Preface

Operating systems are an essential part of any computer system. Similarly, a course on operating systems is an essential part of any computer-science education. This field is undergoing rapid change, as computers are now prevalent in virtually every application, from games for children through the most sophisticated planning tools for governments and multinational firms. Yet the fundamental concepts remain fairly clear, and it is on these that we base this book.

We wrote this book as a text for an introductory course in operating systems at the junior or senior undergraduate level or at the first-year graduate level. We hope that practitioners will also find it useful. It provides a clear description of the *concepts* that underlie operating systems. As prerequisites, we assume that the reader is familiar with basic data structures, computer organization, and a high-level language, such as C or Java. The hardware topics required for an understanding of operating systems are included in Chapter 1. For code examples, we use predominantly C, with some Java, but the reader can still understand the algorithms without a thorough knowledge of these languages.

Concepts are presented using intuitive descriptions. Important theoretical results are covered, but formal proofs are omitted. The bibliographical notes at the end of each chapter contain pointers to research papers in which results were first presented and proved, as well as references to material for further reading. In place of proofs, figures and examples are used to suggest why we should expect the result in question to be true.

The fundamental concepts and algorithms covered in the book are often based on those used in existing commercial operating systems. Our aim is to present these concepts and algorithms in a general setting that is not tied to one particular operating system. We present a large number of examples that pertain to the most popular and the most innovative operating systems, including Sun Microsystems' Solaris; Linux; Microsoft Windows Vista, Windows 2000, and Windows XP; and Apple Mac OS X. When we refer to Windows XP as an example operating system, we are implying Windows Vista, Windows XP, and Windows 2000. If a feature exists in a specific release, we state this explicitly.

Organization of This Book

The organization of this text reflects our many years of teaching courses on operating systems. Consideration was also given to the feedback provided by the reviewers of the text, as well as comments submitted by readers of earlier editions. In addition, the content of the text corresponds to the suggestions from *Computing Curricula 2005* for teaching operating systems, published by the Joint Task Force of the IEEE Computing Society and the Association for Computing Machinery (ACM).

On the supporting Web site for this text, we provide several sample syllabi that suggest various approaches for using the text in both introductory and advanced courses. As a general rule, we encourage readers to progress sequentially through the chapters, as this strategy provides the most thorough study of operating systems. However, by using the sample syllabi, a reader can select a different ordering of chapters (or subsections of chapters).

On-line support for the text is provided by WileyPLUS. On this site, students can find sample exercises and programming problems, and instructors can assign and grade problems. In addition, in WileyPLUS, students can access new operating-system simulators, which are used to work through exercises and hands-on lab activities. References to the simulators and associated activities appear at the ends of several chapters in the text.

Content of This Book

The text is organized in eight major parts:

- **Overview**. Chapters 1 and 2 explain what operating systems *are,* what they *do,* and how they are *designed* and *constructed.* These chapters discuss what the common features of an operating system are, what an operating system does for the user, and what it does for the computer-system operator. The presentation is motivational and explanatory in nature. We have avoided a discussion of how things are done internally in these chapters. Therefore, they are suitable for individual readers or for students in lower-level classes who want to learn what an operating system is without getting into the details of the internal algorithms.
- **Process management**. Chapters 3 through 7 describe the process concept and concurrency as the heart of modern operating systems. A *process* is the unit of work in a system. Such a system consists of a collection of *concurrently* executing processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes (those that execute user code). These chapters cover methods for process scheduling, interprocess communication, process synchronization, and deadlock handling. Also included is a discussion of threads, as well as an examination of issues related to multicore systems.
- **Memory management**. Chapters 8 and 9 deal with the management of main memory during the execution of a process. To improve both the utilization of the CPU and the speed of its response to its users, the computer must keep several processes in memory. There are many different memory-management schemes, reflecting various approaches to memory

management, and the effectiveness of a particular algorithm depends on the situation.

- **Storage management**. Chapters 10 through 13 describe how the file system, mass storage, and I/O are handled in a modern computer system. The file system provides the mechanism for on-line storage of and access to both data and programs. We describe the classic internal algorithms and structures of storage management and provide a firm practical understanding of the algorithms used—their properties, advantages, and disadvantages. Our discussion of storage also includes matters related to secondary and tertiary storage. Since the I/O devices that attach to a computer vary widely, the operating system needs to provide a wide range of functionality to applications to allow them to control all aspects of these devices. We discuss system I/O in depth, including I/O system design, interfaces, and internal system structures and functions. In many ways, I/O devices are the slowest major components of the computer. Because they represent a performance bottleneck, we also examine performance issues associated with I/O devices.
- **Protection and security**. Chapters 14 and 15 discuss the mechanisms necessary for the protection and security of computer systems. The processes in an operating system must be protected from one another's activities, and to provide such protection, we must ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory, CPU, and other resources of the system. Protection is a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide a means of specifying the controls to be imposed, as well as a means of enforcement. Security protects the integrity of the information stored in the system (both data and code), as well as the physical resources of the system, from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency.
- **Distributed systems**. Chapters 16 through 18 deal with a collection of processors that do not share memory or a clock—a *distributed system.* By providing the user with access to the various resources that it maintains, a distributed system can improve computation speed and data availability and reliability. Such a system also provides the user with a distributed file system, which is a file-service system whose users, servers, and storage devices are dispersed among the sites of a distributed system. A distributed system must provide various mechanisms for process synchronization and communication, as well as for dealing with deadlock problems and a variety of failures that are not encountered in a centralized system.
- **Special-purpose systems**. Chapters 19 and 20 deal with systems used for specific purposes, including real-time systems and multimedia systems. These systems have specific requirements that differ from those of the general-purpose systems that are the focus of the remainder of the text. Real-time systems may require not only that computed results be "correct" but also that the results be produced within a specified deadline period. Multimedia systems require quality-of-service guarantees ensuring that the multimedia data are delivered to clients within a specific time frame.

- **Case studies**. Chapters 21 through 23 in the book, and Appendices A through C (which are available on www.wiley.com/college/silberschatz and in WileyPLUS), integrate the concepts described in the earlier chapters by describing real operating systems. These systems include Linux, Windows XP, FreeBSD, Mach, and Windows 2000. We chose Linux and FreeBSD because UNIX—at one time—was almost small enough to understand yet was not a "toy" operating system. Most of its internal algorithms were selected for *simplicity*, rather than for speed or sophistication. Both Linux and FreeBSD are readily available to computer-science departments, so many students have access to these systems. We chose Windows XP and Windows 2000 because they provide an opportunity for us to study a modern operating system with a design and implementation drastically different from those of UNIX. Chapter 23 briefly describes a few other influential operating systems.

Operating-System Environments

This book uses examples of many real-world operating systems to illustrate fundamental operating-system concepts. However, particular attention is paid to the Microsoft family of operating systems (including Windows Vista, Windows 2000, and Windows XP) and various versions of UNIX (including Solaris, BSD, and Mac OS X). We also provide a significant amount of coverage of the Linux operating system reflecting the most recent version of the kernel —Version 2.6—at the time this book was written.

The text also provides several example programs written in C and Java. These programs are intended to run in the following programming environments:

- **Windows systems**. The primary programming environment for Windows systems is the Win32 API (**a**pplication **p**rogramming **i**nterface), which provides a comprehensive set of functions for managing processes, threads, memory, and peripheral devices. We provide several C programs illustrating the use of the Win32 API. Example programs were tested on systems running Windows Vista, Windows 2000, and Windows XP.
- **POSIX**. POSIX (which stands for *Portable Operating System Interface*) represents a set of standards implemented primarily for UNIX-based operating systems. Although Windows Vista, Windows XP, and Windows 2000 systems can also run certain POSIX programs, our coverage of POSIX focuses primarily on UNIX and Linux systems. POSIX-compliant systems must implement the POSIX core standard (POSIX.1): Linux, Solaris, and Mac OS X are examples of POSIX-compliant systems. POSIX also defines several extensions to the standards, including real-time extensions (POSIX1.b) and an extension for a threads library (POSIX1.c, better known as Pthreads). We provide several programming examples written in C illustrating the POSIX base API, as well as Pthreads and the extensions for real-time programming. These example programs were tested on Debian Linux 2.4 and 2.6 systems, Mac OS X 10.5, and Solaris 10 using the `gcc` 3.3 and 4.0 compilers.
- **Java**. Java is a widely used programming language with a rich API and built-in language support for thread creation and management. Java

programs run on any operating system supporting a Java virtual machine (or JVM). We illustrate various operating system and networking concepts with several Java programs tested using the Java 1.5 JVM.

We have chosen these three programming environments because it is our opinion that they best represent the two most popular models of operating systems: Windows and UNIX/Linux, along with the widely used Java environment. Most programming examples are written in C, and we expect readers to be comfortable with this language; readers familiar with both the C and Java languages should easily understand most programs provided in this text.

In some instances—such as thread creation—we illustrate a specific concept using all three programming environments, allowing the reader to contrast the three different libraries as they address the same task. In other situations, we may use just one of the APIs to demonstrate a concept. For example, we illustrate shared memory using just the POSIX API; socket programming in TCP/IP is highlighted using the Java API.

The Eighth Edition

As we wrote the Eighth Edition of *Operating System Concepts,* we were guided by the many comments and suggestions we received from readers of our previous editions, as well as by our own observations about the rapidly changing fields of operating systems and networking. We have rewritten material in most of the chapters by bringing older material up to date and removing material that was no longer of interest or relevance.

We have made substantive revisions and organizational changes in many of the chapters. Most importantly, we have added coverage of open-source operating systems in Chapter 1. We have also added more practice exercises for students and included solutions in WileyPLUS, which also includes new simulators to provide demonstrations of operating-system operation. Below, we provide a brief outline of the major changes to the various chapters:

- **Chapter 1, Introduction,** has been expanded to include multicore CPUs, clustered computers, and open-source operating systems.
- **Chapter 2, Operating-System Structures,** provides significantly updated coverage of virtual machines, as well as multicore CPUs, the GRUB boot loader, and operating-system debugging.
- **Chapter 3, Processes,** provides new coverage of pipes as a form of interprocess communication.
- **Chapter 4, Threads,** adds new coverage of programming for multicore systems.
- **Chapter 5, CPU Scheduling,** adds coverage of virtual machine scheduling and multithreaded, multicore architectures.
- **Chapter 6, Process Synchronization,** adds a discussion of mutual exclusion locks, priority inversion, and transactional memory.
- **Chapter 8, Main Memory,** includes discussion of NUMA.

- **Chapter 9, Virtual Memory,** updates the Solaris example to include Solaris 10 memory management.
- **Chapter 10, File-System Interface,** is updated with current technologies and capacities.
- **Chapter 11, File-System Implementation,** includes a full description of Sun's ZFS file system and expands the coverage of volumes and directories.
- **Chapter 12, Mass-Storage Structure,** adds coverage of iSCSI, volumes, and ZFS pools.
- **Chapter 13, I/O Systems,** adds coverage of PCIX PCI Express, and HyperTransport.
- **Chapter 16, Distributed System Structures,** adds coverage of 802.11 wireless networks.
- **Chapter 21, The Linux System,** has been updated to cover the latest version of the Linux kernel.
- **Chapter 23, Influential Operating Systems,** increases coverage of very early computers as well as TOPS-20, CP/M, MS-DOS, Windows, and the original Mac OS.

Programming Problems and Projects

To emphasize the concepts presented in the text, we have added several programming problems and projects that use the POSIX and Win32 APIs, as well as Java. We have added more than 15 new programming problems, which emphasize processes, threads, shared memory, process synchronization, and networking. In addition, we have added or modified several programming projects that are more involved than standard programming exercises. These projects include adding a system call to the Linux kernel, using pipes on both UNIX and Windows systems, using UNIX message queues, creating multithreaded applications, and solving the producer–consumer problem using shared memory.

The Eighth Edition also incorporates a set of operating-system simulators designed by Steven Robbins of the University of Texas at San Antonio. The simulators are intended to model the behavior of an operating system as it performs various tasks, such as CPU and disk-head scheduling, process creation and interprocess communication, starvation, and address translation. These simulators are written in Java and will run on any computer system with Java 1.4. Students can download the simulators from WileyPLUS and observe the behavior of several operating system concepts in various scenarios. In addition, each simulator includes several exercises that ask students to set certain parameters of the simulator, observe how the system behaves, and then explain this behavior. These exercises can be assigned through WileyPLUS. The WileyPLUS course also includes algorithmic problems and tutorials developed by Scott M. Pike of Texas A&M University.

Teaching Supplements

The following teaching supplements are available in WileyPLUS and on www.wiley.com/college/silberschatz: a set of slides to accompany the book, model course syllabi, all C and Java source code, up-to-date errata, three case study appendices and the Distributed Communication appendix. The WileyPLUS course also contains the simulators and associated exercises, additional practice exercises (with solutions) not found in the text, and a testbank of additional problems. Students are encouraged to solve the practice exercises on their own and then use the provided solutions to check their own answers.

To obtain restricted supplements, such as the solution guide to the exercises in the text, contact your local John Wiley & Sons sales representative. Note that these supplements are available only to faculty who use this text. You can find your Wiley representative by going to www.wiley.com/college and clicking "Who's my rep?"

Mailing List

We use the mailman system for communication among the users of *Operating System Concepts*. If you wish to use this facility, please visit the following URL and follow the instructions there to subscribe:

http://mailman.cs.yale.edu/mailman/listinfo/os-book

The mailman mailing-list system provides many benefits, such as an archive of postings, as well as several subscription options, including digest and Web only. To send messages to the list, send e-mail to:

os-book@cs.yale.edu

Depending on the message, we will either reply to you personally or forward the message to everyone on the mailing list. The list is moderated, so you will receive no inappropriate mail.

Students who are using this book as a text for class should not use the list to ask for answers to the exercises. They will not be provided.

Suggestions

We have attempted to clean up every error in this new edition, but—as happens with operating systems—a few obscure bugs may remain. We would appreciate hearing from you about any textual errors or omissions that you identify.

If you would like to suggest improvements or to contribute exercises, we would also be glad to hear from you. Please send correspondence to os-book-authors@cs.yale.edu.

Acknowledgments

This book is derived from the previous editions, the first three of which were coauthored by James Peterson. Others who helped us with previous

editions include Hamid Arabnia, Rida Bazzi, Randy Bentson, David Black, Joseph Boykin, Jeff Brumfield, Gael Buckley, Roy Campbell, P. C. Capon, John Carpenter, Gil Carrick, Thomas Casavant, Bart Childs, Ajoy Kumar Datta, Joe Deck, Sudarshan K. Dhall, Thomas Doeppner, Caleb Drake, M. Racsit Eskicioğlu, Hans Flack, Robert Fowler, G. Scott Graham, Richard Guy, Max Hailperin, Rebecca Hartman, Wayne Hathaway, Christopher Haynes, Don Heller, Bruce Hillyer, Mark Holliday, Dean Hougen, Michael Huangs, Ahmed Kamel, Morty Kewstel, Richard Kieburtz, Carol Kroll, Morty Kwestel, Thomas LeBlanc, John Leggett, Jerrold Leichter, Ted Leung, Gary Lippman, Carolyn Miller, Michael Molloy, Euripides Montagne, Yoichi Muraoka, Jim M. Ng, Banu Özden, Ed Posnak, Boris Putanec, Charles Qualline, John Quarterman, Mike Reiter, Gustavo Rodriguez-Rivera, Carolyn J. C. Schauble, Thomas P. Skinner, Yannis Smaragdakis, Jesse St. Laurent, John Stankovic, Adam Stauffer, Steven Stepanek, John Sterling, Hal Stern, Louis Stevens, Pete Thomas, David Umbaugh, Steve Vinoski, Tommy Wagner, Larry L. Wear, John Werth, James M. Westall, J. S. Weston, and Yang Xiang

Parts of Chapter 12 were derived from a paper by Hillyer and Silberschatz [1996]. Parts of Chapter 17 were derived from a paper by Levy and Silberschatz [1990]. Chapter 21 was derived from an unpublished manuscript by Stephen Tweedie. Chapter 22 was derived from an unpublished manuscript by Dave Probert, Cliff Martin, and Avi Silberschatz. Appendix C was derived from an unpublished manuscript by Cliff Martin. Cliff Martin also helped with updating the UNIX appendix to cover FreeBSD. Some of the exercises and accompanying solutions were supplied by Arvind Krishnamurthy.

Mike Shapiro, Bryan Cantrill, and Jim Mauro answered several Solaris-related questions. Bryan Cantrill from Sun Microsystems helped with the ZFS coverage. Steve Robbins of the University of Texas at San Antonio designed the set of simulators that we incorporate in WileyPLUS. Reece Newman of Westminster College initially explored this set of simulators and their appropriateness for this text. Josh Dees and Rob Reynolds contributed coverage of Microsoft's .NET. The project for POSIX message queues was contributed by John Trono of Saint Michael's College in Colchester, Vermont.

Marilyn Turnamian helped generate figures and presentation slides. Mark Wogahn has made sure that the software to produce the book (e.g., Latex macros, fonts) works properly.

Our Associate Publisher, Dan Sayre, provided expert guidance as we prepared this edition. He was assisted by Carolyn Weisman, who managed many details of this project smoothly. The Senior Production Editor Ken Santor, was instrumental in handling all the production details. Lauren Sapira and Cindy Johnston have been very helpful with getting material ready and available for WileyPlus.

The cover illustrator was Susan Cyr, and the cover designer was Howard Grossman. Beverly Peavler copy-edited the manuscript. The freelance proof-reader was Katrina Avery; the freelance indexer was WordCo, Inc.

Abraham Silberschatz, New Haven, CT, 2008
Peter Baer Galvin, Burlington, MA, 2008
Greg Gagne, Salt Lake City, UT, 2008