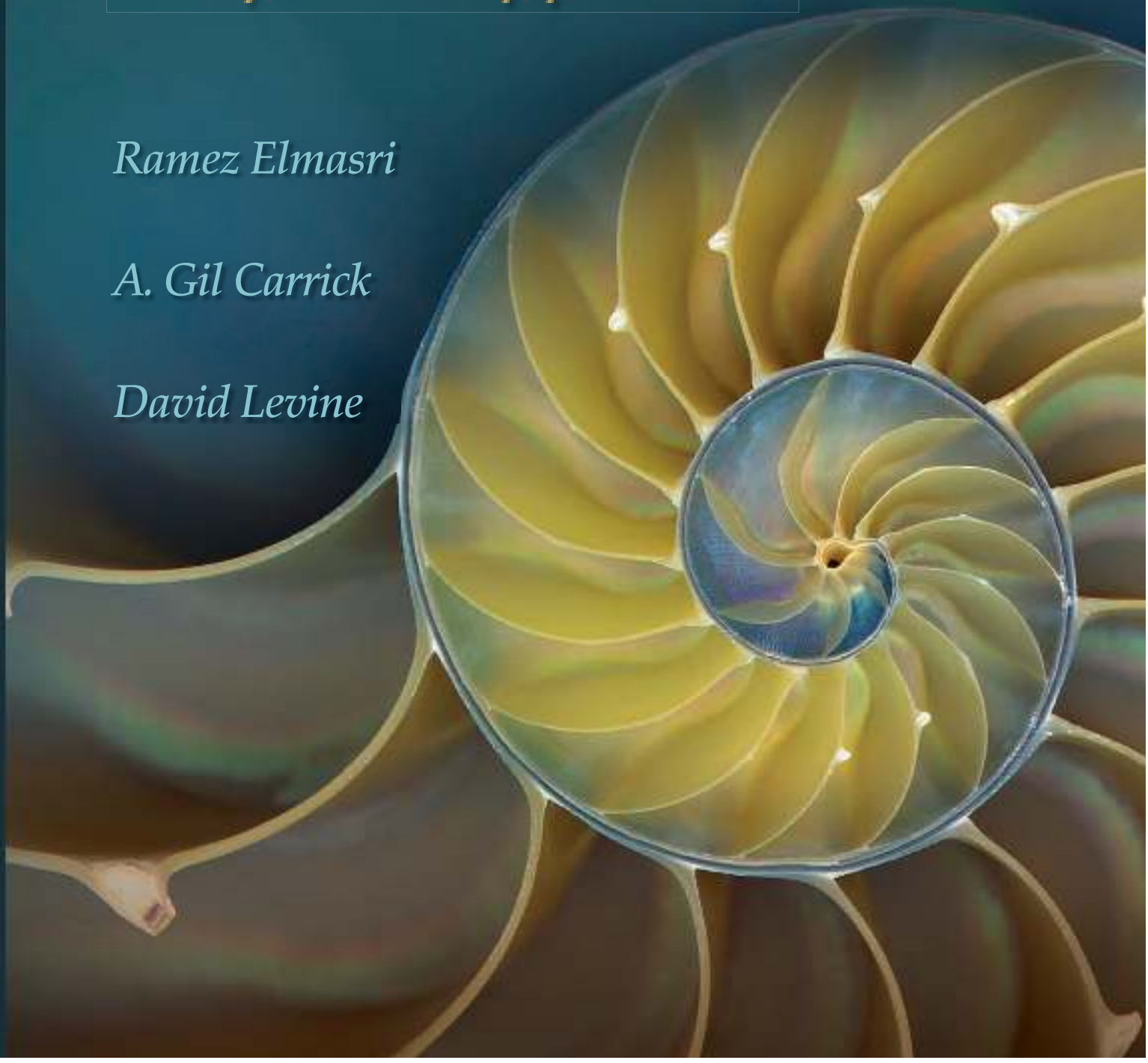
Operating Systems

A Spiral Approach

Ramez Elmasri

A. Gil Carrick

David Levine



Operating Systems: A Spiral Approach

Ramez Elmasri, Professor
University of Texas, Arlington

A. Gil Carrick, Lecturer
Formerly of the University of Texas, Arlington

David Levine, Senior Lecturer
University of Texas, Arlington

Mc Graw Hill **Higher Education**

Boston Burr Ridge, IL Dubuque, IA New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

Higher Education

OPERATING SYSTEMS: A SPIRAL APPROACH

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 9

ISBN 978–0–07–244981–5
MHID 0–07–244981–0

Global Publisher: *Raghothaman Srinivasan*
Director of Development: *Kristine Tibbetts*
Senior Marketing Manager: *Curt Reynolds*
Project Manager: *Joyce Watters*
Senior Production Supervisor: *Kara Kudronowicz*
Senior Media Project Manager: *Jodi K. Banowetz*
Associate Design Coordinator: *Brenda A. Rolwes*
Cover Designer: *Studio Montage, St. Louis, Missouri*
(USE) Cover Image: © *Getty Images*
Compositor: *Laserwords Private Limited*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley Crawfordsville, IN*

Library of Congress Cataloging-in-Publication Data

Elmasri, Ramez.
Operating systems : a spiral approach / Ramez Elmasri, A. Gil Carrick, David Levine. —1st ed.
p. cm.
Includes index.
ISBN 978–0–07–244981–5 — ISBN 0–07–244981–0 (hard copy : alk. paper)
1. Operating systems (Computers)—Textbooks. I. Carrick, A. Gil. II. Levine, David (David E.) III. Title.
QA76.76.O63E4865 2010
005.4'3—dc22

2008051735

www.mhhe.com

Dedication

"To peace, knowledge, and freedom."

—Ramez Elmasri

"To Judith, whose limited patience was strongly tested."

—Gil Carrick

"To close family and friends."

—David Levine

Table of Contents

Preface

WHY WE WROTE YET ANOTHER OPERATING SYSTEMS BOOK

We have long felt that the traditional approach to teaching about Operating Systems (OSs) was not the best approach. The purpose of this book is to support a different approach to this task. When studying any complex domain of knowledge, the order in which one learns the hierarchy of principles, laws, ideas, and concepts can make the process easier or more difficult. The most common technique is to partition the subject into major topics and then study each one in great detail. For OSs, this has traditionally meant that after a brief introduction to some terms and an overview, a student studied isolated topics in depth—processes and process management, then memory management, then file systems, and so on. We can call this a depth-oriented approach or a vertical approach. After learning a great mass of unrelated details in these isolated topic areas, the student then examined case studies, examples of real OSs, and finally saw how the different topics fit together to make a real OS.

We believe that a better model is that followed by children when learning a language: learn a few words, a little grammar, a little sentence structure, and then cycle (or spiral) through; more words, more grammar, more sentence structure. By continuing to spiral through the same sequence, the complexity of the language is mastered. We can call this a breadth-oriented or spiral approach.

We have taken this approach to the subject of OSs. The first few chapters give some basic background and definitions. We then begin to describe a very simple OS in a simple system—early PCs—and evolve toward more complex systems with more features: first limited background tasks (such as simultaneous printing), then multitasking, and so on. In each case we try to show how the increasing requirements caused each system to be designed the way it was. This is not specifically a historical order of OS development. Rather, we choose a representative system at each complexity level in order to see how the different OS components interact with and influence one another. It is our belief that this approach will give the student a greater appreciation of how the various features of each level of OS were put together.

Part of the motivation for this approach has to do with why Computing Science students are told they must study OSs at all. It is highly unlikely that many of these students will work on the development of OSs. However, virtually every system that they do work on will run on top of an OS, though perhaps a very few will work on embedded systems with no OS. For the rest of them, the OS will stand between the applications and the hardware, and failure to thoroughly understand the nature of the OS will mean that these applications will be underperforming at best and hazardous at worst. We believe that our approach will lead students to a better understanding of the entire architecture of modern OSs than does the traditional approach.

THE ORGANIZATION OF THE BOOK

In Part 1 of the book we give some general background information. This information will cover basic principles of OSs and show several different views of an OS. It will also include an overview of typical computer hardware that an OS controls. Another chapter addresses such basic concepts as processes, multiprogramming, time sharing, resource management, and different approaches to OS architecture.

Then in Part 2 of the book, we will cover five types of operating systems in increasing order of complexity, our spiral approach, as follows:

1. A simple single-process OS (CPM)
2. A more complex OS (Palm OS), which allows simple system multitasking
3. An OS with full multitasking for a single user (Apple Mac OS, pre-OS X)
4. An OS that supports multiple users (Linux)
5. A distributed OS (mostly Globus)

In each case we have selected an OS that is typical of the class on which to base the discussion so as to make it more concrete. This selection was made with an eye to practicality. We first discuss simple systems in terms of process, memory, file, and I/O management, and then (slowly) move to more complex systems, gradually introducing such concepts as multitasking, time sharing, networking, security, and other issues. Occasionally we will also mention other well-known OSs that are examples of a class, such as MS-DOS in Chapter 3 and the Symbian OS in Chapter 4.

In Parts 3–5 of the book, we move to an in-depth approach of covering each OS topic in more detail: from processes to memory management to file systems. We also discuss many recent issues in operating systems: threading, object orientation, security, and approaches to parallel and distributed systems. In these chapters we revisit the sample systems discussed in Part 2 and explain the mechanisms in more detail, especially for the modern OSs.

In Part 6 we look more closely at several OSs in what are typically called case studies. Now that we know more about the details, we look at some systems in more depth and see how some features were implemented. In two cases we are revisiting more closely OSs that were covered in Part 2.

An appendix covers basic computer hardware architecture for those institutions that do not require such a course as a prerequisite for an Operating Systems course. It can also be used as a reference for those who need to review a specific topic.

THE STYLE OF THE BOOK

- We use a conversational style to avoid boring the students with excessive pedantry.
- We avoid the use of excessive formalisms. A more formal presentation is provided where needed. This choice stems from our belief that most students will not develop OSs, but rather will use them to support applications.

- We use the normal, accepted terms but also discuss alternative terms when no accepted standard terminology exists or where other terms were used historically.
- We discuss algorithmic solutions as opposed to listing actual code since students at different schools will have been exposed to different languages.
- For each OS that is treated separately, whether in the spiral section or in the case studies, we include some history of the industry at the time, and sometimes the key companies or individuals involved. This follows from our basic belief that a student can understand OSs better if they are placed in a meaningful context.
- We cover modern OSs found in devices not conventionally regarded as computers since the students use these devices every day and have an operational familiarity with them.
- Frequent figures are incorporated as an aid to those who learn best visually rather than by reading sequences of words.
- Each chapter ends with a set of questions that a student can use to assess the level of understanding of the material in the chapter.
- Projects are outlined for many chapters, which can be used by the instructor to ground the students' understanding in reality.

The Authors

We have been teaching OS classes for quite a few years using other materials. We have developed this text because we felt the need for a different methodology. We all have served on the faculty of the Department of Computer Science and Engineering at the University of Texas at Arlington (UTA).

Ramez Elmasri is a Professor at the University of Texas at Arlington. He received his BS in Electrical Engineering from Alexandria University, Egypt, in 1972, and his MS and PhD degrees in Computer Science at Stanford University in 1980. His current research interests are in sensor networks and RFID, mediators for bioinformatics data, query personalization, and systems integration. He is the lead co-author of the textbook "Fundamentals of Database Systems," now in its 5th Edition. His previous research covered various aspects of databases, conceptual modeling, and distributed systems.

A. Gil Carrick was formerly a Lecturer at UTA and is now retired from teaching. He received his BS in Electronics Technology from the University of Houston in 1970 and his MSCS in 2000 from the University of Texas at Arlington. He is a member of Upsilon Pi Epsilon, the Computer Science Honor Society. His career spans the information technology industry: end-user organizations, hardware manufacturers, software publishers, third-party maintenance, universities, and R&D firms. He has written for professional journals and edited IT books, primarily in the networking field. In his career he has used all the operating systems discussed in this text and many others besides.

David Levine has been teaching courses in operating systems, software engineering, networking, and computer architecture. His research interests include mobile computing, mobile objects, and distributed computing and he has presented the results of this research in recent publications and several international conferences. He enjoys discussing Operating Systems, talking about current research with students and reading about new OS advances.

HOW TO USE THIS BOOK—FOR INSTRUCTORS

This text is intended to be used for a one-semester undergraduate course in Operating Systems, probably in the junior or senior year. The first part of the book is designed to consolidate basic background information necessary for the following chapters. Chapter 1 sets the discussion and gives some historical perspective. The instructor can

skim this chapter and decide what to include. The appendix is a brief look at fairly modern hardware architectures. If a course in hardware is not a prerequisite for this course, then this appendix could be included. Chapter 2 defines some simple terms used in OSs and offers some more perspective on the larger topic of OS design. Again, an instructor can review this chapter and select different parts to include or exclude.

Part 2 begins the spiral approach. We believe this is a significant portion of the book. Here the student is gradually introduced to a series of OSs with more complex goals. These increasingly more complex goals lead to increasingly more complex OSs. Only two of these chapters are not normal topics in OS texts—Chapter 4 on a single-user multitasking operating system and Chapter 7 on a distributed operating system. They could be left out at the instructor's discretion, but more and more students will be working in such environments as users and as programmers.

Part 3 begins the in-depth chapters. Each chapter is fairly independent of the others, though Chapters 12 and 13 are strongly related. Beginning with Chapter 14 the individual chapters can probably be left out if the topic is the major subject of another course that the students will be required to take.

Notes about the bibliographies: The chapters in Part 3 all include a bibliography section. The reference papers that are cited are widely regarded as being seminal papers or good summaries. They may cover material that is not covered in the text. If an instructor or a student is looking for material to provide a better understanding of a given topic, then they are suggested reading.

HOW TO USE THIS BOOK—FOR STUDENTS

For students the most important thing about using this text is to understand how one learns best. There are many pathways to get information into the brain. The book itself directly addresses two of these pathways. There is obviously the text for those who learn best through reading the words and the illustrations for those who are more visually oriented. When you attend the lectures you will hear the instructor talk about the material. This is for those who learn best through hearing words. At the same time, the instructor will probably use visual aids such as the PowerPoint slides that are available for the text. Again, this is to the benefit of those who learn best by reading the words and seeing the illustrations. Some students learn best from mechanical skills, so the process of outlining the material or making study notes works well for those students.

Also presented in the book at the end of each chapter are questions about the material. These questions are designed such that a student who has a reasonable grasp of the material should be able to answer the question.

As new information is presented to the brain it takes a certain amount of time to link with other information already there. But the brain gets much information during the day that is not significant and therefore it does not retain it. Only when presented with the same or similar material again a short time later will the brain retain a significant amount of the information. The more different mechanisms that are used and the more times the information is repeated, the stronger the retention of the material. So the best method is to use all these methods combined, focusing

on what works best for you. What we have found works well for most students is the following sequence:

- Print the slides to be covered in the next section, with several slides per page.
- Read the assigned material in the text. Note questions on the slide printouts.
- Come to class and listen to the instructor, amplifying any notes, especially things the instructor says that are not in the text. (Those points are favorite issues for the instructor and they tend to show up on exams.)
- Ask questions about things that are unclear.
- When it is time to review the material for an exam, go over the slides. If there are points that are unclear, go back to the text to fill them in. If any questions remain, then contact the instructor or teaching assistants.
- The review questions can be studied at any time the student finds convenient.

AVAILABLE RESOURCES FOR INSTRUCTORS

The text is supported by a website with separate sections for instructors and students.

- Supplements to the text will be made from time to time as the need presents itself.
- A set of suggested projects will be available for instructors. Most of these projects will have been used by the authors. They should be sufficiently rich and OS independent that they can be readily adapted to fit any situation. They are not based on any specific package that the instructor, students, or assistants will have to master in order to work the labs.
- PowerPoint slides are provided for the students to use, as described earlier. Instructors are encouraged to modify these presentations to fit their needs. Acknowledgement of their source is requested.
- Review question answers are provided for the instructors in order that they not be embarrassed by not knowing some arcane point the authors thought was important.
- A current list of errata will be maintained on the website.
- Reference to web resources are provided for many chapters, but the web is very volatile. The website for the book will contain an up-to-date set of web references.

ACKNOWLEDGMENTS

This text has actually been developing for longer than we would like to remember. The people at McGraw-Hill have been exceptionally patient with us. In particular, we would like to thank the following folks with McGraw-Hill: Melinda Bilecki, Kay Brimeyer, Brenda Rolwes, Kara Kudronowicz, Faye Schilling, and Raghu Srinivasan. We would also like to thank Alan Apt and Emily Lupash, who were our editors when we started working on the book. Finally, we also thank Erika Jordan and Laura Patchkofsky with Pine Tree Composition.

The chapter on Windows Vista was reviewed by Dave Probert of Microsoft. He provided valuable feedback on some items we had only been able to speculate on and brought several problems to our attention. His participation was arranged by Arkady Retik, also with Microsoft Corporation. Two chapters were reviewed by our fellow faculty members at University of Texas, Arlington. These included Yonghe Liu who reviewed the chapter on networking and Matthew Wright who reviewed the chapter on protection and security. Another faculty member, Bahram Khalili, used drafts of the text in his OS class. Naturally any remaining problems are our responsibility and not theirs.

We have used drafts of these materials in our teaching for several years and we wish to thank all our students for their feedback. In particular we wish to thank the following students: Zaher Naarane, Phil Renner, William Peacock, Wes Parish, Kyle D. Witt, David M. Connelly, and Scott Purdy.

REMAINING ERRORS

One difficulty with working on a project with multiple authors is that with the best of intentions, one of the writers can alter a bit of text that he himself did not write, thinking that he is clearing up some minor point, but actually altering the meaning in some subtle but important way. Accordingly, you may find minor errors in the text. Naturally these errors were not the fault of the original author, who doubtless wrote the original text correctly, but were introduced by another well-meaning author who was not as familiar with the material.

Still, such errors may be present, and we must deal with them. So, if you do find errors, we would be very happy to know about them. We will publish any errata, fix them in the next edition, determine who is to blame, and deal with the offending authors appropriately.

Part 1

Operating Systems Overview and Background

In this part:

This part of the book contains two chapters. Chapter 1 gives a basic explanation about what an Operating System (or OS for short) is. It explains how the OS provides services to users and programmers. These services make it possible to utilize a computer without having to deal with the low-level, arcane details, but rather, being allowed to concentrate on the problem(s) to be solved. Such problems may be anything, including not only the things we normally consider computing activities, but also activities such as playing games, dynamically generating art, and monitoring the performance of an automobile engine.

Chapter 2 provides an initial high-level look at OS concepts, components, and architecture. General terms are introduced that a student will need to know in order to study the series of increasingly more complex OSs that are presented in Part 2.

Chapter 1

Getting Started

In this chapter:

Operating systems are at the heart of every computer. The **Operating System** (or **OS** for short) provides services to users and programmers that make it possible to utilize a computer without having to deal with the low-level, difficult-to-use hardware commands. It provides relatively uniform interfaces to access the extremely wide variety of devices that a computer interacts with, from input/output devices such as printers and digital cameras, to wired and wireless network components that allow computers to communicate. It allows users to create, manage, and organize different types of files. In addition, most modern OSs provide graphical user interfaces (GUIs) to allow a relatively easy-to-use interface for computer users.

In this opening chapter, we start in Section 1.1 with a brief introduction to show how important an Operating System is and how they are used not only in computers but also in many types of electronic devices that we all use in our daily routines. Section 1.2 is a more technical look at why even simple devices contain an Operating System. Then in Section 1.3 we discuss the different views of what an Operating System does by looking at the Operating System from two perspectives: the user's perspective and the system's perspective. We also discuss the requirements that each type of user has for the Operating System. Section 1.3 next gives a few simple examples to illustrate some sequences of functions that an Operating System goes through to perform seemingly simple user requests. In Section 1.4 we present some basic terminology and concepts, and give some figures to illustrate typical components for a simple Operating System. We give a brief historical perspective in Section 1.5 and conclude with a chapter summary in Section 1.6.

1.1 INTRODUCTION

For many years, OSs were viewed by most people as uninteresting—except for OS programmers and computer "nerds." Because of a number of high-profile cases, OSs have occasionally become front-page news in recent years. Suddenly, the OS is seen by some as controlling all computing. There are very strongly felt opinions about what constitutes good versus bad OSs. There is also quite a bit of disagreement about what functionality should be provided by the OS. While many people (and some courts!) believe that one company dominates the OS market, others say that the OS is increasingly unimportant—the **Internet browser** *is* the OS. In fact, there is a very wide variety of types of OSs, and OSs exist at some level on every conceivable computing device, including some that may surprise many people.

For example, handheld personal digital assistants (**PDA**s) have very capable, complex, and flexible OSs. Most electronic devices that have some intelligence have complex, yet easy-to-use OSs and system software to control them. The OS that was once thought of as the arcane world of process management and memory management techniques is now occasionally a conversation topic in cafés, bars, and computer stores. Many people now seem to be experts—or at least have an opinion—on OSs.

(Perhaps) Surprising places to find an OS:
Personal digital assistants
Cable TV controller boxes
Electronic games
Copiers
Fax machines
Remote controls
Cellular telephones
Automobile engines
Digital cameras

While we also have our opinions, we try to get behind the hype—generated by marketing and salespeople as well as millions of opinionated users—in order to explain the real systems. We also throw in our own opinions when needed and explain why we hold these beliefs. We give many examples of currently used systems to demonstrate concepts and show what is good and bad about the various systems. We try to avoid the so-called religious issues, such as: Which is the better OS: **Windows** or **Mac-OS?** Or are **UNIX** and its variations such as **Linux** better than both? Instead, we point out how these systems came about and what they provide to users and programmers.

Increasingly, certain parts of the OS—particularly those handling user and application program interaction—are visible to users and programmers and often may be critical in marketing a computer or electronic—or even mechanical—device. Buyers are becoming very critical and have higher expectations of what the OS should provide them. More than ever before, the system must not only provide new features and be easier to use but it must also support those old features and applications that we are used to. Of course, as we add new devices—video devices and disks, high fidelity sound, and wireless networking, for example—we want the system to easily adapt to and handle those devices. In fact, a good OS architecture should even allow the connection of new devices that were not yet available and may not even have been thought of when the OS was created!

1.2 WHAT ARE OPERATING SYSTEMS ALL ABOUT?

In this section, we give a simple example—a simple handheld game system—to illustrate some of the basic functionalities that an OS should provide.

Think about a handheld electronic game system, one that is very cheap but has a small screen, a few buttons, and several games. Although this game system might not require an OS, it probably has one. The main reason is to consolidate the common functions needed by the various games installed on the game system.

The games typically have some common parts. For example, each game needs to get some input from the buttons, and to display something on the screen. While those actions sound easy, they do require some not-so-simple software programming. Getting the input from a button—that sounds easy. Well, except that the user may push two buttons at once—what then? It is also likely that a cheap game does not use sophisticated and expensive buttons, so there is electronic noise that may distort the signal coming in—how should the games deal with that? The easy solution is to handle each of these common issues in one, single place. For example, all button pushes can be read in, have any noise cleaned up, and so forth in a single software routine. Having a single *read-the-button* software routine has the advantage of providing a consistent user interface—all games treat button input in the same way. It also allows the routine to occupy space in only one place in system memory instead of occupying space in each individual game. And where should that *read-the-button* software routine be placed? It should be in the OS—where every game that needs to read a button can call this routine.

The OS should also handle unexpected events. For example, a user may quit a game in the middle (when losing) and start another game. No reboot of the game system should be necessary. The user's need to switch from game to game (task to task) is natural and expected. In fact, users (5-year-olds) may push buttons at unexpected times and the screen should continue to be updated (refreshed) while the game is being played—even while waiting for a button to be pushed. This is called **asynchronicity,** which can be defined informally as the occurrence of events at random or unexpected times—a very important feature in even simple systems like a handheld game.

Several important OS concepts are part of this game system: When a game is started, some part of its software may be loaded into memory, whereas other parts

may have been preloaded in ROM (read-only memory) or fixed memory[1]; dynamic memory is reserved for use by the game and is initialized; timers may be set. All on a cheap (but fun) game! What more does one expect from an OS?

1.3 USER VERSUS SYSTEM VIEW OF AN OS

You have probably heard the old adage; "There are two sides to every question." (Maybe that should be "two *or more* sides.") The idea is that trying to look at some question from different perspectives often helps our understanding. One of the important methods to learning something new is to view it from different perspectives. For an OS, the two most important perspectives are the **user view** and the **system view.**

The user view pertains to how users or programs—programs are the main users of the OS—utilize the OS; for example, how a program reads a keystroke. The system view pertains to how the OS software actually does the required action—how it gets keystrokes, separates out special ones like *shift,* and makes them available to the user or program. We present OS facilities, concepts, and techniques from both user and system points of view throughout the book. First, we elaborate on the different types of users and their views of the OS.

1.3.1 Users' views and types of users

The term **user** is often too vague—especially for persons whose role in computing is so critical—so it is important to first describe the various types of users. Trying to pin down the role of a user of an OS is not simple. There are various types of users. We primarily want to distinguish among end users, application programmers, system programmers and system administrators. Table 1.1 lists some of the most important concerns about what the OS should provide for each of the three main types of users. Of course, there is some overlap among these concerns. We are merely trying to show how those viewpoints sometimes diverge. Further complicating the issue is that sometimes users fit into several of the roles or even all of them. Such users often find themselves having conflicting needs.

Application Users (or End Users)—this group includes all of us, people who use (or run) application or system programs. When we use a word processor, a web browser, an email system, or a multimedia viewer, we are a user of that application. As users, we expect a quick, reliable response (to keystrokes or mouse movement), a consistent user view (each type of command—such as scrolling or quitting an application—should be done in a similar manner), and other features that depend on each specific type of OS. Other needs are listed in Table 1.1. In general, this group of users is most often called simply *users,* or sometimes **end users.**

Application Programmers—this group includes the people who write application programs, such as word processors or email systems. Programmers are very demanding of the OS: "How do I read and write to a file?", "How do I get a user's keystroke?", and "How do I display this box?" are typical questions programmers

[1] We define these terms in Chapters 2 and 3.

TABLE 1.1 Concerns of Various User Classes

End Users	Easy to use and learn Adapts to user's style of doing things Lively response to input Provides lots of visual cues Free of unpleasant surprises (e.g., deleting a file without warning) Uniform ways to do the same thing (e.g., moving an icon or scrolling down a window—in different places) Alternative ways to do one thing (e.g., some users like to use the mouse, others like to use the keyboard)
Application Programmers	Easy to access low-level OS calls by programs (e.g., reading keystrokes, drawing to the screen, getting mouse position) Provide a consistent programmer view of the system Easy to use higher-level OS facilities and services (e.g., creating new windows, or reading from and writing to the network) Portability to other platforms
Systems Programmers	Easy to create correct programs Easy to debug incorrect programs Easy to maintain programs Easy to expand existing programs
System Managers and Administrators	Easy addition or removal of devices such as disks, scanners, multimedia accessories, and network connections Provide OS security services to protect the users, system, and data files Easy to upgrade to new OS versions Easy to create and manage user accounts Average response is good and predictable System is affordable

ask when learning to use a new OS. The facilities that the OS provide are the programmers' view of the OS. Sometimes they are called system calls or an API (application program interface). They may also appear as language library functions or sometimes just as packages of classes. Programmers also want the software they develop to be easily ported to other platforms.

Systems Programmers—these are the people who write software—either programs or components—that is closely tied to the OS. A utility that shows the status of the computer's network connection or an installable driver for a piece of hardware are examples of systems programs. Systems programmers need to have a detailed understanding of the internal functioning of the OS. In many cases, systems programs need to access special OS data structures or privileged system calls. While OS designers sometimes are concerned with portability to other platforms, often they are not—they are charged with developing a specific set of functions for a specific platform and portability is not a concern.

System Administrators—this group includes the people who manage computer facilities, and hence are responsible for installing and upgrading the OS, as well as other systems programs and utilities. They are also responsible for creating and managing user accounts, and for protecting the system. They need to have a detailed understanding of how the OS is installed and upgraded, and how it interacts with other programs and utilities. They must also understand the security and authorization features of the OS in order to protect their system and users effectively.

1.3.2 System view

The system view refers to *how the OS actually provides services.* In other words, it refers to the internal workings of the OS. This is a less common view. Often only a few people, the OS designers and implementers, understand or care about the internal workings of an OS. Indeed this information is often considered secret by companies that produce and sell OSs commercially. Sometimes the overall workings of major parts of the system—management of files, running of programs, or handling of memory—may be described to help programmers understand the use of those subsystems. In some cases, the whole source code for an OS is available. Such systems are known as **open source** systems.[2]

The majority of this book is concerned with the *how*—how does the system run a program, create a file, or display a graphic. To understand the actual "how"—the internal details—we describe algorithms and competing methods for implementing OS functions. We now illustrate the system view (or views) with two examples: tracking mouse and cursor movement, and managing file operations. Although these examples may seem a bit complex, they serve to illustrate how the OS is involved in practically all actions that are performed by a computer user.

1.3.3 An example: moving a mouse (and mouse cursor)

While the movement of a mouse pointer (or cursor) on a screen by moving the mouse (or some other **pointing device** such as a pad or trackball) seems straightforward, it illustrates the many views of an OS. Figure 1.1 illustrates this process. When the pointing device is moved, it generates a hardware event called an **interrupt,** which the OS handles. The OS notes the movements of the mouse in terms of some hardware-specific units—that is, rather than millimeters or inches the readings are in number of pulses generated. This is the **low-level system view.** The actual software reading the mouse movements is part of the OS, and is called a **mouse device driver.** This device driver reads the low-level mouse movement information and another part of the OS interprets it so that it can be converted into a **higher-level system view,** such as screen coordinates reflecting the mouse movements.

On the "other side" or view is the question, What does the user see? The **user's view** is that the cursor will smoothly move on the screen and that as the mouse moves greater distances faster, the screen movement will appear faster too. In between these

[2] The Linux OS is a well-known example of an open source operating system.

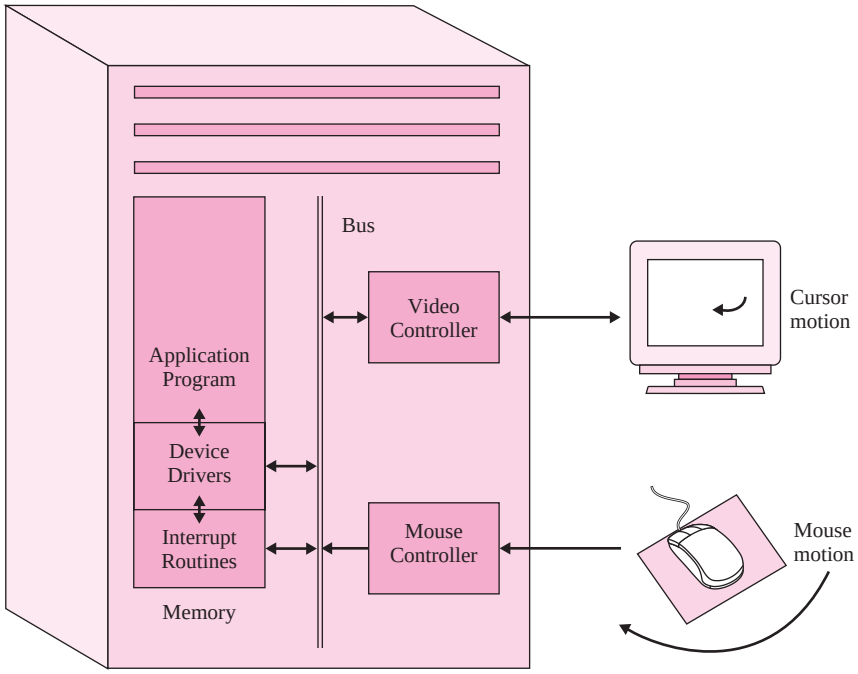


FIGURE 1.1
The cursor tracking mouse motion.

views is the **application programmers' view,** How do I get the mouse movement information in order to use it and display it in my application? Another issue is how this information on mouse movements is presented to the application programmer. This is the higher-level system view mentioned earlier.

And to complete these views a bit let us return to the system's view, Which application gets this mouse movement if there are multiple open windows? The mouse movements may need to be queued up if there are multiple movements before the application retrieves them. The movements may even be lost if the OS is busy doing other things—for example, loading a Web page through a network connection—and cannot receive the device driver's input in a timely manner.

1.3.4 Another (bigger) example: Files

Sometimes the most critical **end user's view** of an OS is the file system—in particular, file names. Can file names contain spaces? How long can they be? Are upper- and lowercase letters allowed? Are they treated as different or the same characters? How about non-English characters or punctuation? An OS may even be called good or bad simply because long file names are not used or the difference between upper- and lowercase characters is not distinguished.

In the **application programmer's view,** the file system is a frequently used, critical part of the system. It provides commands for creating a new file, using an existing file, reading or appending data to a file, and other file operations. There may even be several different types of files provided by the system. The **system**