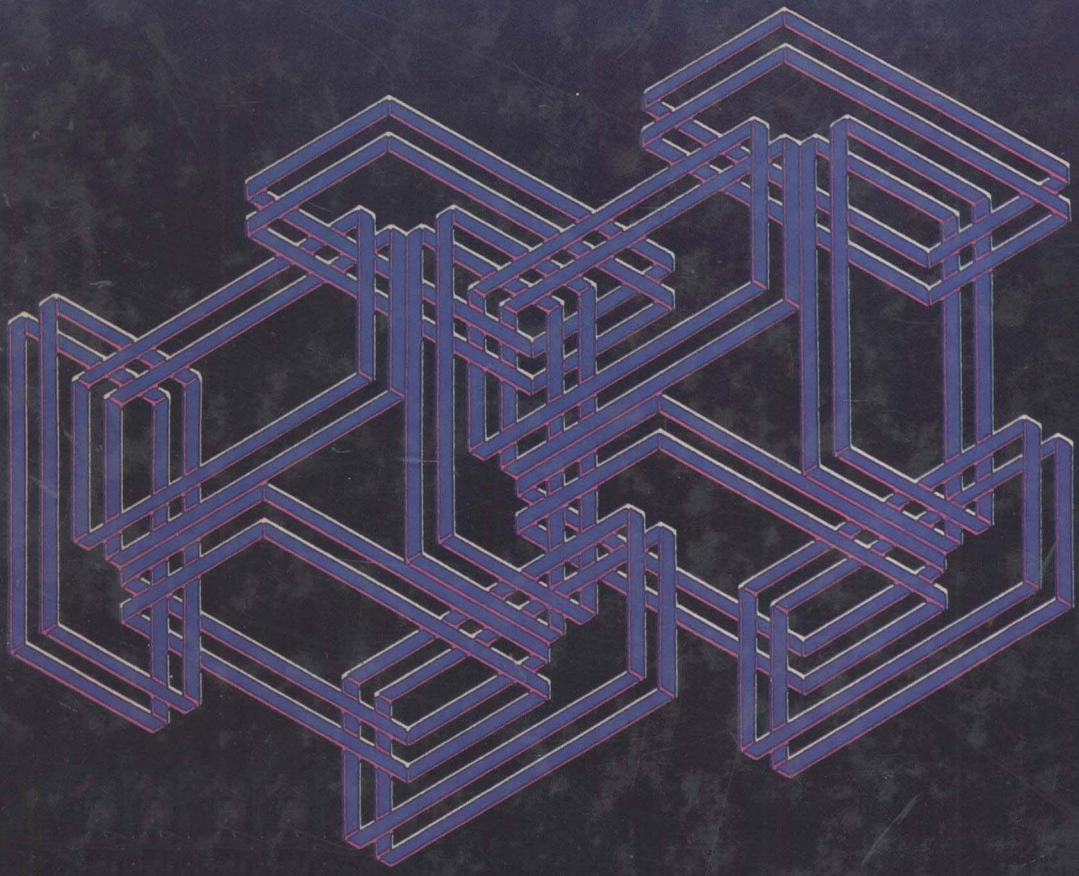
ALTERNATE EDITION

Operating System Concepts



Abraham Silberschatz • James L. Peterson

ALTERNATE EDITION

Operating System Concepts

Abraham Silberschatz
University of Texas at Austin

James L. Peterson
Microelectronic and Computer Technology Corporation (MCC)

Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn • Sydney

Jim DeWolf: Sponsoring Editor
Karen Guardino: Managing Editor
Karen Myer: Production Supervisor
Hugh Crawford: Manufacturing Supervisor
Marshall Henrichs: Cover Designer
Joe Vetere: Technical Art Consultant
Wendy Lewis: Production Coordinator
Lorraine Hodsdon: Layout Artist

This book is in the Addison-Wesley Series in Computer Science.
Michael A. Harrison: Consulting Editor

Library of Congress Cataloging-in-Publication Data

Silberschatz, Abraham.
Operating systems concepts.
Bibliography: p.
Includes index.
1. Operating systems (Computers) I. Peterson, James Lyle. II. Title.
QA76.76.063S5583 1988 005.4'3 87-33451
ISBN 0-201-18760-4

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

Reprinted with corrections April, 1989

EFG-DO-89

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

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* that underlie operating systems.

This book is not centered around any particular operating system or hardware. Instead, it discusses fundamental concepts that 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.

Many comments and suggestions were forwarded to us concerning our first and second editions. These, together with our own observations while teaching at the University of Texas and IBM have prodded us to produce this alternate edition.

The Alternate Edition

The major undertaking was to restructure the organization of the book by moving the discussions concerning concurrency and system design to the front of the text. Since concurrency is the very heart of modern operating systems, it was felt that it is important that students get this fundamental concept early. This allows students to immediately begin work on projects in laboratory-oriented courses, or to start understanding the internals of real systems.

Our basic procedure was to rewrite the material in each chapter, bringing some of the older material up-to-date, improving the exercises, and adding new references.

Content of this Book

As prerequisites, we assume the reader is familiar with general assembly language programming and computer organization. Chapters 1 and 2 explain what operating systems *are*, what they *do*, and how they are *designed* and *constructed*. 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 have avoided a discussion on how things are done internally in these chapters. Therefore, they 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 3 to 5 deal with the process concept and concurrency, which is at the very 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 being user processes (those that execute user code). These chapters cover various methods for process management, cpu scheduling, and deadlock handling.

Chapters 6 through 9 deal with the classic internal algorithms and structures of storage management and file system. 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 10 (protection systems) and Chapter 11 (distributed systems) present advanced topics and current trends. 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 Chapter 12 we illustrate how the many concepts described 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 it.

Each chapter ends with references to further reading. Chapter 13 is essentially a set of references to further reading for the entire book and briefly describes some of the most influential operating systems.

Organization

Operating systems first began to appear in the late 1950s, 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 was changing 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 1960s. The newest operating systems are being developed for the multitude of microcomputer systems, but these are generally either MS-DOS, 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. 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.

Errata

This book has benefited from the careful reading and thoughtful comments of many people in the previous two editions. We have attempted to clean up every error in this new edition, 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 second edition, and we will update it with errors in this edition as they become known.

Acknowledgments

This book is derived from the previous two editions, and so, has been helped by many people, including Gael Buckley, Richard Cohen, Dick Kieburtz, Carol Kroll, Michael Molloy, John Quarterman, Elaine Rich, and Sara Strandtman.

We would also like to acknowledge the helpful reviewing of Richard Schlichting, University of Arizona; Charles Oualline, East Texas State University; John Stankovic, University of Massachusetts at Amherst; and Ajoy Kumar Datta, Arizona State University.

A.S
J.P

Contents

CHAPTER 1

Introduction

An *operating system* is a program that 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 four 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 computing problems of the users. There may be many different users trying to solve different problems. Accordingly, there may be many different applications programs. The operating system controls and coordinates the use of the hardware among the various application programs for the various users.

An operating system is similar to a *government*. The basic resources of a computer system are provided by its hardware, software, and data. The operating system provides the means for the proper use of these resources in the operation of the computer system. Like a government, the operating system performs no useful function by itself. It simply provides an *environment* within which other programs can do useful work.

We can view an operating system as a *resource allocator*. A computer system has many resources (hardware and software) that may be required to solve a problem: cpu time, memory space, file storage space, input/output (I/O) devices, and so on. The operating system acts as the manager of these resources and allocates them to specific programs and users as necessary for their tasks. Since there may be many, possibly conflicting, requests for resources, the operating system must decide which requests are allocated resources to operate the computer system fairly and efficiently.

A slightly different view of an operating system focuses on the need to control the various I/O devices and user programs. An operating system is a *control program*. A control program controls the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

In general, however, there is no completely adequate definition of an operating system. Operating systems exist because they are a reasonable way to solve the problem of creating a usable computing system. The fundamental goal of computer systems is to execute user programs and solve user problems. Towards this goal computer hardware is constructed. Since bare hardware alone is not very easy to use, applications programs are developed. These various programs

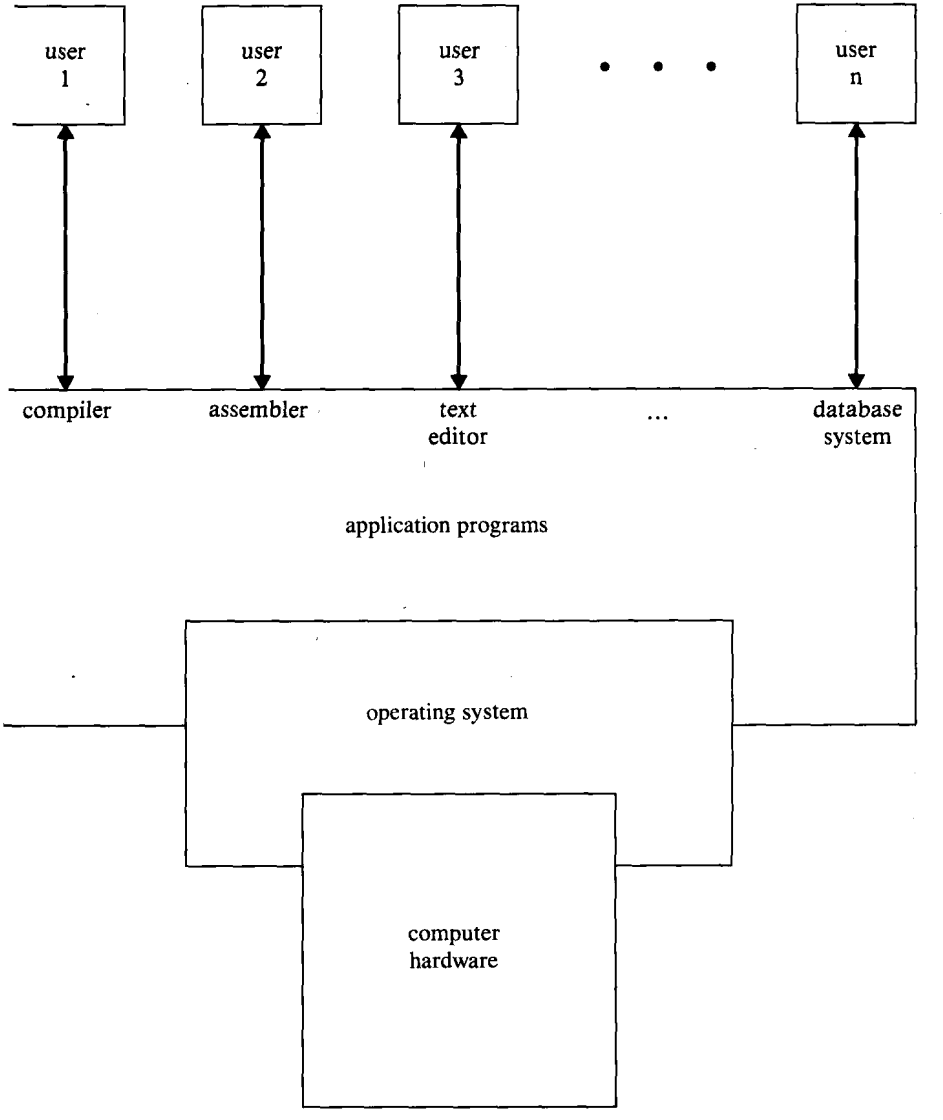


Figure 1.1 Abstract view of the components of a computer system

require certain common operations, such as controlling the I/O devices. The common functions of controlling and allocating resources are then brought together into one piece of software: the operating system.

It is perhaps easier to define operating systems by what they *do*, rather than what they *are*. The primary goal of an operating system is *convenience for the user*. Operating systems exist because they are supposed to make it easier to compute with an operating system than without an operating system. This is particularly clear when you look at operating systems for small personal computers.

A secondary goal is *efficient* operation of the computer system. This goal is particularly important for large, shared multi-user systems. These systems are typically very expensive, and so it is desirable to make them as efficient as possible. These two goals, convenience and efficiency, are sometimes contradictory. In the past, efficiency considerations were often more important than convenience. Thus much of operating system theory concentrates on optimal use of computing resources.

To see what operating systems are and what operating systems do, let us consider how they have developed over the last 30 years. By tracing that evolution we can identify the common elements of operating systems and see how and why they have developed as they have.

Operating systems and computer architecture have had a great deal of influence on each other. To facilitate the use of the hardware, operating systems were developed. As operating systems were designed and used, it became obvious that changes in the design of the hardware could simplify the operating system. In this short historical review, notice how the introduction of new hardware features is the natural solution to many operating system problems.

1.2 Early Systems

Initially, there was only computer hardware. Early computers were (physically) very large machines run from a console. The programmer would write a program and then operate the program directly from the operator's console. First, the program would be manually loaded into memory, either from the front panel switches, paper tape, or punched cards. Then the appropriate buttons would be pushed to load the starting address and to start the execution of the program. As the program ran, the programmer/operator could monitor its execution by the display lights on the console. If errors were discovered, the programmer could halt the program, examine the contents of memory and registers, and debug the program directly from the console. Output was printed, or punched onto paper tape or cards for later printing.