CHAPTER 15

OPERATING SYSTEMS: AN OVERVIEW

COMMANDER RIKER, PLEASE NEGOTIATE A SECURE CONNECTION TO THE INTERNET.

AYE-AYE, SIR.

MR. LAFORGE, PLEASE PLOT A COURSE FOR MY PERSONAL HARD DISK SECTOR.

RIGHT AWAY, SIR.

COUNSELOR TROI, PLEASE UPLOAD DATA, UTILIZING FTP PROTOCOL. DID YOU MEAN COMMANDER DATA, SIR?

YOU WEREN'T KIDDING ABOUT THIS BEING A "NEXT GENERATION" OPERATING SYSTEM.

WESLEY, EMPTY THE TRASH.

GRumble.

FOXTROT © 2001, Bill Amend. Reprinted with permission of UNIVERSAL PRESS SYNDICATE. All rights reserved.
15.0 INTRODUCTION

In Chapter 2, we introduced the modern computer system as a synergistic set of components that work together to make the computer accessible and productive to the user. The operating system (OS) software component provides the basic functionality of the system by offering programs that operate, control, and support the fundamental resources of the computer. Those resources include both CPU and peripheral hardware, network services, application programs, short-term program and data storage for use while a program is executing, time in which to execute programs, and overall access to the system. The operating system programs make system resources available to the user(s), the user’s application programs, and to other application programs running on the computer. The operating system also provides and controls access to other, interconnected systems through its networking and clustering capabilities. Although the operating system programs are tailored to the specific hardware provided on a particular system, it is possible to offer different operating systems on a particular hardware platform and to offer the same operating system on different hardware platforms.

The hardware and the operating system operate together architecturally to form a complete working individual computer environment. The operating system has two fundamental purposes: to control and operate the hardware in an efficient manner and to allow the “users” powerful access to the facilities of the machine by providing a variety of facilities and services. (For this discussion, we will define “users” loosely to include server requests from networked clients on other machines as well as users directly accessing the machine.) These services are available both directly to the users and to the programs that the users execute. In addition, the operating system expands the capability of the computer system to allow for the concurrent processing of multiple programs and support for multiple users, both local and networked, as well as other specialized tasks that would not be possible otherwise. The operating system also makes possible the synergistic implementation of specialized hardware that is designed to improve system performance and capability. The primary example of this, virtual storage, is introduced in Chapter 18.

This chapter provides an overview of the various components, facilities, and services of the operating system. We explore the services that an operating system can provide and show how the operating system integrates these services into a unified working environment. We introduce the tasks that the operating system performs and show how these tasks are interrelated and work together to make it possible for users to get their work done more efficiently.

There are many different types of operating systems, reflecting different purposes and goals, and many different methods of organizing operating systems. These differences are indicated by the way in which the user interacts with the system—an idea that is often surprising to the user who has worked with a single system. This chapter discusses various types of systems and organizations. It notes
PART FIVE  THE SOFTWARE COMPONENT

the different ways in which work is accomplished on a computer system and the different services that are provided.

Although the focus of this book is on IT systems, we note at the outset of this discussion that the use of computer-based operating systems is not limited to business systems or other obvious computer-based devices. Mobile phones, home theater systems, TV sets and DVD players, automobiles, digital cameras, electronic toys, even many household appliances, all rely on computers with their associated operating systems to provide their functionality. The material in this chapter applies, to lesser or greater extent, to all of these. Interestingly enough, the convergence of these diverse areas has, itself, had a major impact on the design and use of business systems.

15.1 THE BAREBONES COMPUTER SYSTEM

Consider once again the Little Man model that we introduced in Chapter 6. To use this model, a single program was stored in memory. The Little Man executed the program by executing each instruction in turn until he encountered a **HALT** instruction, which stopped the computer. For simplicity, the Little Man scenario was designed to ignore several issues that must be considered in a real computer.

First, we assumed that the program was already loaded into memory, without considering how it got there. In a real computer, the contents of RAM are destroyed when power is shut off. When power is again turned on, the contents of memory are initially unknown. Means must be provided to load a program when the machine is turned on. Remember that the CPU simply executes whatever it finds in memory as instructions, so there must be a program in memory before the computer can even begin to execute instructions. After the computer is on, there must be a method to load a program into memory any time a new program is to be executed.

Second, there must be a means to tell the computer to start executing the instructions in a program. The Little Man began executing instructions whenever the location counter was reset to zero.

Third, the barebones computer has no user interface except for the I/O routines that are provided with the executing program. This means that common program requirements such as keyboard and screen I/O, file operations, interrupt capability and other internal facilities, and printout must be created and supplied as a part of every program written. It would be dangerous for programs to share disks because there would be no way to establish and protect ownership of particular space on a disk.

The most important consideration to remember is that once the computer is running, it will continue to execute instructions until a **HALT** instruction is encountered or until power is removed. Halting a program at its conclusion means restarting the computer. This suggests that it is highly desirable that there be an additional program in memory that is always available to execute instructions whenever no other program is being run. This would allow programs to complete execution without halting the machine. Instead, a program would terminate with a jump instruction to the alternative program. The alternative program could be used to accept user commands and to provide a memory loader for the execution of other programs.

As a final consideration, notice that the barebones computer is limited to one program at a time. To run multiple programs concurrently, each program must be in memory and
there must be a method in place for sharing the time to execute instructions in the CPU. Since the barebones computer offers no provisions for the functions required to handle the memory management and time scheduling needed, multitasking—the execution of multiple programs concurrently—is not possible. Sharing of the computer by multiple users is also not possible, for the same reason. In Chapters 8 and 9 you were made aware of the CPU time wasted during I/O transfers that could be used by other programs. An even more important waste of time occurs as a program waits for user input. The barebones computer is not capable of using the CPU productively during these intervals.

Behind these considerations is the realization that ultimately the purpose of the computer is to help the user to get work done. Obviously, modern computers are not meant to be operated in a barebones fashion like the Little Man. The user should be able to start and operate the computer easily, should be able to choose programs to load and execute, should be able to communicate with others users and other systems, and should be able to perform these operations in a convenient, flexible, and efficient manner. Larger computer systems should be able to share the resources among many users. What is required is additional programs that can provide services to make these expanded capabilities possible.

**15.2 THE OPERATING SYSTEMS CONCEPT: AN INTRODUCTION**

The solution to the limitations of a barebones system is to include programs with the computer system that will accept commands from the user and that will provide desired services to the user and to the user’s programs. These included programs are known collectively as an operating system. The operating system acts as a system manager, controlling both hardware and software and acting as an interface between the user and the system. The operating system itself consists of a collection of programs that work together collectively to accomplish these tasks.

An operating system may be defined as

a collection of computer programs that integrate the hardware resources of the computer and make those resources available to a user and the user’s programs, in a way that allows the user access to the computer in a productive, timely, and efficient manner.

In other words, the operating system acts as an intermediary between the user and the user’s programs and the hardware of the computer. It makes the resources available to the user and the user’s programs in a convenient way, on the one hand, and controls and manages the hardware, on the other.

Intuitively we think of a user as a human interacting with a computer system; however, there are situations in which the “user” is actually another computer or a mechanical or electronic device of some sort. A common example of this situation is one in which an application program on one computer requests services from an application program or system service on another machine, for example a Web server application requesting data from a back-end database server. Another example would be a situation in which a user on a client machine requests file or printer services on a server machine.
In serving as an intermediary between the users of computer services and the computer’s resources, the operating system provides three basic types of services:

1. It accepts and processes commands and requests from the user and the user’s programs and presents appropriate output results.
2. It manages, loads, and executes programs.
3. It manages the hardware resources of the computer, including the interfaces to networks and other external parts of the system.

The relationship between the various components of a computer system is shown schematically in Figure 15.1.

In its intermediary role, the operating system makes it possible for users and programs to control the computer hardware transparently without dealing with the details of hardware operation. Programs can be executed and controlled with mouse clicks and keyboard commands and other types of input. When programs are completed or interrupted, control returns to the operating system, enabling the user to continue to operate without restarting the computer.

Effectively, the operating system provides a complete working environment, making the system convenient for the user by providing the services necessary to get work done.

The easiest way to think of an operating system is to consider it as a master “program” that accepts requests from the user, the user’s programs, or other sources, and then calls its own programs to perform the required tasks. At the same time, it also calls programs to control and allocate the resources of the machine, including the use of memory, the use of I/O devices, and the time available to various programs. Thus, if the user issues a command to load a program, a program loader is executed, which then loads the desired program into memory and transfers control to the user’s program to run. That program can then issue its own requests, for example to produce output to a printer or to send a message through the Internet to a Web server somewhere.

If you like, you could picture a command-interpreter-and-program-loader program sitting at the high end of the Little Man Computer memory. When a particular value is received as input, say, 999, that corresponds to the user’s command to load a program, the loader performs a loop that inputs the instructions one at a time from the input box.
into lower memory and then jumps to mailbox 00 to execute the new program. (See Exercise 15.14.)

In a real computer, the operation is more complex, of course. There are many different I/O devices to be controlled, for one thing. There will usually be more than one program sharing the hardware resources, for another. To accept a command from a user, the operating system must first service mouse clicks and input keystrokes from the keyboard. It must interpret these actions, for example, as a command that requests that a program be loaded and executed. It must provide a file system that can interpret the name of the program being requested and determine the location of the file, first by determining the secondary storage device to be used and then by locating the file on the device. It must read the appropriate blocks from the device into memory. Only then can the operating system transfer control to the program being executed.

Modern computer systems enable users to work with more than one program at the same time as a way to improve their efficiency. A user can be listening to music on the Web while word processing a document (which is what the author is doing at this moment). A programmer can be editing one program while compiling another. Nearly every modern system provides means and support for manipulating multiple programs, even on a system with only a single CPU. This technique is known as **multitasking** or **multiprogramming**.

Since a system may be manipulating many tasks on a computer with one or a few CPUs, the operating system must support **concurrency**, which simulates the simultaneous execution of multiple programs to provide multitasking and multiuser support. To support concurrency there will be additional requirements: programs to allocate memory and other computer resources to each program, programs to allot the CPU time in an equitable way to each program, programs to direct input and output appropriately, and programs to maintain the integrity of each program, to name a few.

Multitasking also enables multiple users to share the computer resources of a single system. Such a system, known as a **multiuser** system, would still have to be multitasking, of course, because each user on the system would be running at least one program, and might even be running several programs concurrently.

This suggests that most operating systems will include additional services that augment the basic operating system services to be provided. These additional services include one or more interfaces that simplify the user’s ability to interact with the system and standardize the system’s I/O operations. Modern operating systems also provide the necessary tools to facilitate the sharing of the system services and resources among multiple programs, computers, and users. Typically, an operating system provides most or all of the following capabilities:

- The operating system provides interfaces for the user and also for the user’s programs.
- It provides file system access and file support services.
- It provides I/O support services that can be used by every program.

---

1Note that even though operating systems commonly refer to executing programs as processes, multiprogramming is not the same as multiprocessing. The latter refers to the presence of multiple CPUs within the system.
It provides a means for starting the computer. This process is known as **bootstrapping** or **Initial Program Load (IPL)**. The word *bootstrapping* is often abbreviated simply to *boot* or *booting*. (An explanation of bootstrapping is provided in Chapter 18, Section 18.2)

- It handles all interrupt processing, including error handling and recovery, as well as I/O and other routine interrupts.
- It provides services for networking. Most modern systems also provide services to support symmetric multiprocessing, clustering, and distributed processing. Where necessary, the operating system may also provide support for special features of the system. For example, the operating system for a Sony Playstation 3 must support the asymmetric multiprocessing that is a principal feature of the Cell multiple CPU processor used within.
- The operating system provides services that allocate resources, including memory, I/O devices, and CPU time to programs as they need them.
- It provides security and protection services: specifically, program and file control services to protect users’ programs and files from each other and from outsiders, as well as to make communication between programs possible, when desired.
- It provides information and tools that can be used by the (human) system administrator to control, tailor, and tune the system for appropriate behavior and optimum performance.

Figure 15.2 is a simplified diagram showing the relationships between the different components of an operating system. The diagram focuses on the interactions among the most user-visible services. Specific multitasking and bootstrapping components are not shown. These are part of the core services, which also include process and thread management, resource allocation, scheduling, memory management, security, and interprocess communication. Also not shown on the diagram, many operating systems allow programs to call the command interface directly to execute commands. Thus, a C++ program operating under Linux could issue a Linux command as part of its processing.

The diagram also shows the command interface as part of the operating system. In some systems, this is not quite the case. Instead, the command interface is viewed as a **shell** outside of the operating system per se. As you will see, this view can result in increased user flexibility by allowing the user to select different shells for different types of tasks.

Since the programs that make up the operating system occupy space in memory that might be needed for application programs, the operating system is commonly divided into resident and nonresident parts. Some operating system services are critical to the operation of the system and must be resident in memory all the time. Others can be loaded into memory only when they are needed, and executed just like other programs.

The critical programs are loaded into memory by the bootstrap loader at start-up time and will remain resident as long as the computer is running. The bootstrap for most modern computers is stored in read-only memory; on some computers, part or all of the resident operating system will also be contained in ROM, so that it is permanently resident in memory and always available for use. This is particularly true for operating systems embedded into electronic devices such as mobile phones or DVD players.

The memory resident components of an operating system are commonly known as the **kernel** of the operating system. For example, the operating system program that accepts user commands must always be present whenever the machine is operative, as
must the routines that handle interrupts and manage commonly used resources in a multitasking system. On the other hand, an operating system command that formats a new disk is only used occasionally; it can be loaded and executed only when it is required.

Most people assume that the operating system software for a conventional computer system is stored on a disk that is connected directly to the computer, but this is not necessarily true. If the computer is attached to a network, it may obtain its programs, including the operating system, from another computer on the network. This has led to the concept of the **diskless workstation**, a personal computer that, once booted, relies completely on the network for its data and program storage and access. Diskless workstations are also known as **thin clients**.

The size of the kernel and the particular services provided within a kernel vary from operating system to operating system, depending on the organization and capabilities of the system, as well as by the type of system. Some operating system vendors define the kernel more narrowly than others, precluding from memory residency some components that are deemed less critical to the basic operation of the system. Thus, the kernel in one system may be small, with only the most critical components included, and another might be large, with a tremendous range of services.

There are many different types of operating systems, some tailored for very specific purposes, but general-purpose computing systems can be loosely divided into categories, as follows:

- Single-user, single-tasking systems (this category is essentially obsolete)
- Single-user, multitasking systems
- Mainframe operating systems
- Operating systems for mobile devices
- Distributed systems
- Network servers: Web servers, database servers, application servers, and the like
- Embedded systems, such as those found in medical instruments, basic cellphones, automobile control systems, marketplace kiosks, household appliances, DVD players and TVs, electronic toys, and the like
- Real-time systems, used for instrumentation where system responses are time sensitive

Not surprisingly, these categories are somewhat arbitrary and are not mutually exclusive. Indeed, there is a lot of overlap between the various categories. For example, the embedded computer system that controls the braking system for an automobile must
obviously be capable of real-time response when the driver of the car slams on the brakes in an emergency.

Systems can also be categorized by the degree of activity between the user and the system during program execution. As a student, you are probably most familiar with interactive systems. When the system is interactive, the user interacts directly with the program to provide input data and guidance during program execution. Interactive systems are sometimes known as conversational systems. Most personal computing is done interactively.

Many business tasks are performed more effectively in a batch, where the data input for the program is collected together into a file on disk or tape. It does not make sense to have a user enter data one record at a time if an entire set of data is to be processed into monthly credit card bills, for example. Instead, the user submits the program(s), or job(s), to the computer for processing. This type of processing is known as batch processing. The user does not interact with the program during batch processing. Large-scale billing, payroll, and other similarly data intensive systems are usually processed this way.

We remind you that a CPU can execute only one instruction at a time; therefore, time used by the operating system on a single CPU system is not available for the execution of user programs. In general, the time used by the OS program is considered overhead. In reality, though, the operating system actually saves time for the users in most situations:

- In a single-user system, the operating system program creates minimal overhead. While the OS program is available to the user at any time, the executing user programs have priority; the OS program runs only to distribute CPU time among executing programs, or to handle interrupts, or if the users’ programs request its services.
- The operating system program performs tasks directly for the user that would otherwise have to be performed, with more difficulty, by the user. This includes the various commands available to the user and I/O services to the user’s programs. Most important, this includes the loading and execution of programs. When a user program is not being executed, the OS is always available to the user for these purposes.
- The OS user interface provides a means for the user to get work done more quickly and efficiently. This is especially true for the user interface found on modern operating systems. The best modern operating systems combine graphical simplicity with sophisticated text command input capability and output display of results to provide the user with powerful access to the facilities of the computer.
- Under most conditions, the computer system operates well below full capacity. The CPU sits idle while waiting for I/O transfers to occur. A user sits thinking at the keyboard. Multiuser and multitasking operating systems make it possible for many users or tasks to share the computer resource, providing fuller utilization of the system.
- The operating system extends the capability of the computer to include features that require special coordinated hardware and software that is invisible to the
The operating system provides powerful tools to the user’s programs that improve the quality of the programs and make the user’s work easier. For example, modern OS tools allow work to be easily transferred between applications through a clipboard, or make it possible to embed a spreadsheet into a word processing document. System services are provided by an API, or Application Programming Interface. The API provides file and I/O services, tools that create and support the graphical user interface, even tools to embed a spreadsheet into a word processing document.

We say that the operating system is event driven. This means that the operating system normally sits idle and executes only if some event occurs that requires operating system action. Events may result from interrupts or from service requests by a program or a user. Events include file requests, I/O, keyboard inputs from users, memory requests from programs, messages sent from one program to another, clock interrupts that allow the operating system scheduler program to dispatch programs during time sharing operation, network requests, and much, much more. In reality, the operating system on a large computer has quite a bit of work to do. Service requests and interrupts are a fundamental means of communication with the operating system.

Computer designers attempt to integrate the computer hardware and operating system, so that each supports the features of the other in such a way as to create a powerful environment for the users and for the users’ programs. Such an environment is called symbiotic. This would seem to suggest that each type of computer hardware would require its own proprietary operating system. In fact, this is not necessarily the case. Most modern hardware vendors do not provide their own brand of operating system at all. Instead, their systems are supplied with a standard operating system such as Linux or Windows XP or Vista.

Linux and Windows Vista are both examples of operating systems that operate on a variety of different hardware platforms. There is a strong advantage at providing a standard operating system that works on different hardware. Such a system provides program portability, as well as file portability, and also allows users to move comfortably from one machine to another by providing a recognizable interface and command structure.

Portable operating systems are designed in such a way that they may be tailored for different hardware by changing only the small portion of the operating system program code that interacts directly with the hardware. Most of the operating system is written in high-level language, which can be ported easily to a new machine by recompiling the high level code. The portion of the operating system that must be built for the individual machine is written in a mixture of high-level language and assembly language. Languages such as C++ and Java are ideal system languages, because they provide facilities that make it possible to interact with the hardware with very little need for assembly language. In fact, the language C was originally designed specifically for this purpose. The portability of Linux, and other modern operating systems, stems directly from this capability.

While it is true that a single operating system can be ported to operate with different hardware, it is also true that a particular hardware platform can support different operating
systems. Thus, the user or system designer can select an operating system that provides the desired facilities for the particular use of the system. Although x86-based personal computers have traditionally been provided with some version of Windows, there are other operating system options available that a user could select. An unsophisticated user on a stand-alone system might run Windows XP or Vista for its familiarity and ease of use, but a more sophisticated user with particular needs might prefer Apple MacIntosh OS X for its excellent tools and applications or Linux for its additional power. Particularly, if the machine is supporting multiple users, an X Windows-based Linux or UNIX operating system might be more appropriate.

15.3 SERVICES AND FACILITIES

Section 15.2 provided an overview of the various services and components that make up an operating system. In this section we consider the fundamental building blocks of an operating system in more detail. There are ten major blocks to be considered, not all of which will necessarily be found in any particular operating system:

- The command processor, application program interface, and user interface
- The file management system
- The input/output control system
- Process control management and interprocess communication
- Memory management
- Scheduling and dispatching
- Secondary storage management
- Network management, communication support, and communication interfaces
- System protection management and security
- Support for system administration

Some systems also provide a program known as a system manager, commonly known as a monitor or supervisor, which handles competing requests or conflicts, and which acts as a general controller and arbiter for the entire system. There are other system functions, such as accounting and error handling, that are sometimes handled as separate blocks but frequently appear within the blocks already listed.

In different types of operating systems, some of these components may be combined, or even absent. An embedded system may not require a file system or memory manager if all its programs are permanently resident in ROM, for example, but the listed components represent a collection of the most general operating system requirements.

Some of these modules, particularly the command interface and file system modules, are quite visible to the user. The other modules are primarily used for internal control of the system, controlling and optimizing use of the hardware resources, and maximizing program throughput and efficiency. Most modules also make their services available to user programs through the API.

In this section, we present an overview of the services provided by each of these operating system components. Individual components are discussed in more detail in other chapters, the capabilities and operation of the user interface and related services in
Chapter 16, and the file management services in Chapter 17. Details of the most important internal components and operations of the operating system are discussed in Chapter 18.

**User Interface and Command Execution Services**

To the user, the most important and visible service provided by the operating system is the user interface and the capability that it provides to execute commands.

Some systems do not consider the user interface and command processor to be a part of the operating system kernel, even though much of it is likely to be memory resident. Instead, these systems consider the user interface as a separate shell that is provided with the operating system and that interacts with the kernel to provide the necessary user command capabilities. Theoretically, a different shell could be used that provides different command capabilities. In Linux, for example, two different GUI shells, KDE and Gnome, and three different text-based shells, bash, csh, and zsh are in common use, and many other shells for Linux are available. Each of these shells provides different features and command structures and capabilities.

Different types of user interfaces exist. The most common are the graphical user interface (GUI), and the command line interface (CLI). The graphical user interface accepts commands primarily in the form of icons, drop-down menus or tabbed ribbons, mouse movements, and mouse clicks. Some GUI interfaces are also sensitive to touch or to hand motion in the vicinity of the screen. The command line interface relies on typed commands. Underneath the very different appearances of these interfaces, however, similar commands are being executed.

Regardless of the user interface provided, the command interface provides direct access to various other modules within the operating system. The most often used commands access the file system for file operations and the scheduler for program loading and execution. On some systems, commands may also provide direct access to the I/O system, protection services, network services, and process control services. On other systems, these commands may be processed indirectly, using built-in operating system utilities intended for the purpose.

A few systems even provide commands and built-in utilities for access to memory and to secondary storage. Generally, use of these commands is restricted to users with special access needs, such as the people who control and maintain the system. UNIX and Linux, for example, refer to these individuals as “superusers.”

Some commands are built directly into the operating system. They remain in memory for immediate access. These are known as resident commands. Other commands are loaded only as they are needed. These are called nonresident commands.

Most modern operating systems provide some capability for combining computer commands into pseudo-programs, commonly called shell scripts. Batch-oriented systems also make it possible to combine individual commands into a sequence of control statements, which will be interpreted and executed one at a time without user intervention to control the processing of a multistep “job.” Each step in the job performs an individual task. On large IBM systems, for example, the set of commands used for this purpose form a language known as Job Control Language (JCL).

In addition to the standard operating system commands, shell scripting languages typically provide branch and loop commands and other computer language features. Shell
scripts can be executed as though they were actual programs. Other common features include:

- A means for redirecting I/O data to a device different from that ordinarily used, to a disk file instead of the screen, for example
- A way to combine commands using a technique called piping, so that the output from one command is automatically used as the input for another
- A means for providing additional parameters to the script that can be entered by the user at the time the program is executed

More sophisticated command languages provide larger command sets with a more extensive and powerful set of options and with more extensive control structures that allow the creation of shell scripts with more flexibility, both in design and in run-time execution. Some command languages even provide special powerful commands that can eliminate normal programming effort. UNIX and Linux are particularly notable in this regard, providing commands that can search, select, edit, sort, enumerate, and process data from files in a way that rivals many programming languages.

The simplest Windows scripts are based on a command set that evolved from MS-DOS. These scripts are commonly called \texttt{.BAT files}. Recent versions of Windows also include a more powerful scripting facility called \texttt{Windows PowerShell}. PowerShell is based on an object-oriented language similar to C# and can manipulate both text and graphical objects.

There are a number of scripting languages that are designed to work independently of the particular operating system in use. The most popular of these include perl, python, PHP, Ruby, and JavaScript. Command and scripting languages extend the power and flexibility of the operating system and simplify use of the system for less sophisticated users.

\section*{File Management}

The concept of a file is central to the effective use of a computer system. A file is generally loosely defined as a collection of related information. Defined in this way, a file is a rather abstract concept; indeed, the contents of the file only have meaning in the context of their particular internal description and use. Thus, the sequence of bytes in a file might represent a program, or a graphical image, or maybe the alphanumeric text data for a book, to be used within a word processor. A file may be organized internally into records or it may simply be a stream of bytes. A file constitutes a \textit{logical unit} of storage, that is, logical to the person or program using the file. The logical unit may or may not correspond to the physical storage characteristics of the I/O device where it is stored.

The \texttt{file management system} provides and maintains the mapping between a file's logical storage needs and the physical location where it is stored. The file management system identifies and manipulates files by the names provided by their users. It determines the physical requirements of the file, allocates space for it, stores it in that space, and maintains the information about the file so that it may be retrieved, partially or in full, later. The file management system keeps track of the available space on each device connected to the system. The user and the user’s programs need not be aware of the underlying physical storage issues. Users and programs simply access the files by name, and the file management system handles the details.
The file management system provides a consistent view of files across different I/O devices. This view even extends to files located elsewhere, on devices accessible from a network. To the user, file requests operate in the same way independent of the device, even between devices of different characteristics. Thus, it is not necessary to know the physical differences between, say, disk and tape, to move a file from one to the other. A program can request file services without knowing the file structure of the device being addressed, indeed without even knowing what kind of device the file is stored on.

The file management system provides and maintains

- Directory structures for each I/O device in the system and tools to access and move around these structures. The directory structure allows the retrieval and storage of files by name, keeps track of the mappings, allocates and frees space, allows the mounting and unmounting of file structures, and provides other functions required to maintain the structures of the file system. Provisions are made to move easily from one structure to another.
- Tools that copy and move files from one I/O device to another and from one directory to another, merge files, create and delete files and directories, and undertake other basic file manipulations.
- Information about each file in the system and tools to access that information. Typically, information about a file might include its name, type of file, size, date and time of creation, date and time of the most recent update, and protection and backup characteristics.
- Security mechanisms to protect files and control and limit file access to authorized users. Most modern systems also provide encryption protection and journaling, a technique which assures the currency and integrity of files when system failures occur during file changes.

Some file management systems also provide advanced features, including auditing, backup, emergency retrieval and recovery mechanisms, file compression, and transparent network file access.

File management systems are particularly important in systems in which secondary storage devices are shared in common by multiple users, since they provide a directory system that assures that there is no duplicate use of physical storage. Without this facility, it is likely that users would unintentionally overwrite each other’s files. And, of course, we already noted that the file management system also provides file access protection between the different users. The file management system is discussed more fully in Chapter 17.

Input/Output Services

In Chapter 9, we introduced the concept of interrupts and showed the various techniques for handling I/O. Programs that implement these concepts are known as I/O device drivers. It would be awkward to require each program to provide its own I/O services. I/O device drivers are important because they are available to serve every program that will be executed on the system and provide a standard methodology for the use of each device. Even more important, the use of standard I/O drivers within the operating system limits access and centralizes control of the operations for each device.
The operating system includes I/O device driver programs for each device installed on the system. These drivers provide services to the file management system and are also available, through the API, to other programs for their use. The I/O device drivers accept I/O requests and perform the actual data transfers between the hardware and specified areas of memory.

In addition to the I/O device drivers provided by the operating system, modern systems provide certain I/O drivers with minimal functionality in ROM, to assure access to critical devices, such as the keyboard, display, and boot disk during the system startup process. The ROM-based drivers are replaced or integrated with other I/O drivers during normal system operation. On IBM-type PCs, these drivers are stored in the system BIOS (basic input/output system).

Device drivers for newly installed devices are added and integrated into the operating system at the time of installation. On some systems, the process is manual. On many systems, the Apple Macintosh, for example, this process is completely automatic. In Windows, this capability is known as plug-and-play. Many modern systems even make it possible to add and modify devices on the fly, without shutting down the system. USB and FireWire both provide this capability.

Every operating system, large or small, provides input/output services for each device in the system. The use of one set of I/O services for each device assures that multiple programs will not be competing for the device and assures that the use of each device will be managed through a single point of control. Multiple access can cause serious conflict in multitasking systems. For example, a user would not be pleased to discover that parts of the printouts from two different programs were intermingled on the pages, even more so if the outputs belonged to two different users! The operating system assigns and schedules I/O devices appropriately to each process to eliminate this problem.

**Process Control Management**

Briefly, a **process** is an executing program. It is considered the standard unit of work within a computer system. Every executing program is treated as a process. This includes not only application programs, but the programs within the operating system itself. The process concept considers the program, together with the resources that are assigned to it, including memory, I/O devices, time for execution, and the like. When admitted to the system, each program is assigned memory space and the various resources that it initially requires to complete its work. As the process executes, it may require additional resources, or it may release resources that it no longer needs. The operating system performs various functions with processes, including scheduling and memory management, by providing the various services that we have discussed in this chapter. Processes must often be synchronized, so that processes sharing a common resource do not step on each other’s toes by altering critical data or denying each other needed resources. Systems also provide communication capability between different processes. Processes may cooperate with each other by sending messages back and forth using **interprocess messaging services**. Other services include functions such as setting process priorities and calculating billing information.

Process control management keeps track of each process in memory. It determines the state of each process: whether it is running, ready to run, or waiting for some event, such as I/O to be completed, in order to proceed. It maintains tables that determine the current
program counter, register values, assigned files and I/O resources, and other parameters for each process in memory. It coordinates and manages message handling and process synchronization.

Many modern systems further break the process down into smaller units called threads. A thread is an individually executable part of a process. It shares memory and other resources with all other threads in the same process, but can be scheduled to run separately from other threads.

Memory Management

The purpose of the memory management system is to load programs and program data into memory in such a way as to give each program loaded the memory that it requires for execution. Each program that is being executed must reside in memory. For multitasking to occur, multiple programs will occupy memory simultaneously, with each program in its own memory space.

The memory management system has three primary tasks. It attempts to perform these tasks in a way that is fair and efficient to the programs that must be loaded and executed.

1. It keeps track of memory, maintaining records that identify each program loaded into memory together with the space being used and also keeps track of available space. It allocates additional space for running programs as required. It prevents programs from reading and writing memory outside their allocated space, so that they cannot accidentally or intentionally damage other programs.

2. If necessary, it maintains one or more queues of programs waiting to be loaded into memory as space becomes available, based on such program criteria as priority and memory requirements. When space is available, it allocates memory to the programs that are next to be loaded.

3. It deallocates a program’s memory space when it completes execution. The deallocated space is made available for other programs.

Older systems used a variety of algorithms to divide up the available memory space. Except for special-purpose embedded systems, every modern computer system provides virtual storage, a method of utilizing memory which includes hardware support for sophisticated memory management capability. Virtual storage creates the illusion of a memory space that is potentially much larger than the actual amount of physical storage installed in the computer system; its development was a major breakthrough in system capability. Where virtual storage is available, the memory management module of the operating system works directly with the hardware and provides the software support to create an integrated memory management environment that takes maximum advantage of the features of virtual storage. Virtual storage is explained in detail in Chapter 18, Section 18.7.

Scheduling and Dispatch

The operating system is responsible for the allocation of CPU time in a manner that is fair to the various programs competing for time, as well as maximizing efficient utilization of the system overall.
There are two levels of scheduling. One level of scheduling determines which jobs will be admitted to the system and in what order. Admission to the system means that a job will be placed into a queue, based on some order of priority, and ultimately assigned memory space and other resources, which will allow the program to be loaded into memory and executed. (Some operating systems divide this operation into two separate tasks, one for admittance to the system, the other to assign memory.) This scheduling function is sometimes known as **high-level scheduling**. The other level of scheduling is known as **dispatching**. Dispatching is responsible for the actual selection of processes that will be executed at any given instant by the CPU. The dispatch component of the operating system makes concurrency possible by allocating CPU time in such a way as to make it appear that several processes are executing simultaneously. For those systems that allow the division of processes into threads, dispatch is done at the thread level, instead of at the process level.

In modern systems, with their extensive facilities and capabilities, high-level scheduling is relatively straightforward. Most of the time, new processes will simply be admitted to the system and given memory space if it is available, or held until space is available, then admitted.

Selecting the appropriate candidate for CPU time at any given instant is much more important and difficult, since the capability of the dispatcher directly affects the ability of the users to get their work done. A single program cannot be allowed to "hog" the machine; therefore, the dispatcher must interrupt whatever process is running periodically and run itself to determine the status of the machine's resources and to reassign CPU resources to assure that every user and task is receiving what it needs.

Since a single CPU can process only one instruction at a time, the simultaneous execution of two or more programs is obviously impossible with a single processor. Instead, the dispatcher acts as a controller to provide concurrent processing. There are various ways in which multitasking can be achieved with concurrent processing, but mostly these methods take advantage of two simple strategies:

1. While one program is waiting for I/O to take place, another can be using the CPU to execute instructions. This strategy is shown in Figure 15.3. In Chapter 9 we demonstrated that I/O could be performed efficiently without tying up the instruction executing capability of the CPU. We further showed that most of the time, the CPU was idle, since I/O represents such a large percentage of a typical

---

**FIGURE 15.3**

Sharing the CPU During I/O Breaks

<table>
<thead>
<tr>
<th>Program 1</th>
<th>Executing</th>
<th>I/O</th>
<th>Waiting</th>
<th>Executing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 2</td>
<td>Executing</td>
<td>I/O</td>
<td>Waiting</td>
<td></td>
</tr>
<tr>
<td>Program 3</td>
<td>Executing</td>
<td>I/O</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
program’s execution. This suggests that the idle time can be used to execute other programs, as an effective way to increase utilization of the CPU.

2. The CPU may be switched rapidly between different programs, executing a few instructions from each, using a periodic clock-generated interrupt. This method was discussed in Chapter 9, Section 9.3, and diagrammed in Figure 9.8, redrawn here as Figure 15.4. This strategy will slow down execution of each program, since each program must split its time with other programs. There is also some operating system overhead, as a dispatcher must be invoked at each interrupt to select the next program to receive CPU time. In most cases the CPU is so powerful compared to the requirements of the programs that the slowdown is not even noticeable. This technique is called **time-slicing**.

The algorithms used by the dispatcher combine these two methodologies, taking into account such issues as fairness to each program, the priorities of the different programs, quick response for critical situations, such as displaying a user’s cursor movement or displaying streaming video, the number of CPUs available for dispatching, and other criteria.

Different processes have different requirements. Some processes require extensive amounts of CPU time; such processes are considered to be **CPU bound**. Others are mostly I/O operations, with very little CPU processing; these are known as **I/O bound**. Immediate response time is important under some conditions, for example, when echoing cursor movement to a screen and unimportant in others, such as producing printed output from a batch job that will not be picked up by the user until later in the day. It is obviously desirable to dispatch processes in such a way that the system is used effectively. Various dispatching algorithms are used to meet these different requirements, and there are various criteria for measuring how well the dispatcher is doing its job. Generally, interactive processes require faster response than do batch processes. Processes that must control instrumentation in real time require the fastest response of all.

The dispatcher is also responsible for the actual transfer of control to the process that is being dispatched. This responsibility includes preservation of the previous running program’s program counter, register values, and other parameters that represent the state of the program at the time it was stopped, as well as restoration, if necessary, of the exact previous state of the program being dispatched. This operation is called **context switching**.

The operation of the dispatcher is dependent on the nature of the system and on the nature of the programs that the system is running. The dispatcher can be **preemptive** or **nonpreemptive**. The dispatcher for a nonpreemptive
system replaces an executing program only if the program is blocked because of I/O or some other event or if the program voluntarily gives up the CPU. When necessary, the executing program may be suspended momentarily, so that the CPU can handle interrupts, but when the interrupting task is complete, control is returned to the same program. (This exception is necessary for several reasons. Without it, there would be no way to stop a runaway program, for example, a program with an infinite loop in it. It is also necessary to prevent losing keystrokes from user keyboards and to echo keystrokes back to users’ screens.)

Preemptive multitasking uses the clock interrupt, as described earlier, to preempt the executing program and to make a fresh decision as to which program executes next.

In general, nonpreemptive dispatching algorithms apply mostly to older, batch-oriented systems. Modern dispatchers are predominately preemptive. However, most provide a mechanism to dispatch individual programs nonpreemptively, for programs that must execute to completion without unnecessary interruptions. Linux uses nonpreemptive dispatching to protect certain operating system operations from interrupts that could destroy the integrity of operating system data, for example.

A more detailed explanation of process creation can be found in Chapter 18, Section 18.3. Scheduling and dispatching are discussed further in Chapter 18, Section 18.5.

Secondary Storage Management

The file management system keeps track of free secondary storage space and maintains the file system and its directories. The input/output control system provides device drivers that actually control the transfer of data between memory and the secondary storage devices.

On large multitasking systems there may be many programs requesting I/O services from a secondary storage device at one time. The order in which these requests are fulfilled affects the ability of the different programs to get their work completed, since the programs must usually stop and wait for their I/O requests to be completed before proceeding. Although it would be simplest to process I/O requests in the order received, it may be more efficient to process the requests out of order, particularly if the blocks requested are scattered all over the disk. This is true because the disk seek time (i.e., the time to move from track to track) is long compared to other times within the system.

The secondary storage management system attempts to optimize the completion of I/O tasks by using algorithms that reorder the requests for more efficient disk usage. For example, it might attempt to read all the requested data blocks from the tracks in one area of the disk before going to read data on tracks at the other end of the disk. In some large modern systems, optimization is provided by a combination of I/O hardware and operating system software. Further details of secondary storage management appear in Chapter 18, Section 18.8.

Network and Communications Support Services

With the exception of some specialized embedded systems, nearly all computers today are interconnected, directly or indirectly, into networks. (There is even a trend toward networking embedded computers: modern automobile computers routinely report maintenance problems to the service technician when you bring your car in for service—some
cars even report problems wirelessly from the road to a service representative. And you may have heard of the refrigerator that calls an order in to an Internet grocery delivery service when food stocks are low.) The network and communications support facilities within the operating system carry out the functions required to make the system perform seamlessly in a networked and distributed environment.

Most modern communications services rely on the TCP/IP protocol suite, together with its IP-based applications. TCP/IP provides the facilities to locate and connect to other computer systems, to pass application data in packet form from one system to another, to access files, I/O devices, and programs from remote systems, to provide error checking and correction when appropriate, and to support the requirements of distributed processing. Network and communication services within the operating system provide the communication software necessary to implement the features and facilities of TCP/IP. Most systems also implement a substantial set of TCP/IP applications and extensions, including e-mail, remote login, Web services, streaming multimedia, voice over IP telephony (VoIP), secure networking across the Internet (called a virtual private network, or VPN), and more. Some systems also offer support for alternative communication protocols, for example, Novell IPX/SPX and IBM Systems Network Architecture.

Communications services within the operating system also provide the interface between the communication software and the OS I/O control system that provides access to the network. The I/O control system includes the software drivers for modems, network interface cards, wireless communication cards, and other devices that are used to connect the computer physically and electrically to the network or networks.

Larger computers used for server applications often require the capability for additional growth and reliability to serve the needs of their clients. These capabilities are sometimes referred to as system scalability and fail-safe operation respectively. In addition to networking support, the operating systems for such machines often include clustering software, so that these computers can be clustered together and viewed transparently by clients and users as a single, high-powered system. The clustering software provides single-point logins, single-point user and client requests, request steering, failure detection and cutover, and system load balancing between the individual nodes within the cluster.

Security and Protection Services

It is certainly no surprise to anyone that modern systems require security and protection services to protect the operating system from user processes, to protect processes from each other, and to protect all processes from the outside world. Without protection, a buggy or malicious program, for example, could unintentionally or intentionally modify or destroy the program code or data in the memory space belonging to the operating system or to another process. It is also important to protect the system and user processes from unauthorized entry to the system, and against unauthorized use of the system, even by authorized users.

In most modern systems, executing processes are limited to the execution of instructions and access to data within their own memory space. All other services, such as file management and I/O, must be requested by the process from the operating system, using the service requests provided by the OS for that purpose. This methodology is fundamental to the security of the system. In this way, the operating system, the file system, and other
processes are protected from unauthorized use or operations, protecting the integrity of
the system as a whole. Interprocess messaging services are usually provided by the oper-
ating system to allow processes to communicate with each other without compromising
the system. Critical parts of the operating system execute in a specially protected mode of
operation provided as part of the CPU design. In protected mode, the operating system can
prevent programs from executing certain instructions and from accessing parts of memory
specified by the operating system, for example, parts being used by other programs.

Each module in the operating system includes provisions that protect its assets. Thus,
the file management system would not allow a process to store data on a part of a disk
that is being used by another file. Nor would process management allow the assignment
of an I/O resource that would prevent another process from completing its task. Since all
services are requested from the operating system, the OS has the capability to determine
that requests will not damage other processes or the system itself.

The operating system also provides login and password services that can help to prevent
entry from unauthorized users and access control facilities that allow users to protect their
individual files at various levels of availability to other users and outsiders. The modern
operating system includes firewall protection, which, artfully administered, can make it
more difficult for outsiders to penetrate the system, but is not foolproof; the need for
security must be carefully balanced with the needs of the users to get their work done.
Despite all the protection offered by a modern system, bugs, viruses, and vulnerabilities
within the operating system, poor configuration of firewalls and other security features,
and poor user management policies such as weak password enforcement can make a system
vulnerable to attack by outsiders. The design and deployment of effective security and
prevention services is an important ongoing concern in operating system design. A number
of research projects that show promise are attempting to design OS security mechanisms
that prevent infiltrators from moving beyond the actual program that they invaded into
other areas of the operating system.

System Administration Support

The system administrator, or sysadmin, for short, is the person who is responsible for
maintaining the computer system or systems. In a large organization, the sysadmin may
support hundreds, or even thousands, of computers, including those of the employees.
Some of the important administrative tasks managed by a system administrator include:

- System configuration and setting group configuration policies
- Adding and deleting users
- Controlling and modifying user privileges to meet the changing needs of the users
- Providing and monitoring appropriate security
- Managing, mounting, and unmounting file systems
- Managing, maintaining, and upgrading networks
- Providing secure and reliable backups
- Providing and controlling software, installing new software, and upgrading
  software as required
- Patching and upgrading the operating systems and other system software
CHAPTER 15 OPERATING SYSTEMS: AN OVERVIEW 499

- Recovering lost data
- Tuning the system for optimum availability and performance
- Monitoring system performance and recommending system modifications and upgrades when necessary to meet user requirements

These and other important tasks must be applied both to central server systems and to client machines and other desktop computers on a network to coordinate and maintain a reliable and useful system. Modern operating systems provide software to simplify these tasks.

On small personal computers, the user is often the administrator as well. The major administrative tasks of the user are to install and upgrade software, to reconfigure the system and the desktop from time to time, to maintain network connections as required, and to perform regular file backup and disk maintenance and defragmentation. For user administration of this type, simple tools are sufficient. Indeed, the goal of a desktop operating system might be to hide the more sophisticated tools from the typical user. For example, Windows operating systems store the system configuration within a registry that is normally hidden from the user, and provide, instead, a variety of simple tools specifically for tailoring the system to user preferences and performing maintenance tasks. The Windows operating system supplies default configuration parameters for many tasks that suit the needs of most users, with tools to modify the parameters to meet specific user requirements. The simplest tools are sufficient for most users to perform routine system administration. Knowledgeable users can also manipulate the system registry directly, when necessary. On desktop computers connected to a larger system within an organization, central administration tools allow the application of group policies and configuration to individual desktop computers without user involvement.

On larger systems, administration is much more important and much more complex. The hardware and software to be managed is far more extensive, and there are numerous users requiring accounts and service. Installation of new equipment on large systems is common, and in some cases, the system must be reconfigured to use the new equipment. IBM calls this process sysgen, for system generation. It is one of the most important tasks of system administration on large systems. Modern systems provide software for simplifying common system administration tasks. Large mainframe operating systems provide tools for performing all the major system administration requirements. They also provide tools that allow the administrator to tailor the system to optimize its performance, for example, to optimize throughput or the use of resources. This is done by modifying system parameters and selecting particular scheduling and memory management algorithms. Among the parameters that can be adjusted on various systems are the amount of memory allocated to a program, user disk space allocation, priorities, assignments of files to different disks, the maximum number of programs to be executed concurrently, and the scheduling method employed. IBM z/OS even includes a Workload Manager, which attempts to optimize system resources automatically, without administrator intervention.

The system administrator on a traditional UNIX/Linux system, for example, can log in to the system as a superuser, with privileges that override all the restrictions and security built into the system. The superuser can modify any file in the system. (However, the new security mechanisms mentioned above might make it much more difficult to override the security, thereby protecting a system from a hacker who manages to infiltrate the
More important, the UNIX system provides tools that simplify the tasks of system administration. These tools take the form of commands that can be executed only by the superuser and text-based configuration files that can be modified with any text editor.

For example, UNIX/Linux systems typically provide a menu-driven or graphical `adduser` program for administering user accounts. This program provides a simple procedure for performing all the tasks required to add a new user to the system, including setting up the user name and ID number, building entries to the appropriate user and group tables, creating the user’s home directory, assigning login shells, and establishing user initialization files (corresponding to the user’s particular terminal hardware, prompt preferences, and the like).

Other typical UNIX/Linux administration commands include a partition tool for partitioning hard disk drives; `newfs` for building a file system; `mount` and `umount` for mounting and unmounting file systems; `fsck` for checking and repairing the file system (similar in concept to, but much more complex and thorough than, CHKDSK on Windows systems); `du` and `df` for measuring disk usage and free space; `tar` for collecting files into archives; and `ufsdump` and `ufsrestore` for creating backups and recovering damaged files. `config` is used to build the system. There are many additional tools available to the UNIX/Linux sysadmin.

Like other large systems, server-based versions of Windows provide a full suite of tools for measuring system performance and managing the system, including the ability to control and configure client systems remotely.

Most systems provide a variety of statistical information that indicates the load on the system and the efficiency of the system. This information is used by the system administrator as a basis for tuning the system. Part of a typical system status report appears in Figure 15.5. This particular report comes from a Linux system. The report indicates the load on the system as a function of time, shows CPU and memory usage, identifies the most CPU-intensive processes, together with the name of the user and the percent of CPU and memory resources consumed, shows the efficiency of virtual storage, and provides many other useful system parameters. It even provides an analysis of the data shown. Although the typical user might not find a report such as this very useful in terms of what steps to take as a result of the information presented, a skilled system administrator can make valuable use of the information in determining ways in which to improve system performance. A consistently heavy load on a particular disk might suggest splitting the most used files on that disk onto two separate disks, so that they might be accessed in parallel, for example. Or heavy use of the CPU by a particular user during peak hours might suggest lowering the priorities for that user at those times.

**SYSTEM GENERATION**

One of the most important system administration tasks to be performed is the creation of an operating system tailored to the specific needs of a particular installation. The process of building a system is called a **system generation**, or more familiarly, a `sysgen`. The result of a sysgen matches the operating system to the characteristics and features of the hardware provided and includes the desired operating system features and performance choices. Two primary means are used to tailor the system:

- By selecting the operating system program modules to be installed. Typically, an operating system provides a large number of modules that might be used under different circumstances. Only those modules that are relevant to the installation
are selected. As an example, a particular installation has an individualized selection of I/O devices. Only those device drivers that are required for the installed I/O devices would be included in the tailored system.

- By assigning values to parameters of the system. Parameters are used to provide the details of an installation. On a Windows-based PC system, for example, devices are assigned to specific, numbered interrupt channels known as IRQs; memory locations for each device interrupt driver are also specified. Another example of a parameter would be the number of concurrent users permitted on a multiuser system. On some systems, a parameter might be used to determine whether a module is memory resident or is loaded on demand. Most large systems also provide parameters that tailor the system scheduling mechanism and adjust the behavior of other resource control modules. These and other parameters must be determined by the system administrator to meet the needs of the installation.
Some systems provide a lot of flexibility, with many options. Other systems may provide only a minimal amount of selection, perhaps no more than a selection of I/O device drivers.

The method used to perform a sysgen depends on the operating system. Some systems provide the operating system modules in source code form. Modules and parameters are selected, and the operating system is assembled or compiled and linked to form the loadable binary operating system. A barebones operating system with the appropriate compilation tools may be provided to enable the sysgen procedure to take place on the target system, or the procedure may be executed on a different machine. Other operating systems use an installation program to determine which modules should be included in the operating system, and parameters are selected during the installation procedure. On these systems, the various modules are already provided in binary form and need only be linked during the sysgen procedure.

On many systems, the sysgen procedure is provided as a series of menu selections and parameter entry forms that guide the operator through the procedure. On some systems, the procedure is entered as a script or batch file. Most systems also allow some degree of dynamic configuration, which makes it possible to build changes into the system without rebuilding the entire system. We noted earlier that in Linux configuration script files are used for this purpose.

### 15.4 ORGANIZATION

There is no standard model for the organization of an operating system. Some systems were developed in a deliberate and carefully planned manner, while others grew topsy-turvy over a long period of time, adding new functions and services as they were required. Thus, the programs that make up an operating system may be relatively independent of each other, with little central organization, or they may form a formal structure.

Overall, the organization of most operating systems can be described generally by one of three configuration models. These are commonly referred to in the literature as the **monolithic configuration**, the **layered or hierarchical configuration**, and the **microkernel configuration**. Within a configuration, individual programs can be categorized in different ways. As we noted earlier, operating system programs can be memory resident or nonresident, depending on their function. Of the resident programs, some will operate in a protected mode, often called **kernel mode**, others in a conventional user mode.

As an example of a monolithic configuration, UNIX is commonly described by the model shown in Figure 15.6. In this model, the various memory resident operating system functions are represented by a monolithic kernel. There is no specific organization. The operating system programs simply interact as required to perform their functions. The critical functions within the kernel operate in protected mode, the remainder, in user mode. The shell is separate from the kernel and serves as an interface between the users, utilities, and user programs with the kernel. Thus, the shell can be replaced without affecting kernel operations. (UNIX organization is considered in more detail as a case study in Supplementary Chapter 2.)

The major difficulty with a monolithic configuration is the stability and integrity of the system as a whole. Any defect in a program within
the kernel can crash the entire system, as can unexpected interactions between different programs in the kernel. Thus, the addition of a new device driver, for example, could compromise the entire system. Nonetheless, with proper design and control, it is possible to build a secure and stable system, as evidenced by Linux.

An alternative operating system organization is built around a *hierarchical* structure. A simple representation of a hierarchical operating system organization is shown in Figure 15.7. This representation shows the operating system divided into layers. The upper layers are the ones that are visible to the user; the middle layers comprise the major kernel operations. The lowest layers are the I/O device drivers that interact with the hardware.

In this model, each layer is relatively independent of the other layers. Thus, the file management layer determines the location of a file identified by logical name and interprets the nature of the request, but does not attempt to access the hardware directly. Instead, it makes a request to the kernel. Local requests are then passed on to the I/O device driver level for access to the hardware. Network requests are passed on to the I/O device drivers on the machines providing the services.

The hierarchy is arranged so that access to the various layers of the operating system is from the top. Each layer calls the next lower layer to request the services that it needs. Most computer systems today provide appropriate hardware instructions that allow the operating system design to enforce this procedure. This provides security, as well as a clean interface between the different functions within the operating system.

![FIGURE 15.7](image-url)
Layered operating systems must be designed carefully, because the hierarchy requires that services be layered in such a way that all requests move downward. A program at a particular layer must never require services from a higher layer because this could compromise system integrity. Another disadvantage of the layered approach is the time required to pass the request through intermediate layers to receive services from the lowest layers. In contrast, a program in a monolithic operating system could request the service directly from the program that supplies the service, resulting in much faster operation. The obvious advantage of the layered approach is the stability and integrity that result from a well-structured modular design.

Still another approach to operating system design is the microkernel. An illustration of a microkernel configuration is shown in Figure 15.8. The microkernel configuration model is based on a small protected kernel that provides the minimum essential functionality. The definition of “minimum essential functionality” differs from system to system. The Mach operating system kernel includes message passing, interrupt processing, virtual memory management, scheduling, and a basic set of I/O device drivers. It is possible to build a microkernel with nothing but message passing, interrupt processing, and minimal memory management, although the practical advantage of doing so has not been shown.

The microkernel configuration constitutes a client-server system, where clients and servers reside on the same system. Operating system services outside the essential functionality are performed by programs in user mode. Each program acts as a server that performs specific operating system tasks upon request of application programs as well as other operating system programs, the clients in this model. Clients request services by sending messages directly to the microkernel. The microkernel passes the messages to the appropriate server, which performs the required function, and replies to the request by sending a message back to the client. The reply is also passed through the microkernel. System security and integrity is maintained, because all communication must pass through the microkernel.

One of the advantages of the microkernel configuration is that it is possible to create different operating system designs simply by changing the service programs that reside outside the microkernel, while maintaining the security and stability of the microkernel. For example, Macintosh OS X is one of many operating systems built on the Mach microkernel.
CHAPTER 15  OPERATING SYSTEMS: AN OVERVIEW  505

The microkernel approach offers reliability, flexibility, extensibility, and portability. It is particularly amenable to object-oriented design. New features can be added easily without compromising the system. The extensive message-passing required in a microkernel configuration can result in a performance penalty over other types of designs, but practical applications of the model have shown that, with care, the potential disadvantage of this approach can be minimized.

15.5 TYPES OF COMPUTER SYSTEMS

Modern computer system hardware is essentially similar regardless of the type of system. Therefore, the differences among computer systems are set primarily by the operating system software. The operating system software is selected to meet desired requirements and goals.

As we outlined briefly in Section 15.2, there are many different types of operating systems, each designed to meet a particular set of requirements and needs. Some of the factors that influence operating system design/architecture are the primary type of user base, whether the system is intended for direct user access or behind-the-scenes server access, or whether the system is to be used for a specific purpose, such as embedded electronic control or mobile use.

For example, one computer might be designed for business end users, another for programmers and engineers, and other high-technology specialists. The Macintosh, for example, is well designed for the inexperienced user (and for other users too, of course!) Windows is adequate for a user with simple needs; a more sophisticated user might choose a system with Linux instead. The PC is adequate for many single users, but a large mainframe type system might be more appropriate for use as a large server, or, perhaps, a network- or cluster-based system is more appropriate for a particular server application. Special-purpose applications that require specialized designs might include embedded control applications (such as automotive and microwave oven applications), CAD/CAM graphics, multimedia (the Pixar computer is a special system designed specifically for motion picture animation and special effects), and real-time control applications. An operating system designed for mobile computing may have to operate with limited resources, particularly in the areas of power consumption, network connectivity, memory size, and display. Each of these systems has different needs and requirements that are met by the operating system design.

There are, of course, costs associated with increasing sophistication in operating system software. As more features are added, more memory is required for the operating system. The original version of MS-DOS ran successfully in 64 KB of memory. The IBM MVS operating system for the IBM S/370 family required more than 6 MB of memory even before any applications were considered. Some computer experts have recommended a minimum of 2 GB of memory for Windows Vista and its applications. The overhead time required for the operating system to perform its functions becomes a sizable fraction of the overall time. One hopes that the overhead is worthwhile in terms of increased efficiency and ease of use. For example, graphical user interfaces and multimedia support consume a high percentage of system resources on personal computers. On a computer used primarily as a Web server, those resources might be better utilized in providing faster Web access or supporting more users.
Within the context of the previous discussion, we can loosely categorize computer systems into seven types: single-user systems and workstations; mainframe systems; operating systems for mobile devices, network server systems; real-time systems; embedded control systems; and distributed systems. (We noted earlier that systems capable of only a single task at a time, while historically important, are essentially obsolete.)

- The predominant systems in current use are single-user, multitasking systems. These are the systems found on laptop and desktop computers, workstations, tablets, personal digital assistants, “smart phones”, and other similar devices. Common examples include various versions of Windows, Macintosh OS X, Linux, and Sun Solaris. A GUI is usually a key feature of these systems since it allows the user easily to run several processes at the same time, maximizing overall productivity. Windowing interfaces allow output presentations from several tasks to appear on the screen simultaneously, and provide methods for easy task switching. (Note, however, that a windows environment is not a requirement for multitasking. Some single-user systems allow an individual user to multitask from a command line interface. Linux, Sun Solaris, and other UNIX-based systems, in particular, allow users to specify that processes are to execute in the “background.” Background processes can present output to the screen, but only the foreground process can accept input from the keyboard. The operating system provides commands that allow the user to select which process is in the foreground at any particular time.) Workstations generally provide single-user multitasking operating systems, although most workstations have the capability to be configured for multiuser or server operation.

- Mainframe operating systems are designed to manage large-scale computing resources, particularly in major enterprise environments, where large numbers of transactions requiring extensive I/O capability are the norm. Mainframe systems were originally created to allow hundreds of users to share the computing power of a central facility, as well as to support batch data processing operations such as billing and credit card processing. Today, mainframe operating systems manage typical mainframe computer hardware made up of clusters of multiprocessor units, all designed to work together as a single processing unit, with hundreds of gigabytes of memory, femtobytes of disk storage and I/O and networking capabilities of hundreds of gigabytes per second. Mainframe operating systems differ from smaller multitasking systems in the variety of features that they offer, in the versatility with which they can be configured, with the level of security that they supply, with the degree of control that they offer the system administrator(s), and in the overall amount of power and capability that they offer.

- Network server systems are similar to single-user multitasking systems in most respects. However, the major focus of system use is shifted from meeting the needs of the direct user to the support of clients connected to the server through a network. The server may have no direct user facilities of its own, other than those required for management of the system. The server is designed to provide Web services, file services, print services, application services, and/or database services to the clients, as determined by the particular requirements of the system.
application. It may also provide some program execution services for clients, including support for client system start-up, particularly on networks with thin clients. Network servers often work together in clusters. For these applications, one would expect the network server OS to provide improved security and system integrity protection, high reliability file management and backup with large file capability, strong support for clustering and multiprocessing, improved mechanisms for failure prevention, automatic cutover to alternative systems when failures occur, and failure recovery, as well as strong system administration capability.

- **Mobile operating systems** are operating systems designed for small hand-held devices, such as personal digital assistants (PDAs) and smart phones. These systems must provide the basic capabilities and features of traditional single-user multitasking systems within the constraints of electrical power limitations, limited memory, lower CPU execution speed, slower network capability, and file storage that is generally limited to small stationary nonvolatile memory devices, along with some special features that would not be required in a larger system, such as touch screen capability, special keyboard handling, careful management of battery power consumption, support for special I/O devices such as global positioning and telephony mechanisms, features for synchronizing data with other systems, and handwriting recognition.

- **Real-time systems** are systems in which one or more processes must be able to access the CPU immediately when required. Real-time systems are used for applications in which one or more programs are measuring or controlling I/O devices that must respond within specific time restraints. A real-time system might be used to control instrumentation, such as the control rockets on a space flight, or to measure time-sensitive data, such as the periodic measurements of the temperature in a nuclear reactor. Although some real-time systems are created special for the particular application, most are general-purpose multitasking systems that have been designed so that they can be used for other tasks except when the time-sensitive application is being executed. A real-time system could be viewed as a multitasking system in which the interrupts that cause execution of the real-time program or programs have very high priority, but in many cases, special effort is made to assure that the real-time program can operate within its required time restraints.

- **Embedded control systems** are specialized systems designed to control a single piece of equipment, such as an automobile or microwave oven. The software for embedded control systems is usually provided in ROM. Nonetheless, many functions of the operating system may still be found in these systems. The computer that controls an automobile, for example, requires most of the features of a multitasking system. There are many measurement sensors representing CPU input on a car and many different control functions to manage. The service technician must be able to connect an I/O terminal to the system for car analysis. Effectively, an embedded control system is a real-time system that is dedicated to the particular application.

- Finally, **distributed systems** are rapidly growing in prominence and importance. In a distributed system, processing power is distributed among the computers in
a cluster or network. Even the Internet can be used as a distributed system. Programs, files, and databases may also be dispersed. Programs may be divided into functional pieces, with execution distributed throughout the network. Alternatively, program components may be stored on different systems, and executed in place upon remote request. .NET and CORBA, discussed briefly in Chapter 16, are two standards designed to expedite this process. Regardless of which method is used, the operating system or systems require additional complexity to handle the distribution of tasks or instructions within a process, the sharing of memory and I/O, and the intercommunication of data and control that are required of these systems. Many modern computing systems include additional operating system modules to make distributed processing feasible and practical. Distributed Computing Environment (DCE), is an OpenGroup standard that establishes a set of features for a distributed computing operating system. (OpenGroup is an organization that promotes open computing by setting standards and certifying products in a number of major areas of computing. UNIX is arguably the best known OpenGroup standard.) The DCE standard is supported and incorporated into the operating systems of a number of major vendors, including Microsoft, Sun, and IBM.

There are, of course, other ways of categorizing computer systems. One way of describing systems that is sometimes useful is to consider the intent and philosophy of the designers of the system. This description can sometimes provide a lot of insight into the strengths and weaknesses of a system. For example, the IBM mainframe operating system, z/OS, is an offshoot of an operating system that was originally designed primarily for large, batch-oriented business transaction processing systems. As business users moved their operations online, predecessors of z/OS were provided with capabilities to handle large numbers of online transactions. This would suggest that the modern z/OS is well equipped to handle routine Web transactions from hundreds or thousands of network clients concurrently. At the same time, it might suggest that z/OS is not particularly user-friendly to individuals doing their own independent work on the system. Development tools are more difficult to use on z/OS than on many other systems. Most people would agree that these statements describe z/OS fairly well.

As a different example, the Apple Macintosh system was designed to make tasks as easy as possible for the average, minimally trained computer end user. As a result, much of the design effort for the Macintosh system has continually gone into the user interface. The operating system provides powerful interface and graphical resources to the user and to the user’s programs. Other operating system facilities, such as time sharing and memory management, became secondary to the stated purpose. Indeed, these functions in OS X are implemented with a kernel built from a UNIX variant called FreeBSD.

Finally, consider an operating system whose primary design goal is to be capable of open system operation. The primary features that define an open system are as follows:

- The system should be capable of operating on many different hardware platforms.
- Communication between systems should be simple and straightforward. Commands that access remote systems should perform nearly identically to those performing local operations and should appear as transparent as possible to the user or the user’s programs. Thus, a COPY command that copies files between
systems should operate essentially the same as one copying files between different points on a single system.

- Shell programs should behave identically, regardless of platform. Source level application programs should operate identically, once compiled on the new platform.

These features dictate an operating system with considerable thought given to networking, as well as to a system with minimum dependency on the particular hardware being used. This suggests an operating system with a small kernel, with powerful networking facilities built in, and with the hardware-specific part of the system concentrated into a single part of the kernel, isolating all other parts of the system from the platform. FreeBSD is an example of such a system, which makes it an ideal basis for the MacInosh OS X design.

There have been many attempts to build operating systems whose activities are truly distributed across a network. Some of the best known of these are Mach, Amoeba, Locus, and Chorus.

**SUMMARY AND REVIEW**

Chapter 15 presents a comprehensive overview of the operating system. The operating system software is a collection of programs that extend the power of the computer hardware by providing a user interface to the computer, plus control and support for the computer’s resources, plus other facilities that make it easier to manage and control the computer system. Many operating systems also make possible the sharing of computer resources concurrently among multiple users and among multiple tasks for each user.

The operating system provides one or more user interfaces, file support, control for I/O devices, network support, and management of the computer resources, including memory, the various I/O devices, and the scheduling of time. The operating system is event driven. It performs these tasks in response to user commands, program service requests, and interrupts. We noted that although the operating system represents overhead, under most conditions the overall computer system performance is improved and enhanced by the presence of the operating system. Some operations, particularly concurrency, would be difficult or impossible without the operating system.

In our discussion of the various operations performed by an operating system, we identified ten of the major services and facilities provided within an operating system and described each. These included the user interface and command execution, the file system, the I/O control system, process control, memory management, scheduling and dispatch, secondary storage management, network management, security, and system administration facilities.

The programs that provide these services must be organized in some way. There is a considerable amount of interaction between the different program modules that make up an operating system. Many operating systems use a hierarchical model to organize the various modules. This model has the advantage of a significant amount of protection, since it is easy to control access and the flow of information between modules using a hierarchy. Other models in use include the monolithic model and the microkernel model.

The chapter concluded by presenting various types of computer systems in use and compared the operating system facilities required for each. We noted that these categories are somewhat arbitrary, with substantial overlap between them.
FOR FURTHER READING

There are a number of excellent textbooks that describe operating systems in detail; recommended are books by Silberschatz and others [SILB08], Deitel [DEIT03], Tanenbaum [TANE07], Davis and Rajkumar [DAVI04], McHoes and Flynn [McHO08], and Stallings [STAL08]. Davis, in particular, presents a very practical, hands-on view of operating systems, with many examples. McHoes and Flynn is also quite practical and readable. The others tend to be deeper and more theoretical. For particular topics in operating systems, see the references at the back of this textbook and references in any of the other books. There are also numerous trade books that discuss particular topics in operating systems and specific operating systems. Henle and Kuvshinoff [HENL92] provides a satisfying low-level introduction to desktop computer operating systems.

KEY CONCEPTS AND TERMS

application programming interface (API)  
graphical user interface (GUI)  
operating system (OS)  
plug-and-play
basic input/output system (BIOS)  
hierarchical configuration  
preemptive dispatch
.BAT file  
high-level scheduling  
process
batch processing  
Initial Program Load (IPL)  
real-time system
bootstrapping  
interprocess message  
resident commands
command line interface (CLI)  
servicing  
service request
concurrent processing  
I/O bound  
shell
context switching  
I/O services  
shell scripts
control statements  
Job  
supervisor
convosational systems  
Job Control Language (JCL)  
system administrator
CPU bound  
Journaling  
system generation
diskless workstation  
kernel  
sysadmin
dispatching  
kernel mode  
sysgen
Distributed Computing Environment (DCE)  
layered configuration  
system languages
distributed system  
microkernel configuration  
system scalability
embedded control system  
monitor  
thin client
event driven  
multiprogramming  
threads
fail-safe operation  
multitasking  
time-slicing
file management system  
multiprocessor system  
virtual private network (VPN)

READING REVIEW QUESTIONS

15.1  The definition of an operating system specifies two primary purposes served by the operating system. What are they?

15.2  Explain the major error in the following sentence: “One of the major tasks performed by the operating system program is to load and execute programs.”
15.3 Explain concurrent processing. Briefly describe at least two services that an operating system must provide to support concurrent processing.

15.4 What are the memory resident parts of an operating system called? When are these parts loaded into memory?

15.5 What is a diskless workstation or thin client?

15.6 What does API stand for? What is the purpose of an API?

15.7 Operating systems are said to be event driven. Explain what this means.

15.8 What is the difference between multiprogramming and multiprocessing?

15.9 Explain dispatching. Describe the two basic methods that are used by operating systems to implement dispatching.

15.10 What tasks are performed by device drivers?

15.11 The basic role of a file management system is to provide a mapping service. Between what and what?

15.12 Briefly describe at least three of the four major services provided by a file management system.

15.13 Explain the concept of a process. How does a process differ from a program?

15.14 Describe at least two primary tasks performed by the memory management component of an operating system.

15.15 Explain what is meant by preemptive and nonpreemptive scheduling.

15.16 Identify at least four different tasks performed by a sysadmin.

15.17 What is true of system administration on small personal computers that is usually not true of larger systems or of personal computers within an organization?

15.18 What is the purpose of system generation?

15.19 The UNIX kernel is described as a monolithic organization. What does this mean? What are the major challenges presented by a monolithic organization?

15.20 Describe the organization of a hierarchically structured operating system.

15.21 How do real-time systems differ from other types of operating systems?

**EXERCISES**

15.1 What are the specific limitations of a computer system that provides no operating system? What must be done to load and execute programs?

15.2 For each of the most popular commands in Windows (or Linux if you prefer), identify the type of operating system service that is being provided, and identify the basic module or modules that are involved. Which commands would you assume are memory resident and which loaded as required? Explain your assumptions.

15.3 Concurrency, of course, is a requirement for modern operating systems. What are the major challenges that an OS designer faces in supporting efficient concurrency that she would not face if the operating system could just run one program at a time?

15.4 What are the limitations of providing a BIOS in ROM?
15.5 You are probably familiar with the standard Windows interface. Suppose you could replace the Windows shell with a different interface shell. What might be the advantages and disadvantages of selecting a different command shell as a replacement for the standard Windows interface?

15.6 Describe the two methods that are used to provide concurrent operation of multiple processes on a single CPU. What are the advantages of each method? What is the advantage of providing concurrent operation?

15.7 An operating system is described as an event-driven program. What is meant by event driven? Explain how the dispatching operation fits this description.

15.8 What is the difference between the logical description of a file and the physical description?

15.9 Nearly every operating system separates the file system from the I/O services. What is the advantage in doing so?

15.10 Discuss the similarities and differences between memory management fragmentation and disk fragmentation.

15.11 Early versions of Windows did not support true preemptive multitasking. Instead, the designers of Windows provided something they called “cooperative multitasking” in which each program was expected to give up control of the CPU at reasonable time intervals, so that the Windows dispatcher could provide execution time to another waiting program. Describe the disadvantages of this method.

15.12 If you have access to the system administrator of a large system, find out the steps that are required to perform a sysgen on the system. Also, determine the options that are available for that system.

15.13 One approach to operating system design is to provide as small a kernel as possible and to make all other modules optional. What are the minimum services that must be provided in such a miniature kernel?

15.14 Write a Little Man bootstrap loader that will reside permanently in high memory for the Little Man Computer. The reset button will automatically cause the Little Man to start executing the first instruction of your bootstrap loader. Assume that the application program to be loaded will be input one instruction at a time through the input basket and will be loaded into consecutive locations of memory. The last instruction of the application program will be a 999. When your loader sees this slip of paper, it will cause the Little Man to start executing the program.

15.15 Windows hides most of its configuration in a binary file called the registry. Special Windows tools must be used to read and modify the registry. What are the advantages and disadvantages of this approach versus the use of text-based configuration files?

15.16 Based on the system status report shown in Figure 15.5, describe some of the ways in which the system could be tailored, and explain how the various items in the report would influence your tailoring decisions.

15.17 What are the conditions and restrictions that you would want to impose on a multitasking system that is being used with real-time processes?
15.18 What operating system functions would you expect to find in the computer that is built in to control your automobile, and which functions would be omitted? Justify your answer.

15.19 Clearly explain the differences between multiprogramming, multiuser, and multiprocessing.

15.20 a. Of what use is the list of active processes shown in Figure 15.5? What changes might a system administrator make in the system on the basis of this information?

b. What does the average number of processes data tell you about the way that this system is normally used?

c. Compare the three graphs in the figure.