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. 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 7, Windows 2000, and Windows XP; and Apple Mac OS X. When we refer to Windows XP as an example operating system, we mean 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).

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 6 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 7 and 8 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 9 through 12 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 13 and 14 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.
- **Case studies**. Chapters 15 and 16 in the book, and Appendices A through C (which are available on www.wiley.com/college/silberschatz and in www.os-book.com), integrate the concepts described in the earlier chapters by describing real operating systems. Chapters 15 and 16 cover the Linux and Windows 7 operating systems. The online Appendices include 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 7 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.

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 7, 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 XPand Windows 7.
- **POSIX**. POSIX (which stands for *Portable Operating System Interface*) represents a set of standards implemented primarily for UNIX-based operating systems. Although Windows 7, 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 10.6, 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 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.

Operating System Essentials

We have based *Operating System Essentials* on the Eighth Edition of *Operating System Concepts*, published in 2009. Our intention behind developing this *Essentials* edition is to provide readers with a textbook that focuses on the

core concepts that underlie contemporary operating systems. By focusing on core concepts, we believe students are able to grasp the essential features of a modern operating system more easily and more quickly.

To achieve this, *Operating System Essentials* omits the following coverage from the Eighth Edition of *Operating System Concepts*:

- We remove coverage of pipes as a form of interprocess communication in Chapter 3.
- We remove coverage of Atomic Transactions in Chapter 6.
- We remove Chapter 7—Deadlocks—and instead offer a brief overview of deadlocks in Chapter 6.
- We remove Chapters 16 through 18, which cover distributed systems.
- Chapter 19 (Real-Time Systems) and Chapter 20 (Multimedia Systems) are removed.
- Chapter 16, which covers Windows 7 and is a new chapter, replaces the chapter on Windows XP in the Eighth Edition.

This *Essentials* edition includes updated coverage of many topics relevant to the study of operating systems. Most importantly, it includes updated coverage of multicore CPUs, virtual machines, and open-source operating systems as well as updated content on file and I/O Systems.

Programming Problems and Projects

To emphasize the concepts presented in the text, we have several programming problems and projects that use the POSIX and Win32 APIs, as well as Java. The programming problems emphasize processes, threads, shared memory and process synchronization. In addition, we have included several programming projects that are more involved than standard programming exercises. These projects include adding a system call to the Linux kernel, using UNIX message queues, creating multithreaded applications, and solving the producer–consumer problem using shared memory.

Teaching Supplements

The site www.wiley.com/college/silberschatz contains the following teaching supplements: a set of slides to accompany the book, model course syllabi, all C and Java source code, up-to-date errata, and two case study appendices.

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?"

Contacting Us

We have attempted to clean up every error in this edition, but—as happens with operating systems—a few obscure bugs may remain; an up-to-date errata list is accessible from the book's home page. We would appreciate hearing from you about any textual errors or omissions in the book that are not on the current list of errata.

We would be glad to receive suggestions on improvements to the book. We also welcome any contributions to the book's Web page that could be of use to other readers, such as programming exercises, project suggestions, on-line labs and tutorials, and teaching tips.

E-mail should be addressed to os-book-authors@cs.yale.edu. Any other correspondence should be sent to Avi Silberschatz, Department of Computer Science, Yale University, 51 Prospect Street, P.O. Box 208285, New Haven, CT 06520-8285 USA.

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

Chapter 16 was written by Dave Probert and was derived from Chapter 22 of the Eighth Edition of Operating System Concepts. Parts of Chapter 11 were derived from a paper by Hillyer and Silberschatz [1996]. Chapter 15 was derived from an unpublished manuscript by Stephen Tweedie. Cliff Martin 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 and Bryan Cantrill helped with the ZFS coverage. 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. Judi Paige and 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 executive editor, Beth Golub, provided expert guidance as we prepared this edition. She was assisted by Mike Berlin, who managed many details of this project smoothly. The Senior Production Editor, Ken Santor, was instrumental in handling all the production details.

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

Abraham Silberschatz, New Haven, CT, 2010
Peter Baer Galvin, Burlington, MA, 2010
Greg Gagne, Salt Lake City, UT, 2010