Operating System Concepts

Second Edition

James L. Peterson
Abraham Silberschatz

Operating System Concepts

Second Edition

James L. Peterson
Abraham Silberschatz

University of Texas at Austin

Addison-Wesley Publishing Company

Reading, Massachusetts • Menlo Park, California
Don Mills, Ontario • Wokingham, England • Amsterdam
Sydney • Singapore • Tokyo • Mexico City • Bogotá
Santiago • San Juan

To my parents, Wira and Mietek, my wife, Haya, and my children, Lemor, Sivan and Aaron.

Avi Silberschatz

To my wife, Jeanne, and my children, Jennifer and Kathryn.

Jim Peterson

Mark Dalton: Sponsoring Editor

Hugh Crawford: Manufacturing Supervisor
Karen Guardino: Production Manager
Thomas A. Philbrook and Barbara Atkinson: Cover Designers
Susan E. Vicenti: Art Editor
Natasha Wei: Production Editor

This book is in the Addison-Wesley series in Computer Science.

Michael A. Harrison: Consulting Editor

Library of Congress Cataloging in Publication Data

Peterson, James Lyle.
Operating system concepts.

Bibliography: p.
Includes index.
1. Operating systems (Computers) I. Silberschatz, Abraham. II. Title.
QA76.6.P475 1985 001.64'2 84-21637
ISBN 0-201-06198-8

Scope® Registered trademark of Control Data Corporation.
VMS™ Trademark of Digital Equipment Corporation.
CP/M® Registered trademark of Digital Research Incorporated.
UNIX™ Trademark of Bell Laboratories.

Reproduced by Addison-Wesley from camera-ready copy prepared by the authors.

CDEFGHIJ-AL-89876

Preface

Operating systems are an essential part of a computer system. Similarly, a course on operating systems is an essential part of a computer science education. This book is intended as a text for an introductory course in operating systems at the junior, senior, or first-year graduate level. It provides a clear description of the *concepts* which underlie operating systems.

This book is not centered around any particular operating system or hardware. Instead, it discusses fundamental concepts which are applicable to a variety of systems. Our emphasis is on solving the problems encountered in designing an operating system, regardless of the underlying hardware on which the system will run.

Content of this Book

The overall content of the book is as follows:

As prerequisites, we assume the reader is familiar with general assembly language programming and computer organization. We do not discuss in any detail the characteristics of I/O devices or how to write device drivers.

Chapters 1, 2, and 3 explain what operating systems *are* and what they *do*. These chapters explain how the concept of an operating system has developed, the common features of an operating system, what it does for the user, and what it does for the computer system operator. It is motivational, historical, and explanatory in nature. We try to avoid how things are done internally in these chapters. Therefore, these chapters are suitable for individuals or lower-level classes who want to learn what an operating system is, without getting into the details of the internal algorithms.

Chapters 4 to 8 deal with the classic internal algorithms and structures of operating systems: *cpu scheduling*, *memory management*, and *device management*. They provide a firm practical understanding of the algorithms used: their properties, advantages, and disadvantages. The algorithms are presented in a natural order, so that new, more complicated systems can be built upon the understanding of simpler systems.

Chapter 9 introduces the unifying concept of the computer system as a collection of cooperating sequential processes. Chapters 10, 11, 12, and 13 present advanced topics and current trends, including high-level languages for writing concurrent programs, protection systems, design principles, and distributed systems. These topics are still being researched and may well need later revision. However, we include them in the book for two reasons. First, although research is still ongoing and final solutions to these problems are still being sought, there is general agreement that these topics are important and students should be exposed to them. Second, existing systems use these solutions, and anyone working with operating systems over the next five years will need to be aware of the developments in these directions.

In response to many requests, however, we have included a new chapter to illustrate how the many described concepts can be put together in a real system. We have chosen the Unix operating system, specifically Berkeley's 4.2BSD, for this example system. This operating system was chosen in part because it was at one time almost small enough to understand and yet is not a "toy" operating system. Most of its internal algorithms were selected for *simplicity*, not speed or sophistication. Unix is readily available to computer science departments, so many students may have used Unix.

Each chapter ends with references to further reading. Chapter 15 is essentially a set of references to further reading for the entire book, describing briefly some of the most influential operating systems.

Organization

Operating systems first began to appear in the late 1950's, and for twenty years underwent major changes in concepts and technology. As a result, the first-generation operating system textbooks that appeared during this period (such as Brinch Hansen [1973a], Madnick and Donovan [1974], Shaw [1974], Tsichritzis and Bernstein [1974]) tried to explain a subject that changed even as they were being written.

Now, however, operating system theory and practice appears to have matured and stabilized. The fundamental operating system concepts are now well defined and well understood. While there will undoubtedly be new algorithms, the basic approach to cpu scheduling, memory management, the user interface, and so on, is not likely to change. Few really new operating systems are being written. Most large computers use operating systems that were designed in the 1960's. The newest operating systems are being developed for the multitude of microcomputer systems, but these are generally either CP/M, Unix, or imitations of these. It is now possible to write a book that presents well-understood, agreed-upon, classic operating system material.

This text is one of a second generation of operating system textbooks (such as Calingaert [1982]). Our text differs from other texts in the level of content and organization. The basic concepts have been carefully organized and presented; the material flows naturally from these basic principles to more sophisticated ones.

The only controversial aspect of this book is its organization, specifically the definition of the formal process model as late as Chapter 9. Almost every other text places this material at the beginning as Chapter 2. In our experience, this arrangement does not work. The process model is a powerful and convenient unifying concept. However, when operating systems are first introduced, the student does not know the basic principles. To benefit from the process model, the student needs to understand how cpu scheduling and memory management can present an image of separate virtual processors, each with its own separate virtual memory space. Then, and only then, will the student really be able to understand why the process model is useful. Once the student has the proper background to be able to appreciate the process

model of operating systems, the standard material concerning processes, process coordination, synchronization, and communication can be presented.

Concurrency itself, in the form of overlapped I/O, spooling, multiprogramming, and time-sharing, is introduced as early as Chapter 1. However, we feel that the *formal* process model is best reserved until the basic concepts (cpu scheduling and memory management) are well understood.

The Second Edition

Many comments and suggestions were forwarded to us concerning our first edition. These, together with our own observations while teaching at the University of Texas and IBM, have prodded us to produce this second edition. Our basic procedure was to reorganize and rewrite the material in each chapter, to bring some of the older material up-to-date, to improve the exercises, and to add a new chapter on Unix.

Substantive revisions were made in the following chapters:

- **Chapter 1.** A new organization clearly separates the performance and protection aspects of operating systems.
- **Chapter 3.** Sections have been reorganized to present a more natural flow of information, discussing first files, and then directories.
- **Chapter 8.** Additional examples illustrate the behavior of the various algorithms. Section 8.6, on recovery, has been rewritten to obtain a better organization.
- **Chapter 9.** Section 9.5 has been rewritten to bring the material up-to-date. In particular, a new, more concise definition of the critical section problem has been included, Dekker's algorithm for the synchronization of two processes has been replaced with Peterson's algorithm, and a new synchronization algorithm using special hardware instructions (that is, test and set) has been included. The section of the RC4000 system has been replaced by a discussion of the more modern Accent system.
- **Chapter 13.** A new subsection on the Byzantine general problem has been added.
- **Chapter 14.** A problem common to students (and professors) using the first edition was that they felt overwhelmed by the variety of solutions to the many aspects of operating systems. It was difficult

to see how a complete system would fit all the pieces together. To address this problem, we have added a chapter on a complete operating system. The initial problem was to create either a "paper" design (such as that used in Lister [1979]) or a real system. We decided to present a real system. The next problem was to choose a particular system. We chose Unix, specifically 4.2 BSD.

- Chapter 15. Since a new Chapter 14 was added, the old Chapter 14 (without the section on Unix) was renumbered Chapter 15.

Errata

We have attempted to clean up every error in this book, but as with operating systems, there will undoubtedly still be some obscure bugs. We would appreciate it if you, the reader, would notify us of any errors or omissions in the book. If you would like to suggest improvements or contribute exercises, we would be glad to hear from you. An errata sheet is available to instructors for the first edition, and we will update it with errors in this edition as they become known.

Acknowledgments

Eight years of CS 372 students at the University of Texas at Austin suffered through permutations of this material until we got it right. David Orshalick helped with the early table of contents. Dick Kieburtz helped with the contents of Section 11.9 on language-based protection. During the writing stage, we were invited to design and teach an operating system course for IBM, which helped clarify our organization.

As the text was written, Carol Engelhardt deciphered our handwriting and edited our text into Scribe format. Carol's efforts throughout this project were the only thing that got it done.

Jeff Ullman helped us to get draft copies on the Dover at Stanford. Arthur Keller helped get those drafts back to Texas. Susan Lilly was able to understand what we were trying to say in the drafts and edit them into readable text. Elaine Rich, Richard Cohen, and Brian Reid explained the subtleties of Scribe, helping us to define our documents and make them work. The manuscript was read in various forms by Michael Molloy, Gael Buckley, and the reviewers.

During the lengthy revision process for this Second Edition, the Information Technology Center of Carnegie-Mellon University provided a supportive work environment.

Chapter 14 is derived from a draft by John Quarterman, who received comments on earlier drafts from Samuel J. Leffler, Bill Shannon, William N. Joy, and John B. Chambers. John Quarterman endured months of our questions and endless reviews as we struggled to write and rewrite the chapter. The credit for this chapter should go to John Quarterman, while any errors in presentation or fact for Chapter 14, as with the rest of the book, are, of course, ours.

Finally, we would like to acknowledge the helpful reviewing of Larry Flanigan, University of Michigan; William Appelbe, University of California at San Diego; Christopher Haynes, Indiana University; and Raymond Hookway, Case Western Reserve University.

J.P.
A.S.

F8702/61 （英3-5/3880）
操作系统概念 第2版
BG000680

Contents

Chapter 4 CPU Scheduling

Chapter 5 Memory Management

Chapter 6 Virtual Memory

Chapter 7 Disk and Drum Scheduling

Chapter 8 Deadlocks

Chapter 9 Concurrent Processes

Chapter 10 Concurrent Programming

Chapter 11 Protection

Chapter 12 Design Principles

Chapter 13 Distributed Systems

Chapter 14 The Unix Operating System

Chapter 15 Historical Perspective

1 Introduction

An *operating system* is a program which acts as an interface between a user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user may execute programs. The primary goal of an operating system is thus to make the computer system *convenient* to use. A secondary goal is to use the computer hardware in an *efficient* manner.

To understand what operating systems are, it is necessary to understand how they have developed. In this chapter, we trace the development of operating systems from the first hands-on systems to current multiprogrammed and time-shared systems. As we move through the various stages, we see how the components of operating systems evolved as natural solutions to problems in early computer systems. Understanding the reasons behind the development of operating systems gives an appreciation for what an operating system does and how it does it.

1.1 What Is an Operating System?

An operating system is an important part of almost every computer system. A computer system can be roughly divided into 4 components (Figure 1.1):

- The **hardware** (cpu, memory, I/O devices).
- The **operating system**.
- The **applications programs** (compilers, database systems, video games, business programs).
- The **users** (people, machines or other computers).

The hardware provides the basic computing resources. The applications programs define the ways in which these resources are used to solve the