

Kirkland & Ellis LLP Library



5 0628 01266699 3

# An Introduction to Operating Systems

Revised First Edition

Harvey M. Deitel

Including Case Studies in:

UNIX<sup>TM</sup>  
VAX<sup>TM</sup>  
CP/M<sup>®</sup>  
MVS  
VM

Concurrent Programming in Ada<sup>®</sup>

DEVRY DECATUR LIBRARY



3 1944 108398 1

Sponsoring Editor: William B. Gruener  
Production Editor: William J. Yskamp

Text Designer: Herb Caswell  
Illustrator: ANCO/Boston  
Cover Designer: Gary Fujiwara  
Art Coordinator: Susanah H. Michener

Production Manager: Susan Zorn  
Production Coordinator: Helen M. Wythe

The text of this book was composed in  
Times Roman by International Computaprint Corporation

CP/M®, MP/M® and CP/NET® are registered trademarks of Digital Research, Incorporated  
UNIX™ is a trademark of Bell Laboratories, Incorporated  
XENIX™ is a trademark of Microsoft Corporation  
VAX™ is a trademark of Digital Equipment Corporation  
MVS™ and VM™ are trademarks of International Business Machines Corporation  
Ada® is a registered trademark of the United States Government, Ada Joint Program Office.

#### Library of Congress Cataloging in Publication Data

Deitel, Harvey M., 1945-  
An introduction to operating systems.  
Includes indexes.  
1. Operating systems (Computers) I. Title.  
QA76.6.D44 1983b 001.64'25 83-7153  
ISBN 0-201-14501-4

*Reprinted with corrections, July 1984*

Copyright © 1984 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

GHIJKLNNOP-HA-898765

## 1.1 OPENING REMARKS

Only a few people who will read this text can recall a time when there were no operating systems. Operating systems today are used on computers ranging in size from giant *mainframes* to small *personal computers*. Operating systems often shape the user's view of a computer even more than the hardware itself. In the personal computing field, for example, the *CP/M*<sup>®</sup> operating system (see Chapter 21) has become a de facto standard, and a very large number of computer manufacturers build hardware to support *CP/M*. The user sees the hardware's packaging such as the CRT and the keyboard, but the functional view is provided by *CP/M*.\*

In the 1960s one might have defined operating system as the *software that controls the hardware*. But today there is a significant trend for functions to migrate from software into *firmware*, i.e., *microcode* (see Chapter 2). This trend has become so pronounced that it is likely the functions coded in firmware may soon exceed those in software on some systems.

It is clear that a better definition of operating system is needed. *We view an operating system as the programs, implemented in either software or firmware, that make the hardware usable. Hardware provides "raw computing power." Operating systems make this power conveniently available to users.* They also manage the hardware carefully to achieve good performance (see Part 6 of the text).

However we choose to define them, operating systems are an integral part of the computing environment, and they must be understood to some degree by every computer user.

Some of the operating systems available today consume a very large part of a computer's resources as they function. Some users view this as a conspiracy by hardware manufacturers to sell more hardware. In fact, most hardware manufacturers view operating systems as a necessary evil without which their hardware would not sell at all. These views are being tempered somewhat by the availability of microprocessor-based systems and personal computers with relatively simple operating systems. Instead of providing only the all-encompassing, large-scale, general-purpose operating systems the industry thought necessary in the 1960s and early 1970s, vendors (and independents) now also supply simpler operating systems that provide convenient computing environments meeting the specific needs of individual users.

An operating system is primarily a resource manager, and the primary resource it manages is computer hardware. It provides many features including

- defining the "user interface"
- sharing the hardware among users
- allowing users to share data among themselves

---

\**CP/M*<sup>®</sup> is a Registered Trademark of Digital Research.

## 6 Introduction

- scheduling resources among users
- facilitating input/output
- recovering from errors

The key resources an operating system manages are

- processors
- storage
- input/output devices
- data

The operating system interfaces with

- computer operators
- applications programmers
- systems programmers
- administrative personnel
- programs
- hardware
- users (such as bank tellers)

*Users* are the customers in a computing environment who utilize the computer to perform useful work. *Computer operators* are people charged with the responsibility of monitoring the operating system, responding to requests for intervention, mounting and dismounting tapes and disks, loading and unloading cards, making sure that the right forms are placed on printers and that the forms are properly aligned, etc. Operators are integral to the smooth operation of a system. They provide those functions that have not as yet been automated.

*Systems programmers* are generally concerned with maintaining the operating system, tailoring it to the needs of the installation, and modifying it to support new types of devices.

*Systems administrators* are people who set policy and interface with the operating system to ensure that those policies are properly implemented.

Programs interface with operating systems via special instructions variously called *supervisor calls*, *monitor calls*, *executive requests*, etc. These instructions allow the users to obtain services from the operating system without compromising its integrity.

The operating system is generally given the status of *most trusted user*. It is allowed to access all features of the hardware, all user programs and data, etc. In some systems, portions of the operating system are less trusted because the people and programs with access to these portions do not require carte blanche access, and so it is best not to give it to them (see Chapter 17).

## 2.1 INTRODUCTION

In this chapter we consider hardware, software, and firmware. All of these contribute to the operation and control of computer systems. *Hardware* consists of the devices of a computer system—its processors, its storages, its input/output devices, and its communications connections. *Software* consists of the programs of machine language instructions and data that are interpreted by the hardware. Some common types of software are compilers, assemblers, loaders, linkage editors, linking loaders, user applications programs, database management systems, data communications systems, and operating systems. *Firmware* consists of microcode programs executed from very high-speed control storage. Commonly used object programs “burned” into read-only memories (ROMs and PROMs) are also sometimes called firmware. The last portion of this chapter discusses microprogramming and its importance in today’s computer architectures and operating systems.

## 2.2 HARDWARE

In the next several sections various hardware items of importance to operating systems are discussed. The reader who needs more detail than is presented here should consult texts on computer architecture such as (Ba80) (Il82) and (Le80).

### 2.2.1 Storage Interleaving

*Storage interleaving* is used to speed the accessing of primary storage. Ordinarily while any one of the locations in a *primary storage bank* is being accessed, no other references may be in progress. Storage interleaving places adjacent storage locations in different storage banks so that multiple references may be in progress at the same time. Two-way interleaving, for example, would place the odd-numbered storage addresses in one storage bank and the even-numbered addresses in another. In simple sequential references to primary storage, locations are referenced in order. Thus storage interleaving allows several locations to be referenced at once as long as they are in different storage banks.

### 2.2.2 Relocation Register

A *relocation register* allows programs to be relocated dynamically. The base address of the program in primary storage is placed in the relocation register. The contents of the relocation register are added to each address developed by a running program. The user is able to program as if the user’s program begins at location zero. At execution time, as the program runs, all address references involve the relocation register; this allows the program to reside in locations other than those which it was translated to occupy.

### 2.2.11 Base-plus-Displacement Addressing

As the need for larger primary storages became apparent, computer architectures were modified to accommodate a very large range of addresses. A system designed to support 16M bytes ( $M=1,048,576$ ) would require 24-bit addresses. Incorporating such lengthy addresses into every instruction on a machine with single-address instructions would be costly; on a machine with multiple-address instructions it would be intolerable. So to achieve very large address ranges, systems use *base-plus-displacement addressing* in which all addresses are added to the contents of a *base register*. This scheme has the additional advantage of making programs *location-independent*, a particularly valuable property for programs in multiuser environments in which a program may have to be placed at different locations in primary storage each time it is loaded.

### 2.2.12 Problem State, Supervisor State, Privileged Instructions

Computer systems generally have several different *execution states*. Varying the state of a machine makes it possible to build more secure systems. Normally, when the machine is in a particular state only some subset of its instructions is executable by a running program. For user programs, the subset of instructions the user may execute in *problem state* precludes, for example, the direct execution of input/output instructions; a user program allowed to perform arbitrary input/output could dump the system's master list of passwords, print the information of any other user, or destroy the operating system. The operating system ordinarily runs with *most trusted user status* in a *supervisor state*; it has access to all of the instructions in the machine's instruction set. Such a problem state/supervisor state dichotomy has been adequate for most modern computing systems. In the case of highly secure systems, however, it is desirable to have more than two states. This allows a finer *granularity of protection*. It also allows access to be granted by the *principle of least privilege*: Any particular user should be granted the least amount of privilege and access that user needs to accomplish its designated tasks.

It is interesting that as computer architectures have evolved, the number of *privileged instructions*, i.e., those instructions not accessible in problem state, has tended to increase. This indicates a definite trend toward the incorporation of more operating systems functions in hardware. Some microprocessors already have complete operating systems in firmware; many have much operating system functionality in hardware.

### 2.2.13 Virtual Storage

Virtual storage systems allow programs to reference addresses that need not correspond to the real addresses available in primary storage. The virtual addresses developed by running programs are translated dynamically (i.e., at execution time) by

and sometimes on the operating system. The association of instructions and data items with particular storage locations is called *binding*. In machine language programming, binding is performed at the moment of coding. The trend has been to defer binding as much as possible; today's virtual storage systems perform binding dynamically as a program executes. Associated with the delay of binding is an increase in flexibility for both the user and the system, but this occurs at the cost of increasingly sophisticated translators, loaders, hardware, and operating systems.

A loader is a program that places a program's instructions and data into primary storage locations. An *absolute loader* places these items into the precise locations indicated in the machine language program. A *relocating loader* may load a program at various places in primary storage depending perhaps on the availability of primary storage at the moment of loading (called *load time*).

### 2.3.10 Linking Loaders and Linkage Editors

In early computing environments, a programmer would produce a machine language program that contained every instruction needed to solve a particular problem. Even the complex and awkward control of input/output had to be hand-coded into each machine language program.

Today, user programs often contain only a small portion of the instructions and data needed to solve a given problem. Large *subroutine libraries* are supplied so that a programmer wanting to perform certain common operations may use system-supplied routines to do so. Input/output in particular is normally handled by routines outside of the user program. Therefore, the machine language program produced by a translator must normally be combined with other machine language programs to form a useful execution unit. This process of *program combination* is performed by *linking loaders* and *linkage editors* prior to program execution time.

At load time, a linking loader combines whatever programs are required and loads them directly into primary storage. A linkage editor also performs this program combination, but it creates a load image that it preserves on secondary storage for future reference. The linkage editor is particularly useful in production environments; when a program is to be executed, the load image produced by the linkage editor may be loaded immediately without the overhead (often extensive) of recombining program pieces.

## 2.4 FIRMWARE

The concept of microprogramming is generally attributed to Professor Maurice Wilkes. His 1951 paper (Wi51) presented the concepts that form the basis of current microprogramming techniques. However, it wasn't until the System/360 appeared in the mid-1960s that microprogramming was used on a wide scale. During the 1960s computer manufacturers used microprogramming to implement machine language instruction sets (Hu70).

organization and computer architecture such as (Ba80) (Le80) (Ha78) (Hy78) (St75) (Ge74).

### 3.9 THE NUCLEUS OF THE OPERATING SYSTEM

All of the operations involving processes are controlled by a portion of the operating system variously called its *nucleus*, *core*, or *kernel* (Lo81, Br70, Sc77, Sh75, Wu74). The nucleus represents only a small portion of the code of the entire operating system, but it is among the most intensively used code. For this reason, the nucleus ordinarily remains in primary storage while other portions of the operating system are shuttled to and from secondary storage as needed.

One of the most important functions included in the nucleus is interrupt processing. In large, multiuser systems, there is a constant flood of interrupts directed at the processor. Rapid response to these interrupts is essential to keeping the resources of the system well utilized, and to providing acceptable response times to interactive users.

The nucleus disables interrupts while it is responding to an interrupt; interrupts are again enabled after the processing of an interrupt is complete. With a steady flow of interrupts, it is possible that the nucleus could have interrupts disabled for a large portion of the time; this could result in poor response to interrupts. Therefore, nuclei are designed to do the "bare minimum" amount of processing that is possible on each interrupt, and then to pass the remaining processing of each interrupt to an appropriate system process that can operate while the nucleus is enabled for further interrupts. This ultimately means that interrupts may be enabled a much larger percentage of the time, and that the system is more responsive.

#### 3.9.1 A Summary of Nucleus Functions

An operating system nucleus normally contains the code to perform the following functions.

- Interrupt handling.
- Process creation and destruction.
- Process state switching.
- Dispatching. (See also Chapter 10.)
- Process suspension and resumption. (See also Chapter 10.)
- Process synchronization. (See Chapters 4 and 5.)
- Interprocess communication. (See Chapters 4 and 5.)
- Manipulation of process control blocks.
- Support of input/output activities. (See the case studies of Chapters 18 through 22. Input/output is generally unique to each different type of computer system.)
- Support of storage allocation and deallocation. (See Chapters 7, 8, and 9.)

- Support of the file system. (See Chapter 13.)
- Support of a procedure call/return mechanism.
- Support of certain system accounting functions. (See Chapter 21.)

The reader interested in a detailed explanation of the structure and operation of the nucleus should refer to Chapter 6 of the text by Lorin and Deitel (Lo81).

### **3.9.2 Enabling and Disabling Interrupts**

The nucleus is normally entered by an interrupt. The nucleus disables interrupts while it responds to the interrupt. Once the cause of the interrupt is determined, the nucleus passes the processing of the interrupt to a specific system process designed to handle that type of interrupt.

In some systems, all of the processing of each interrupt is done by a large, one-piece operating system. In such systems, interrupts are disabled for a larger percentage of the time, but the systems are conceptually simpler. Such a design is useful in small systems that support a limited number of processes; in large multiuser systems the nucleus approach yields far better performance.

### **3.9.3 Hierarchical System Structure**

The literature includes papers (Di68) that advocate a hierarchical approach to designing operating systems. At the base of the hierarchy is the computer hardware itself, sometimes called the “raw machine” or “naked iron.” At the next level up in the hierarchy (or the next several levels in some designs) are the various nucleus functions. These are viewed as creating an *extended machine*, i.e., a computer that offers not only its machine language in support of the operating system and its users but also a group of additional capabilities provided by the nucleus functions. These additional capabilities are often called *primitives*.

Above the nucleus in the hierarchy are the various operating system processes that operate in support of user processes—for example, the device manager processes that actually supervise the input/output operations for system devices in behalf of various users. At the top of the hierarchy are the user processes themselves.

Such hierarchical designs have proven easier to debug, modify, and to prove correct (Di68). In designs in which the nucleus itself is spread over several levels of the hierarchy, the choice of which nucleus function is to be placed at which level requires careful thought. Often in such designs, the restriction is imposed that only downward calls in the hierarchy are allowed; each level may call upon only those functions it sees in the level directly below.

### **3.9.4 Migration of the Nucleus to Microcode**

One definite trend that has emerged in recent systems is for the placement of much of the nucleus into microcode. This is an effective security technique, and with careful microcoding can make the nucleus functions execute more quickly.