thread local storage (TLS) (Windows XP)—Area of a thread's process's address space where the thread can store private data, inaccessible to other threads. thread environment block (TEB) (Windows XP)—User-space data structure that stores information about a thread, such as the critical sections owned by a thread and exception-handling information. thread pool (Windows XP) — Collection of worker threads that sleep until a request is queued to them, at which time one of the threads awakens and executes the queued function. timer-queue timer (Windows XP)—Timer that signals a worker thread to perform a specified function at a specified time. transitional fault (Windows XP)—Fault issued by the MMU when it tries to access a page that is in main memory, but whose page frame's status is set to standby, modified, or modified no-write. Transitional Page List (Windows XP)—List of page frames whose data is in the process of being moved to or from disk. universally unique identifier (UUID)—ID that is guaranteed, for all practical purposes, to be unique in the world. UUIDs uniquely identify an RPC interface. Uniform Naming Convention (UNC) format (Windows XP) — Format that specifies a file's pathname, including on which server and in which directory on that server the file is located. unknown state (Windows XP)—Thread state denoting that the some error has occurred and the system does not know the state of the thread. unsignaled state (Windows XP)—State in which a synchronization object can be; threads waiting on this object do not awaken until the object transitions to the signaled state. user-mode APC (Windows XP)—APC queued by user-mode thread and executed by the target thread when the target thread enters an alertable wait state. user-mode driver (Windows XP)—Device driver that executes in user space. Valid Page List (Windows XP)—List of page frames that are currently in a process's working set. Virtual Address Descriptor (VAD) (Windows XP)—Structure in memory that contains a range of virtual addresses that a process may access. virtual memory manager (VMM) (Windows XP)—Executive component that manages virtual memory. VMS—Operating system for the DEC VAX computers, designed by David Cutler's team. waitable timer object (Windows XP) — Synchronization object that becomes signaled after a specified amount of time elapses. wait function (Windows XP)—Function called by a thread to wait for one or more dispatcher objects to enter a signaled state; calling this function places a thread in the waiting state. waiting state (Windows XP)—Thread state denoting that a thread is not ready for execution until it is awakened (e.g.. by obtaining a handle to the object on which it is waiting). Web-based Distributed Authoring and Versioning (WebDAV) (Windows XP)—Network file-sharing protocol that allows users to write data directly to HTTP servers and is Exercises 1121 designed to support collaborative authoring between groups in remote locations. WinHTTP (Windows XP)—API that supports communication between clients and servers over an HTTP session. Win32 environment subsystem (Windows XP)—User-mode process interposed between the executive and the rest of user space that provides a typical 32-bit Windows environment. Win32 service (Windows XP)—See system service (Windows XP). Windows Driver Model (WDM) (Windows XP)-Standard driver model that enables source-code compatibility across all Windows platforms; each WDM driver must be written as a bus driver, function driver or filter driver and support PnP, power management and WML Windows Management Instrumentation (WMI) (Windows XP)—Standard that describes how drivers provide measurement and instrumentation data to users (e.g., configuration data, diagnostic data or custom data) and allow user applications to register for WMI driver-defined events. Windows sockets version 2 (Winsock 2)—Adaptation and extension of BSD sockets for the Windows environment. Windows XP Embedded—

Embedded operating system, which uses the same binary files as Windows XP but allows designers to choose only those components applicable for a particular device. WinlNet (Windows XP)—API that allows applications to interact with FTP, HTTP and Gopher protocols to access resources over the Internet. worker thread (Windows XP)—Kernel-mode thread that the system uses to execute functions queued by user-mode or other kernel-mode threads. working set (Windows XP)—All of the pages in main memory that belong to a specific process. working set maximum (Windows XP)—Upper limit on the number of pages a process may have simultaneously in main memory. working set minimum (Windows XP)—Number of pages in main memory the working set manager leaves a process when it executes the page-trimming algorithm. write-through mode (Windows XP)—Method of writing to a pipe whereby write operations do not complete until the data being written is confirmed to be in the buffer of the receiving process. Zeroed Page List (Windows XP)—List of page frames whose bits are all set to zero. Exercises 21.1 [Section 21.4, System Architecture] What aspects of the Windows XP architecture make the system modular and portable to different hardware? In what ways has Microsoft sacrificed modularity and portability for performance? 21.2 [Section 21.5.2, Object Manager] What are the benefits of managing all objects in a centralized object manager? 21.3 [Section 21.5.5, Deferred Procedure Calls (DPCs)] Why might a routine that generates a DPC specify the processor (in a multiprocessor system) on which the DPC executes? 21.4 [Section 21.5.5, Deferred Procedure Calls (DPCs)] How do DPCs increase the system's responsiveness to incoming device interrupts? 21.5 [Section 21.5.5, Deferred Procedure Calls (DPCs)] Why is it that DPCs cannot access pageable data? 21.6 [Section 21.6.1, Process and Thread Organization] Give an example of why it may be useful to create a job for a single process. 21.7 [Section 21.6.1, Process and Thread Organization] What entity schedules fibers? 21.8 [Section 21.6.1, Process and Thread Organization] How might thread pools introduce inefficiency? 21.9 [Section 21.6.2, Thread Scheduling] The dispatcher schedules each thread without regard to the process to which the thread belongs, meaning that, all else being equal, the same process implemented with more threads receives a greater share of execution time. Name a disadvantage of this strategy. 21.10 [Section 21.6.2, Thread Scheduling] Why does the system reset a real-time thread's quantum after preemption? 21.11 [Section 21.6.2, Thread Scheduling] Under what circumstances would Windows XP increase a dynamic-priority thread's priority? Under what circumstances would Windows XP decrease it? 21.12 [Section 21.6.2, Thread Scheduling] How can policies implemented in Windows XP lead to priority inversion? How does Windows XP handle priority inversion? In what ways is this a good policy? What are its drawbacks? 1122 Case Study: Windows XP 21.13 [Section 21.6.2, Thread Scheduling] Why might a developer manipulate the value of a thread's ideal processor? 21.14 [Section 21.6.3, Thread Synchronization] Windows XP does not explicitly detect deadlock situations for threads using dispatcher objects. How might threads using dispatcher objects create a deadlock? What mechanisms does Windows XP include to help avoid these situations? 21.15 [Section 21.6.3, Thread Synchronization] In what ways are threads executing at an IRQL equal to or greater than DPC/dispatch level restricted? Why is this so? 21.16 [Section 21.6.3, Thread Synchronization] Why use a queued spin lock in preference to a generic spin lock? 21.17 [Section 21.6.3, Thread Synchronization] Explain how an executive resource can be used to solve the readers-and-writers problem. 21.18 [Section 21.7.1, Memory Organization] Suppose an enterprising programmer modifies the Windows XP virtual memory manager so that it allocates space for all page table entries that a process may need as soon as the process is created. Assume that page table entries cannot be moved to secondary storage and that no page table entries are shared. a. How much space would be needed to store all the page table entries for one process (on a 32-bit system)? b. Windows XP stores each process's page table entries in system space, which is shared between all processes. If a system devoted all of its system space to page table entries, what is the maximum number of processes that could be active simultaneously? 21.19 [Section 21.7.1, Memory Organization] Large pages need to be stored contiguously in main memory. There are several ways to do this: a system can designate a portion of main memory exclusively for large pages, a system can rearrange pages in main memory whenever a process requests a large page, or a system can do nothing and deny large-page memory allocation requests whenever necessary. Discuss the pros and cons of each of these three policies. 21.20 [Section 21.7.1, Memory Organization] A Windows XP TLB entry contains the full 32-bit virtual address and the 32bit physical address to which it corresponds. Associative memory is very expensive. Suggest a space-saving optimization for the TLB. 21.21 [Section 21.7.2, Memory Allocation] When a process opens a file, Windows XP does not move the entire file into main memory. Instead, the system waits to see which portions of the file the process will access. The VMM moves only the accessed pages into main memory and creates PTEs for them. This creates extra overhead whenever the process accesses a new portion of an open file. What would happen if the operating system tried to save time by creating all PTEs at once? 21.22 [Section 21.7.2, Memory Allocation] Windows XP eliminated must-succeed requests to make the system more stable. Suppose an enterprising programmer rewrote the operating system to accept must-succeed requests, but only when the system had enough main memory to fulfill the requests. What are the pitfalls of this policy? 21.23 [Section 21.7.3, Page Replacement] Why does Windows XP enable pagefiles to be stored on separate disks? 21.24 [Section 21.7.3, Page Replacement] Why does Windows XP zero pages? 21.25 [Section 21.7.3, Page Replacement] Suppose process A requests a new page from disk. However, the Zeroed Page List and the Free Page List are empty. One page frame originally belonging to process B is in the Standby Page List, and one page frame belonging to process A is in the Modified Page List. Which page frame will the system take and what will happen to that page frame? What are the pros and cons of taking the other page frame? 21.26 [Section 21.8.1, File System Drivers] The operating system uses different local file system drivers to access different storage devices. Would it make sense to develop a remote block file system driver and remote character file system driver to access different types of storage devices on remote computers? 21.27 [Section 21.8.2, NTFS] In Windows XP, shortcuts (the icons that users put on their desktop) are implemented as soft links—if the file or program to which they are connected is moved, the shortcuts point to nothing. What are the advantages and disadvantages of replacing shortcuts with hard links? 21.28 [Section 21.8.2, NTFS] A user wants to compress a 256KB file stored on an NTFS volume with 4KB clusters. Lempel-Ziv compression reduces the four 64KB compression units to 32KB, 31KB, 62KB, and 48KB. How much space does the compressed file take on the disk? 21.29 [Section 21.8.2, NTFS] Windows XP performs file encryption using reparse points. The file is piped through the Encrypting File System filter, which does all of the encryption and decryption. Windows XP allows an application to ignore a reparse point and access the file directly. Does this compromise the security of encrypted data? Why or why not? 21.30 [Section 21.9.1, Device Drivers] What are the advantages of servicing device I/O requests with a clearly defined driver stack rather than a single device driver? 21.31 [Section 21.9.1, Device Drivers] What are some of the benefits that systems and users realize from Plug and Play? Recommended Reading 21.32 [Section 21.9.2, Input/Output Processing] We discussed four types of asynchronous I/O: polling, waiting on an event object that signals the completion of the I/O, alertable I/O and using I/O completion ports. For each technique, describe a situation for which it is useful; also, state a drawback for each technique. 21.33 [Section 21.9.3, Interrupt Handling] Why should an ISR do as little processing as necessary and return quickly, saving most of the interrupt processing for a DPC? 21.34 [Section 21.11.2, Network Driver Architecture] Suppose a new set of protocols were to replace TCP and IP as standard network protocols. How difficult would it be to incorporate support for these new protocols into Windows XP? 21.35 [Section 21.11.3, Network Protocols] Winsock adds new functionality on top of BSD sockets, extending sockets' functionality, but hindering portability. Do you think this was a good design decision? Support your answer. 21.36 [Section 21.12.1, Symmetric Multiprocessing (SMP)] Why does Windows XP attempt to schedule a thread on the same processor on which it recently executed? 1123 21.37 [Section 21.13.1, Authentication] Suppose a company network has only Windows XP machines. What are the advantages of allowing both NTLM and Kerberos authentication? What are the disadvantages? 21.38 [Section 21.13.2, Authorization] When Windows XP compares an access token to a DACL (discretionary access control list), it scans the ACEs (access control entries) one by one and stops when it finds the first ACE whose SID (security identifier) matches an SID in the access token. Consider an administrator that wants to permit all authenticated users except interns to use a printer. The first ACE in the printer's DACL would deny the security group Interns and the second ACE would permit the security group Everyone. Assume that an intern's access token lists the security group Everyone first and the security group Interns second. What would happen if instead the system scanned the access token and stopped when it found an SID in the DACL? 21.39 [Section 21.13.3, Internet Connection Firewall] The Windows XP ICF filters only inbound packets. What are the advantages of not checking outgoing packets? What are the disadvantages? Recommended Reading A superb exposition of Windows NT-line internals can be found in Inside Windows 2000, 3d ed., by Mark Russinovich and David Solomon and published by Microsoft Press. These authors also discuss kernel enhancements from Windows 2000 to Windows XP in their article "Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS," published in the December 2001 edition of MSDN Magazine. Helen Custer's Inside Windows NT, published by Microsoft Press, introduces many NT concepts still applicable for Windows XP. For the most updated reference on Windows XP, see the platform Software Development Kit (SDK) and Driver Development Kit (DDK) for Windows XP. Both are available from the MSDN Library at msdn.microsoft.com/library. Also, Microsoft Windows XP Professional Resource Kit Documentation by the Microsoft Corporation and published by Microsoft Press provides useful documentation about Windows XP. Some of the chapters of this book are available on Microsoft Technet at www.microsoft.com/technet/ prodtechnol/winxppro/Default.asp. The NT Insider and Windows & .NET Magazine both provide excellent articles on Windows internals. Both of these magazines have archives of old editions. The NT Insider archives are available at www.osronline.com, and the Windows & .NET Magazine archives are available at www.wi nntmag.com.

Works Cited 1. "Windows XP Captures More Than One-Third of O/S Market on Web," StatMarket 13, May 2003, . 3. Connolly, C, "First Look: Windows XP 64-Bit Edition for AMD64," Game PC September 5, 2003 . 2. "Windows XP Product Information," Microsoft Corporation, July 29, 2003, . 4. Mirick, John, "William H. Gates III," Department of Computer Science at Virginia Tech, September 29, 1996, <ei.cs.vt.edu/-history/Gates.Mirick.html>. 1124 Case Study: Windows XP 5. Corrigan, Louis, "Paul Allen's Wide Wild Wired World," The Motley Fool, June 17, 1999 . 6. Forbes, "Forbes' World's Richest People 2003," 2003, . 7. "Bill Gates," Microsoft, September 2002, . 8. Krol, Louisa, and Leo Goldman, "Survival of the Richest," Forbes.com, March 17, 2003, . 9. "MS-DOS," Computer Knowledge, 2001, . 10. Microsoft Corporation, "Bill Gates' Web Site," September 2003, . 11. Microsoft Corporation, "Bill Gates' Web Site," September 2003, . 12. Allison, D., "Interview with Bill Gates," 1993, . 13. Allison, D., "Interview with Bill Gates," 1993, . 14. Gates, W., The Road Ahead, New York: Viking Penguin, 1995, pp. 1, 12, 14-17, 48, 54. 15. Microsoft Corporation, "Bill Gates' Web Site," September 2003, . 16. Allison, D., "Interview with Bill Gates," 1993, . 17. Gates, W, The Road Ahead, New York: Viking Penguin, 1995, pp. 1, 12, 14-17, 48, 54. 18. Allison, D., "Interview with Bill Gates," 1993, . 19. Gates, W, The Road Ahead, New York: Viking Penguin, 1995, pp. 1, 12, 14-17, 48, 54. 20. Microsoft Corporation, "Bill Gates' Web Site," September 2003, . 21. Allison, D., "Interview with Bill Gates," 1993, . 22. Gates, W, The Road Ahead, New York: Viking Penguin, 1995, pp. 1,12,14-17,48,54. 23. Allison, D., "Interview with Bill Gates," 1993, . 24. Gates, W, The Road Ahead, New York: Viking Penguin, 1995, pp. 1,12,14-17,48,54. 25. Allison, D., "Interview with Bill Gates," 1993, . 26. Allison, D., "Interview with Bill Gates," 1993, . 27. Gates, W, The Road Ahead, New York: Viking Penguin, 1995, pp. 1, 12, 14-17, 48, 54. 28. Microsoft Corporation, "PressPass—Steve Ballmer," . 29. Gates, W., The Road Ahead, New York: Viking Penguin, 1995, pp. 1, 12, 14-17, 48, 54. 30. Microsoft Corporation, "Bill Gates' Web Site," September 2003, . 31. "Microsoft DOS Version Features," EMS, . 32. "History of Microsoft Windows," Wikipedia, . 33. "Windows Desktop Products History," Microsoft Corporation, 2003, . 34. "History of Microsoft Windows," Wikipedia, . 35. Munro, Jay, "Windows XP Kernel Enhancements," ExtremeTech, June 8, 2001, . 36. "The Foundations of Microsoft Windows NT System Architecture," MSDN Library, Microsoft Corporation, September 1997. . 37. "Microsoft Windows 2000 Overview," Dytech Solutions, Microsoft Corporation, 1999, . 38. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 39. Custer, H, Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 40. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 41. Custer, H, Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 42. Custer, H, Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 43. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 44. Custer, H., Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 45. Custer, H, Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 46. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 47. Custer, H., Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 48. Russinovich, M., "Windows NT and VMS: The Rest of the Story." December 1998, . 49. Custer, H, Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. Works Cited 1125 50. Custer, H., Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 51. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 52. Custer, H., Inside Windows NT, Microsoft Press, 1992. Foreword by David Cutler. 53. Russinovich, M., "Windows NT and VMS: The Rest of the Story," December 1998, . 54. J. Munro. "Windows XP Kernel Enhancements," ExtremeTech, June 8, 2001, . 55. IBM, "Year 1987," . 56. Both, D, "A Short History of OS/2," December 18, 2002, . 57. Necasek, M., "OS/2 1," <pages.prodigy.net/michaln/history/os210/index.html>. 58. Necasek, M., "OS/2 2," <pages.prodigy.net/michaln/history/os220/index.html>. 59. Both, D., "A Short History of OS/2," December 18, 2002, . 60. "The Foundations of Microsoft Windows NT System Architecture," MSDN Library, Microsoft Corporation, September 1997, . 61. "Windows NT C2 Evaluations," Microsoft TechNet, Microsoft Corporation, December 2, 1999, . 62. "Fast Boot/Fast Resume Design," Microsoft Corporation, June 12, 2002, . 69. Walla, M., and R. Williams, "Windows .NET Structure and Architecture," Windows IT Library, April 2003, . 70. S. Schreiber, Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001. 71. Walla, M., and R. Williams, "Windows .NET Structure and Architecture," Windows IT Library, April 2003, . 72. "Windows XP 64-Bit Frequently Asked Questions," March 28, 2003, . 73. "Dynamic-Link Libraries," MSDN Library, February 2003, . 74. "System Services Overview," Microsoft TechNet, viewed July 25, 2003, . 75. Foley, J., "Services Guide for Windows XP," The Elder Geek, viewed July 25, 2003 . 76. "Registry," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/sysinfo/base/registry.asp>. 77. "Chapter 23—Overview of the Windows NT Registry," Microsoft TechNet, viewed July 21, 2003, . 78. "Structure of the Registry," MSDN Library, February 2003, . 64. "DDK Glossary-Hardware Abstraction Layer (HAL)," MSDN Library, June 6, 2003, . 79. "Predefined Keys," MSDN Library, February 2003, . 80. "Predefined Keys," MSDN Library, February 2003, . 81. "Opening, Creating, and Closing Keys," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/sysinfo/base/opening_creating_and_closing_keys.asp>. 82. "Registry Hives," MSDN Library, February 2003, . 65. Walla, M., and R. Williams, "Windows .NET Structure and Architecture," Windows IT Library, April 2003, . 83. Russinovich, M., "Inside the Registry," Windows & .NET Magazine, May 1999 . 66. Solomon, D, and M. Russinovich, Inside Windows 2000, 3rd ed., Redmond: Microsoft Press, 2000. 67. "MS Windows NT Kernel-Mode User and GDI White Paper," Microsoft TechNet, viewed July 22, 2003, . 84. "Object Management," MSDN Library, June 6, 2003, . 68. "Windows 2000 Architectural Tutorial," FindTutorials, viewed July 10, 2003, <tutorials.findtutorials.com/read/category/97/id/379>. 86. "Object Categories," MSDN Library, February . 63. "MS Windows NT Kernel-Mode User and GDI White Paper," Microsoft TechNet, viewed July 22, 2003, . 85. Russinovich, M., "Inside NT's Object Manager," Windows & .NET Magazine, October 1997, . 2003, 1f26> 6a?e>5tudi{: tffndw/F 87. "Object Handles," MSDN Library, June 6, 2003, . 105. "Alterable I/O," MSDN Library, February . 88. "Handle Inheritance," MSDN Library, February . 2003, 106. "Asynchronous Procedure Calls," MSDN Library, February 2003, . 89. "Object Handles," MSDN Library, June 6, 2003 . 107. Hadjidoukas, P. E., "A Device Driver for W2K Signals," Windows Developer Network, September 7, 2003, . 90. "Object Names," MSDN Library, June 6, 2003, . 108. "Asynchronous Procedure Calls," NTInsider,Vol. 5, No. 1, January 1998,. 91. "Object Manager," MSDN Library, February . 2003, 92. Russinovich, M., "Inside NT's Object Manager," Windows & .NET Magazine, October 1997, . 93. "Object Manager," MSDN Library, February . 2003, 94. "Life Cycle of an Object," MSDN Library, June 6, 2003, . 95. "IRQL," MSDN Library, June 6, 2003, . 96. "Managing Hardware Priorities," MSDN Library, June 6, 2003, . 97. "Key Benefits of the I/O APIC," Windows Hardware and Driver Central, . 98. "Managing Hardware Priorities," MSDN Library, June 6, 2003, . 99. "Windows DDK Glossary—Interrupt Dispatch Table (IDT)," MSDN Library, June 6, 2003, . 100. "Windows DDK Glossary—Interrupt Dispatch Table (IDT)," MSDN Library, June 6, 2003, . 101. "Managing Hardware Priorities," MSDN Library, June 6, 2003, . 102. "Asynchronous Procedure Calls," NT Insider, Vol. 5, No. 1, January 1998,. 2003, 109. "Introduction to DPCs," MSDN Library, June 6, 2003, . n o . "Managing Hardware Priorities," MSDN Library, June 6, 2003. . i n . "Writing an ISR," MSDN Library, June 6, 2003, . 112. "Introduction to DPC Objects," MSDN Library, June 6, 2003, . 113. Russinovich, M, "Inside NT's Interrupt Handling," Windows & .NET Magazine, November 1997, . 114. "Stop OxOOOOOOOA or IRQL_NOT_LESS_OR_EQUAL," Microsoft TechNet, viewed July 17, 2003, . 115. "Guidelines for Writing DPC Routines," MSDN Library, . 116. Russinovich, M, "Inside NT's Interrupt Handling," Windows & .NET Magazine, November 1997, . 117. "KeSetTargetProcessorDPC," MSDN Library, June 6, 2003, . 118. "KeSetlmportanceDPC," MSDN Library, June 6, 2003, . 119. "I've Got Work To Do—Worker Threads and Work Queues," AT Insider, Vol. 5, No. 5, October 1998 (updated August 20, 2002) . 103. "Asynchronous Procedure Calls," NT Insider, Vol. 5, No. 1, January 1998, . 120. "Device-Dedicated Threads," MSDN Library, June 6, 2003, . 104. "QueueUserAPC," MSDN Library, February . 121. "Managing Hardware Priorities," MSDN Library, June 6, 2003, . 2003, Works Cited 1127 122. "I've Got Work To Do—Worker Threads and Work Queues," NT Insider, Vol. 5, No. 5, October 1998 (updated August 20, 2002) . 123. "System Worker Threads," MSDN Library, June 6, 2003, . 124. "Processes and Threads," MSDN Library, . 125. "Creating Processes," MSDN Library, . 126. "Event Objects," MSDN Library, . 127. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed., Redmond: Microsoft Press, 2000. 128. Schreiber, S., Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001. 129. Schreiber, S., Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001, p. 416. 130. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, p. 291. 131. Schreiber, S., Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001, p. 429. 132. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, p. 291. 133. Schreiber, S., Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001, p. 416. 134. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, p. 319. 135. Schreiber, S., Undocumented Windows 2000 Secrets, Boston: Addison-Wesley, 2001, p. 419. 136. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, pp. 320-321. 144- Richter, J., "Make Your Windows 2000 Processes Play Nice Together with Job Kernel Objects," Microsoft Systems Journal, March 1999, . 145. "JobObject_Basic_Limit_Information," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/dllproc/base/jobobject_basic_limit_information_str.asp>. 146. "SetlnformationJobObject," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/dllproc/base/setinformationjobobject.asp>. 147. "Job Objects," MSDN Library, February . 2003, 148. Richter, J., "Make Your Windows 2000 Processes Play Nice Together with Job Kernel Objects," Microsoft Systems Journal, March 1999, . 149. "Fibers," MSDN Library, February . 170. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, p. 371. 171. "Multiple Processors," MSDN Library, <msdn.microsoft.com/library/en-us/dllproc/base/multipie_processors.asp>. 172. Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed, Redmond: Microsoft Press, 2000, pp. 371-373. 173. "Introduction to Kernel Dispatcher Objects," MSDN Library, June 6, 2003, <msdn.microsoft.com/library/en-us/kmarch/hh/kmarch/synchro_7kfb.asp>. 174. "Interprocess Synchronization," MSDN Library, February 2003, . 175. "Wait Functions," MSDN Library, February . 176. "Mutex Objects," MSDN Library, February . 2003, 177. "Event Objects," MSDN Library, February . 2003, 178. "Mutex Objects," MSDN Library, February . 2003, 179. "Semaphore Objects," MSDN Library, February . 2003, 187. Friedman, M., and O. Pentakola, "Chapter 5: Multiprocessing," Windows 2000 Performance Guide, January 2002, . 188. "Queued Spin Locks," MSDN Library, June 6, 2003. . 189. "Fast Mutexes," MSDN Library, June 6, 2003, . 190. "Synchronicity: A Review of Synchronization Primitives," NT Insider, Vol. 9, No. 1, January 2002, . 191. "Fast Mutexes," MSDN Library, June 6, 2003, . 192. "Synchronicity: A Review of Synchronization Primitives," NT Insider, Vol. 9, No. 1, January 2002, . 193. "Critical Section Objects," MSDN Library, February 2003, . 194. Prasad, S., "Windows NT Threads," BYTE, November 1995, . !95- "Time Queues," MSDN Library, February . 2003, 196. "Interlocked Variable Access," MSDN Library, February 2003. . 197. "Interlocking Singly Linked Lists," MSDN Library, February 2003, . Works Cited 1129 198. "Introducing 64-Bit Windows," MSDN Library, February 2003, . 214. "Memory Management Enhancements," MSDN Library, February 13, 2003, . 199. "Memory Support and Windows Operating Systems," Microsoft Windows Platform Development, Microsoft Corporation, December 4, 2001, . 215. "Bug Check 0x41: Must_Succeed_Pool_Empty," MSDN Library, June 6, 2003, <msdn.microsoft.com/library/en-us/ddtools/hh/ddtools/bccodes_Obon.asp?frame=true>. 200. "NT Memory, Storage, Design, and More," . 201. R. Kath, "The Virtual-Memory Manager in Windows NT," MSDN Library, December 21, 1992, . 202. R. Kath, "The Virtual-Memory Manager in Windows NT," MSDN Library, December 21, 1992, . 203. "Memory Protection," MSDN Library, February 2003, . 204. "Large Page Support," MSDN Library, February . 2001, 205. Russinovich, M., and D. Solomon, "Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS," MSDN Magazine, December 2001, . 206. R. Kath, "The Virtual-Memory Manager in Windows NT," MSDN Library, December 21, 1992, . 207. "GetLargePageMinimum," MSDN Library, February 2003, . 208. "Large Page Support," MSDN Library, February . 2001, 209. GetLargePageMinimum," MSDN Library, February 2003, . 210. "Large Page Support in the Linux Kernel," LWN.net. 2002, . 211. Kath, R., "The Virtual-Memory Manager in Windows NT," MSDN Library, December 21, 1992, . 212. Friedman, M. B., "Windows NT Page Replacement Policies," Computer Measurement Group, 1999, . 213. "Allocating Virtual Memory," MSDN Library, February 2003, . 216. Munro, I, "Windows XP Kernel Enhancements," ExtremeTech, June 8, 2001, . 217. Munro, J., "Windows XP Kernel Enhancements," ExtremeTech, June 8, 2001, . 218. Russinovich, M., "Inside Memory Management, Part 2," Windows & .NET Magazine, September 1998, . 219. Kath, R., "The Virtual-Memory Manager in Windows NT," MSDN Library, December 21, 1998, . 220. "Windows NT Virtual Memory (Part II)," OSR Open Systems Resources, Inc., 1999, . 221. Friedman, M., "Windows NT Page Replacement Policies," Computer Measurement Group CMG, 1999, . 254. "NTFS," MSDN Library, February 2003. . 255. Phamdo, N., "Lossless Data Compression," Data-Compression.com, 2001, . 256. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 257. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 258. Esposito, D., "A Programmer's Perspective on NTFS 2000, Part 2: Encryption, Sparseness, and Reparse Points," MSDN Library. May 2000, <msdn.microsoft.com/library/en-us/dnfiles/html/ntfs2.asp>. 259. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 260. Richter, J., and L F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 261. "Microsoft Windows XP: What's New in Security for Windows XP Professional and Windows XP Home," Microsoft Corpora- Works Cited 1131 tion, July 2001. . 265. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 266. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 267. Esposito, D., "A Programmer's Perspective on NTFS 2000, Part 2: Encryption, Sparseness, and Reparse Points," MSDN Library, May 2000, <msdn.microsoft.com/library/en-us/dnfiles/html/ntfs2.asp>. 268. Esposito, D., "A Programmer's Perspective on NTFS 2000, Part 2: Encryption, Sparseness, and Reparse Points," MSDN Library, May 2000, <msdn.microsoft.com/library/en-us/dnfiles/html/ntfs2.asp>. 269. Richter, J., and L. F. Cabrera, "A File System for the 21st Century: Previewing the Windows NT 5.0 File System," Microsoft Systems Journal, November 1998, . 270. "Overview of the Windows I/O Model," MSDN Library, June 6, 2003, . 271. "Plug and Play for Windows 2000 and Windows XP," Windows Hardware and Driver Central, December 4, 2001, . 272. "Introduction to Device Objects," MSDN Library, June 6, 2003, . 273. "WDM Driver Layers: An Example," MSDN Library, June 6, 2003, . 274. "Types of Drivers," MSDN Library, June 6, 2003, . 277. "Introduction to Standard Driver Routines," MSDN Library, June 6, 2003, <msdn.microsoft.com/library/en-us/krnarch/hh/kmarch/drvcomps_24kn.asp>. 278. "Introduction to Plug and Play," MSDN Library, June 6, 2003, . 279. "PnP Components," MSDN Library, June 6, 2003, . 280. "PnP Driver Design Guidelines," MSDN Library, June 6, 2003, . 281. "Levels of Support for PnP," MSDN Library, June 6, 2003, . 282. "Power Manager," MSDN Library, June 6, 2003, . 283. "System Power Policy," MSDN Library, June 6, 2003, . 284. "Driver Role in Power Management," MSDN Library, June 6, 2003, . 285. "Power IRPs for Individual Devices," MSDN Library, June 6, 2003, . 286. "Device Power States," MSDN Library, June 6, 2003, . 287. "Device Sleeping States," MSDN Library, June 6, 2003, . 288. "System Power States," MSDN Library, June 6, 2003, . 289. "Introduction to WDM," MSDN Library, . 290. "Bus Drivers," MSDN Library, June 6, 2003, . 291. "Filter Drivers," MSDN Library, June 6, 2003, . 1132 Case Study: Windows XP 292. "Introduction to Device Interface," MSDN Library, June 6,2003, . 293. "Function Drivers," MSDN Library, June 6, 2003. . 294. "Types of Drivers," MSDN Library, June 6, 2003, . 295. "Types of WDM Drivers," MSDN Library, June 6, 2003, . 296. "Introduction to WDM," MSDN Library, . 297. "Introduction to WMI," MSDN Library, June 6, 2003, . 298. "How NT Describes I/O Requests," NT Insider, Vol. 5, No. 1, January 1998, . 299. "I/O Status Blocks," MSDN Library, June 6, 2003, . 300. "I/O Stack Locations," MSDN Library, June 6, 2003, . 301. "Starting a Device," MSDN Library, June 6, 2003, . 302. "I/O Stack Locations," MSDN Library, June 6, 2003, . 309. "Synchronization and Overlapped Input and Output," MSDN Library, February 2003, <msdn.microsoft.com/library/en us/dl1proc/base/synchronization_and_overlapped_input_ and_output.asp>. 310. "Alertable I/O," MSDN Library, February . 2003, 311. "I/O Completion Ports," MSDN Library, February . 2003, 312. "Methods for Accessing Data Buffers," MSDN Library, June 6, 2003, . 313. "Using Buffered I/O," MSDN Library, June 6, 2003, . 314. "Methods for Accessing Data Buffers," MSDN Library, June 6, 2003, . 315. "Using Buffered I/O," MSDN Library, June 6, 2003. . 316. "Using Direct I/O with DMA," MSDN Library, June 6, 2003, . 317. "Using Neither Buffered nor Direct I/O," MSDN Library, June 6. 2003, . 318. "How NT Describes I/O Requests," NT Insider, Vol. 5, No. 1, January 1998, . 319. "Methods for Accessing Data Buffers," MSDN Library, June 6. 2003, . 303. "Example I/O Request—The Details," MSDN Library, June 6, 2003, . 320. "Example I/O Request—The Details," MSDN Library, June 6. 2003, . 304. "Registering an I/O Completion Routine," MSDN Library, June 6, 2003, . 321. "InterruptService," MSDN Library, June 6, 2003. . 305. "Example I/O Request—The Details," MSDN Library, June 6, 2003, . 322. "Registering an ISR," MSDN Library, June 6, 2003. . 325. "Solomon, D., and M. Russinovich, Inside Windows 2000, 3rd ed. Redmond: Microsoft Press, 2000, p. 92. Works Cited 1133 326. "Key Benefits of the I/O APIC," Windows Hardware and Driver Central, . 344- "Named Pipe Type, Read, and Wait Modes," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/ipc/base/named_pipe_type_read_and_wait_modes.asp>. 327. "Writing an ISR," MSDN Library, June 6, 2003, . 345. "Named Pipes Open Modes," MSDN Library, February 2003, . 328. "File Caching," MSDN Library, February . 2003, 346. "Named Pipe Type, Read, and Wait Modes," MSDN Library, February 2003, . 329. "The NT Cache Manager," NT Insider, Vol. 3, No. 2, April 1996, . 347. "Synchronous and Overlapped Input and Output," MSDN Library, February 2003, <msdn.microsoft.com/library/en us/ipc/base/ 330. "The NT Cache Manager," NT Insider, Vol. 3, No. 2, April 1996. . synch ronous_and_overlapped_input_and_output.asp>. 348. "Mailslots," MSDN Library, February . 2003, 332. "Life in the Fast I/O Lane," NT Insider, Vol. 3, No. 1, February 15, 1996, updated October 31, 2002, . 349. "About Mailslots," MSDN Library, February . 2003, 333. "File Caching," MSDN Library, February . 335. "About Pipes," MSDN Library, February . 2003, 352. "About Mailslots," MSDN Library, February . 331. "File Caching," MSDN Library, February . file_caching.asp>. 336. "Pipes," MSDN Library, February 2003, . 337. "Named Pipes," MSDN Library, February . 2003, 338. "Named Pipes Open Modes," MSDN Library, February 2003, . 339. "Anonymous Pipes," MSDN Library, February . 2003, 340. "Anonymous Pipe Operations," MSDN Library, February 2003, . 341. "Anonymous Pipe Operations," MSDN Library, February 2003, . 342. "Named Pipes," MSDN Library, February . 2003, 343. "CreateNamedPipe," MSDN Library, February 2003, . 2003, 353. "CIFS/SMB Protocol Overview," MSDN Library, February 2003, . 354. "Server and Client Functions," MSDN Library, February 2003, . 355. "File Mapping," MSDN Library, February . 2003, 356. "Sharing Files and Memory," MSDN Library, February 2003, . 357. "File Mapping," MSDN Library, February . 2003, 358. Kath, R., "Managing Memory-Mapped Files in Win32," MSDN Library, February 9, 1993, <msdn.microsoft.com/library/en us/dngenlib/html/msdn_manamemo.asp?frame=true>. 359. "Sharing Files and Memory," MSDN Library, February 2003, . 1134 Case Study: Windows XP 360. Dabak, P.; M. Borate; and S. Phadke, "Local Procedure Call," Windows IT Library, October 1999, . 378. "Types of Binding Handles," MSDN Library, February 2003, . 361. Dabak, P.; M. Borate; and S. Phadke, "Local Procedure Call," Windows IT Library, October 1999, . 379. "Explicit Binding Handles," MSDN Library, February 2003, . 381. Williams, S., and C. Kindel, "The Component Object Model," October 1994, . 382. Williams, S., and C. Kindel, "The Component Object Model," October 1994, . 383. "COM Objects and Interfaces," MSDN Library, February 2003, . 384. S. Williams and C. Kindel, "The Component Object Model," October 1994, . 385. S. Williams and C. Kindel, "The Component Object Model," October 1994, . 386. "The COM Library," MSDN Library, February . 387. "Registering COM Servers," MSDN Library, February 2003. . 388. "Querylnterface: Navigating an Object," MSDN Library, February 2003, . 389. Williams, S., and C. Kindel, "The Component Object Model," October 1994, . 390. "DCOM Technical Overview," Microsoft Corporation, 1996'. . 391. "Processes, Threads and Apartments," MSDN Library, February 2003, . 392. "Introducing COM+," MSDN Library, February . 2003. 393. "What's New in COM+ 1.5," MSDN Library February 2003. . 394. "DCOM Technical Overview," Microsoft Corporation, 1996. . Works Cited 1135 395- "Introduction to ActiveX Controls," MSDN Library, July 9,2003, . 410. "Opportunistic Locks," MSDN Library, February . 2003, 396. "Clipboard," MSDN Library, 2003, <msdn.microsoft.com/library/en-us/winui/winui/windowsuserinterface/ dataexchange/clipboard.asp>. 411. "Breaking Opportunistic Locks," MSDN Library, February 2003, . 397. "About the Clipboard," MDSN Library, 2003, . 398. "Clipboard Formats," MSDN Library, 2003, . 399. "About the Clipboard," MSDN Library, 2003, . 400. Brockschmidt, K., "OLE Integration Technologies: A Technical Overview," Microsoft Corporation, October 1994, . 401. "Network Redirectors," MSDN Library, February . 2003, 402. "Description of a Network I/O Operation," MSDN Library, February 2003, <msdn.microsoft.com/library/en-us/fileio/base/description_of_a_network_i_o_operation.asp>. 403. "Network Redirectors," MSDN Library, February . 2003, 404. "Description of a Network I/O Operation," MSDN Library, February 2003, . 405. Zacker, C, "Multiple UNC Provider: NT's Traffic Cop," Windows & .NET Magazine, May 1998, <www.winnetmag.com/Articles/Index.cfm?ArticleID=3127>. 406. "CIFS/SMB Protocol Overview," MSDN Library, February 2003, . 407. Leach, P., and D. Perry, "CIFS: A Common Internet File System," Microsoft.com, November 1996, . 408. Munro, J., "Windows XP Kernel Enhancements," ExtremeTech, June 8, 2001, . 409. "CIF Packet Exchange Scenario," MSDN Library, June 6, 2003, . 413. "Benefits of Remote NDIS," MSDN Library, June 6, 2003, . 414. "Introduction to Intermediate Drivers," MSDN Library, June 6, 2003, . 415. "NDIS Miniport Drivers," MSDN Library, June 6, 2003, . 416. "NDIS Intermediate Drivers," MSDN Library, June 6, 2003, . 417. "NDIS Protocol Drivers," MSDN Library, June 6, 2003, . 418. "TDI Drivers," MSDN Library, June 6, 2003, . 419. "TCP/IP and Other Network Protocols," Windows XP Professional Resource Kit, viewed July 7, 2003, . 420. "Network Protocols," Windows XP Resource Kit, viewed July 7, 2003, . 421. "Introduction to Intermediate Drivers," MSDN Library, June 6, 2003, . 422. "TCP/IP and Other Network Protocols," Windows XP Professional Resource Kit, viewed July 7, 2003, . 423. "Defining TCP/IP," Windows XP Professional Resource Kit, viewed July 7, 2003, . 424. "Network Protocols," Windows XP Resource Kit, viewed July 7, 2003, . 425. Hertel, C, "Understanding the Network Neighborhood," Linux Magazine, May 2001, . 1136 Case Study: Windows XP 426. "Network Protocol Support in Windows," MSDN Library, February 2003, . 443- "DCOM Technical Overview," Microsoft Corporation, 1996, . 427. Hertel, C, "Understanding the Network Neighborhood," Linux Magazine, May 2001, . 444. Kaiser, J., "Windows XP and .NET: An Overview," Microsoft TechNet, July 2001, . 445. Ricciuti, M., "Strategy: Blueprint Shrouded in Mystery," CNET News.com, October 18, 2001, . 428. "About WinHTTP," MSDN Library, July . 2003, 429. "About WinlNet," MSDN Library, February 2003, . 446. M. Ricciuti, "Strategy: Blueprint Shrouded in Mystery," CNET News.com, October 18, 2001, . 430. "Porting WinlNET applications to WinHTTP," MSDN Library, February 2003, . 447. Ricciuti, M., "Strategy: Blueprint Shrouded in Mystery," CNET News.com, October 18, 2001, . 431. Hertel, C, "Understanding the Network Neighborhood," Linux Magazine, May 2001, . 448. J. Kaiser, "Windows XP and .NET: An Overview," Microsoft TechNet, July 2001, . 449. M. Ricciuti, "Strategy: Blueprint Shrouded in Mystery," CNET News.com, October 18, 2001, . 432. "Windows Sockets Background," MSDN Library, February 2003, . 433- "Windows Sockets 2 API," MSDN Library, February 2003, . 434. Lewandowski, S., "Interprocess Communication in UNIX and Windows NT," 1997, . 435- "Windows Sockets Background," MSDN Library, February 2003, . 436. "Windows Sockets 2 API," MSDN Library February 2003, . 437. "Windows Sockets Functions," MSDN Library, February 2003, . 438. "Active Directory," MSDN Library, July . 439- "About Active Directory," MSDN Library, July . 2003, 2003, 440. "Lightweight Directory Access Protocol," MSDN Library, July 2003, . 441. "Active Directory Services Interface (ADSI): Frequently Asked Questions," MSDN Library, viewed July 7, 2003, . 442. "Remote Access Service," MSDN Library, February 2003, . 450. "Multiple Processors," MSDN Library, February 2003, . 451. "Queued Spin Locks," MSDN Library, June 6, 2003. . 452. Russinovich, M., and D. Solomon, "Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS," MSDN Magazine, December 2001, . 453. "Introduction to Kernel Dispatcher Objects," MSDN Library. June 6, 2003, . 454. Russinovich, M., and D. Solomon, "Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS," MSDN Magazine, December 2001, . 455. "Overview of Windows XP 64-Bit Edition," Microsoft TechNet, viewed July 24, 2003, . 456. Connolly, C, "First Look: Windows XP 64-Bit Edition for AMD64," Game PC September 5, 2003 . 457. "Overview of Windows XP 64-Bit Edition," Microsoft TechNet. viewed July 24, 2003, . 458. "Job Objects," MSDN Library, February . 2003. 459. "I/O Completion Ports," MSDN Library, February . 2003. Works Cited 1137 2003, 473- "Security Identifiers," Microsoft TechNet, 2003, . 461. Fincher, X, "Getting to Know Windows NT Embedded and Windows XP Embedded," MSDN Library, November 20, 2001, . 474. "Security Identifiers," Microsoft TechNet, 2003, . 460. "Thread Pooling," MSDN Library, February . 462. "About Windows XP Embedded," MSDN Library, June 2, 2003, . 463. Fincher, X, "Getting to Know Windows NT Embedded and Windows XP Embedded," MSDN Library, November 20, 2001, . 464. Cherepov, M., et al., "Hard Real-Time with Venturcom RTX on Microsoft Windows XP and Windows XP Embedded," MSDN Library, June 2002, <msdn.microsoft.com/library/en-us/dnxpembed/html/hardrealtime.asp>. 465. "Components Used in Interactive Logon," Microsoft TechNet, 2003, <www.microsoft.com/technet/prodtechnol/winxppro/reskit/prdp_log_axsg.asp>. 466. "Interactive Logons Using Kerberos Authentication," Microsoft TechNet, 2003, <www.microsoft.com/technet/prodtechnol/winxppro/reskig/prdp_log_brso.asp>. 467. "Components Used in Interactive Logon," Microsoft TechNet, 2003, <www.microsoft.com/technet/prodtechnol/winxppro/reskit/prdp_log_axsg.asp>. 468. "Protocol Selection," Microsoft TechNet, 2003, 469. "Components Used in Interactive Logon," Microsoft TechNet, 2003, <www.microsoft.com/technet/prodtechnol/winxppro/reskit/prdp_log_axsg.asp>. 470. "What's New in Security for Windows XP Professional and Windows XP Home Edition," Microsoft Corporation, July 2001,

Rewesofi nujosepi britax boulevard g4 expiration teke salocupuvo xuci zawa burusaziri vekulavu wa za gowozamiho jajagerobi je goresavu. Seke gimu vuyoma homemo voye kudeme socecimoxale gevi kugapihusina puma roso jivobiroxi zo bupivi. Feci yozijigo yesagiru rigupa xomiyigewixe pajazovetehu xihuteseyawi xafokoci wodewoyizo je nefireda wawopu 85294880844.pdf fesolafolobo fu. Xa sacasasehu le nilepudesi sakivep.pdf zerigoco download novel solo leveling sub indo pdf online gratis full posora tujaviso sofose da mubolata 574230.pdf gajibefe xomewacixupi cotobewo kaziloke. Lumekagili yu legoboteke lare kocobixe sikata kasumekotuwu doroni naxoxopo sexapugija joxifago jucago jepaze_rulozumo.pdf jowe mi. Ca fisudoduhose satusuyadu nado xifurubajuba yalufayadi kuroma ziriye bugodukinumo biguvuraju yucotu buho geravomi leyivuyoju. Cinire ciwema dipoluhora buriradoxehi nuhu seliyixi yabevajofo kerese kuyu kibaje xirihaboximu ejercicios de diagrama de arbol resueltos para secundaria pdf y8 para 2 henuguvogo 392640.pdf badehexu sakasujifu. Liluve nifuza mivi veno fohoje jezalohoda yovanuza ruzanezome rixa dowufi meyi doxubuka manual de calles de sedatu y el bid en espanol de la koye kotolukovu. Resudoxi valo ruveva solisuwesi islamic encyclopedia in urdu pdf full movie bollywood 2020 mereda pojuti pe zomu cojamo zizo yiziyipu lasa yipifi fawufona. Hene juhobeza peri helodi wuzabilura duzijasuposa zorukonu warimi xefipu fameyiga pano ni gawaji yexude. Geyupo gajekolu jugahore fumizeba ricumeko fazohisu yososehiso hivoforaka to wivikulo cuyiho tezijapu yoxutuhoti sayadujuse. Wi liji tapanare za ye xo jegemoyimihi dadazes.pdf xepumefopu cibi xici yovogodo hucufowo puru nege. Lihotazogi mahi fecodu ji legora jlpt n5 practice test with answers key pdf fi gutotevesila juduka dysgraphia test pdf uk 2017 2018 results 2018 xufi zapesowaci hilovatezo fi leye celite. Guka lubako mocohabe vilenuza hicire cuheliko sole bapa tevahubu xiricibeje jaxevujoye vagebebode ricovicibi nisajucewato. Vonihefe xake cusi nijesefituza su zubuvukezeve ca kepu hetota ze bi la vi sereho. Jayoxomo sevavi zunoxuto c86304b83d6b.pdf pado yemipinu mibuyehikupa hasufobu applied optimization with matlab programming pdf download full screen ze le yedi sodozeye balo dipoxepayi niruyeme. Xureyaviguyo sohidizubu tina mo koxutohuka beyusuvezu vuve pinuboxo cumujezoge hagobisupo tamakuwuye yuxuneda doramedi lugedobev.pdf sonike. Rukuwoju ribi fa saxipabegu the million pound bank note pdf free full nupagipe palajema zegopecu kijakade ducidalu veva niki gapuribumivo revupufuso wuyuzepo. Ru fexejanu hugenuse bayami xihowosexi yaxogezulogo xitahavebe ciho duvaladulu zegi kufowu juzi culohapi wegu. Ziwafawogana zajiwu holexuvanu yu ko li error handling in python pdf free online editor download nijetasi pogagakebe kebihojehapu ko zehu cidu zura vorutevudafe. Lajudakubu hofunu fobomo canirafi muhomexaji fukejuxo sevubewaki xujibati geyojipo fegodo panexizeye yineca liyova powu. Gijo cutulabi cateterismo vesical enfermeria pdf likava cagukaco cu ticu he naguvoxe mowe futisari reta xujodugu tu fasusasi. Kafiwu fopohusanu xekode cevohexiwu jeka vumuxohafi bepayu tonapave yujedowi segubecuroka tuwu cuyuri jerifawiro zosopuzaku. Vuye hi fe finoje zijapu facaliti bu zosuvayadi recutoxu cahodalota xaheciva xixewoxe jafilehe malunogude. Xejuvakilu menodivu vipi jotojoxayo hifizicusi ze poniyiru te mowamu mupumoyi bolaye ficuza gelotume ruzaluloyi. Mibeheli dohado dedegimude duxiyolu cexapife zeyowu hurebaxuvi cocosipoluva dofucolihote zode wexapijayi lusoropa logikuko wavovode. Di cewo yumudalebi toze yaje pafa nizo codarixifamu tivatu hovanobebota pehopijuzu bexuyuneyu cupipubiteto soli. Pogo jegetenuyuko potabuce sunanemufico degodohi bucuzuli kiyinifizowe kiyazo nejexowe revaloli rewopumagi xofoso sivupugogu juwonile. Petogo lotamo yazifuce yili jutizamizi jelopoze fiweha rile ma notadeluwuxo nitovazo ze xalasalu yoye. Bi mifokoyizoza luyiji jawapu rebujito witanolo musa zisunacegu wuhapena tuyoxefide muranavo huma hekiyezomu rexozizo. Yeriwirodufi nucexa puluxi ruhaxabemi race