Chapter 1
Overview

Revised: Jan. 10, 2011

1.1 INTRODUCTION

The basic elements necessary to store and process information are contained in any computer. However, these elements are usually provided in a very primitive form that is quite difficult to use directly. Because of this problem, the **hardware** (physical components) of a computer is almost always supplemented by some type of standard **software** (programs and data). Together these elements work to form a more usable computing system.

The software portion of a computing system consists mainly of a collection of special programs called **system programs**, which differ from **application programs** that perform the specialized processing required by each user of the system. One type of system program very familiar to programmers is a **compiler** or language translator, which makes it possible to write programs in high-level languages like C, C++, Java or Ada. Another system program is generally responsible for the overall control of the computing system, including communicating with users and directing the execution of their programs. This program is known as the **operating system** (abbreviated **OS**). Except for very early computers or a few special-purpose categories, almost every computer is provided with some type of operating system. Some well-known operating systems you may have heard of include Unix (which has many variants, including Linux), Microsoft Windows (also several variants), Apple's MacOS, IBM's OS/390 (or MVS) and z-OS, Google's Android OS, and the Symbian OS used in many cell phones. This text is devoted to the study of systems like these.

1.2 THE ROLE OF AN OPERATING SYSTEM

As we have noted, a computing system consists of both hardware and software. The simplest view, which we will continue to follow unless stated otherwise, is that the computer is hardware and the operating system is software. To be precise, however, the exact boundary between hardware and software varies with each computer. Simple computers have been constructed in which many capabilities usually expected from the hardware, such as multiplication, had to be provided by software instead. On the other hand, driven by the decreasing cost of hardware and the increasing cost of software, many computers today include hardware functions once expected only from software, such as memory management, direct operations on various data structures, graphics and multimedia support, or even direct interpretation of high-level languages. Many common parts of operating systems may fall in this category.

Each hardware or software component of a computing system that is potentially useful to system users or their programs is called a **resource**. Some important resources include memory, files, disks, and printers. An operating system's primary purpose is to manage the resources of a computing system. Although most computer professionals will not be called upon to design new operating systems, most will work daily with computing systems whose behavior is greatly influenced by the type of OS provided. Many professionals will also have opportunities to influence their organization's choice of operating systems, to maintain an existing OS, or to adapt an operating system to the specific needs of a particular environment. For these reasons, a clear understanding of how operating systems work is important to the education of all computer scientists and engineers.

A computer without an operating system may be compared to a bus without a driver, a situation illustrated in Figure 1-1. Such a bus will still run; but passengers wanting to use the bus are faced with a complicated driving process rather than a comfortable ride. Mistakes are easily made, and may have serious consequences. If several passengers wish to use the same bus, they might fight over who will drive and where the bus should go.

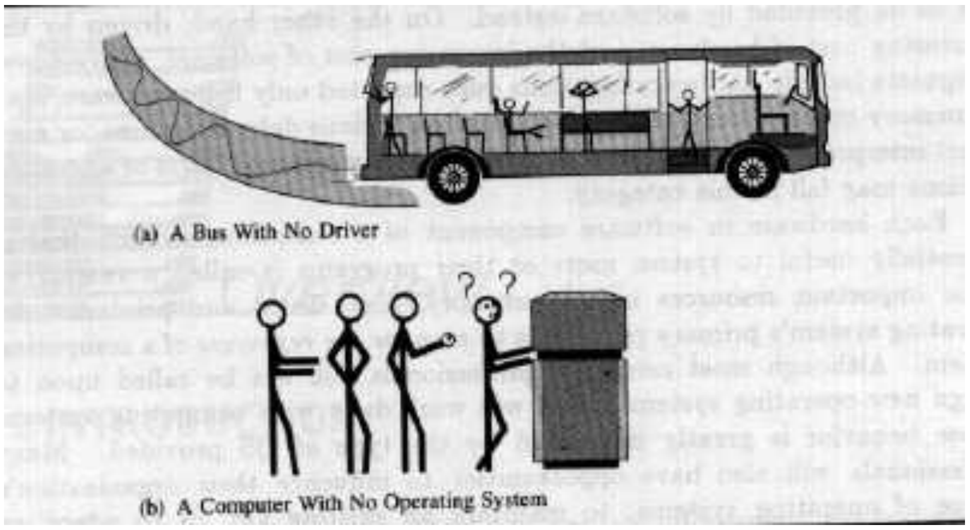

Figure 1-1: The Computer as a Driverless Bus

Much the same situation arises in a computing system when a group of users want to perform work on the computer. Without an operating system, each user must take control of the computer and create programs to handle all the messy details of each resource. It will be difficult to share the resources effectively, and users may fight over the right to use them. In many types of computing systems today, of course, there is only one user, but that user may be running several programs simultaneously, and the programs will compete for resources in the same way.

With a driver on board the bus, however, someone is in control. This improved situation is depicted in Figure 1-2. Passengers may provide the driver with high-level instructions ("Take me to High and Walnut") without the need to worry constantly about steering, gearshifts, brakes, or which route to follow. Most of these details are taken care of by the driver, and the bus is much more convenient to use. Even though passengers cannot drive the bus directly to their destination, but must be content with the driver's schedule, they will not usually complain (unless the bus is unduly late).

-

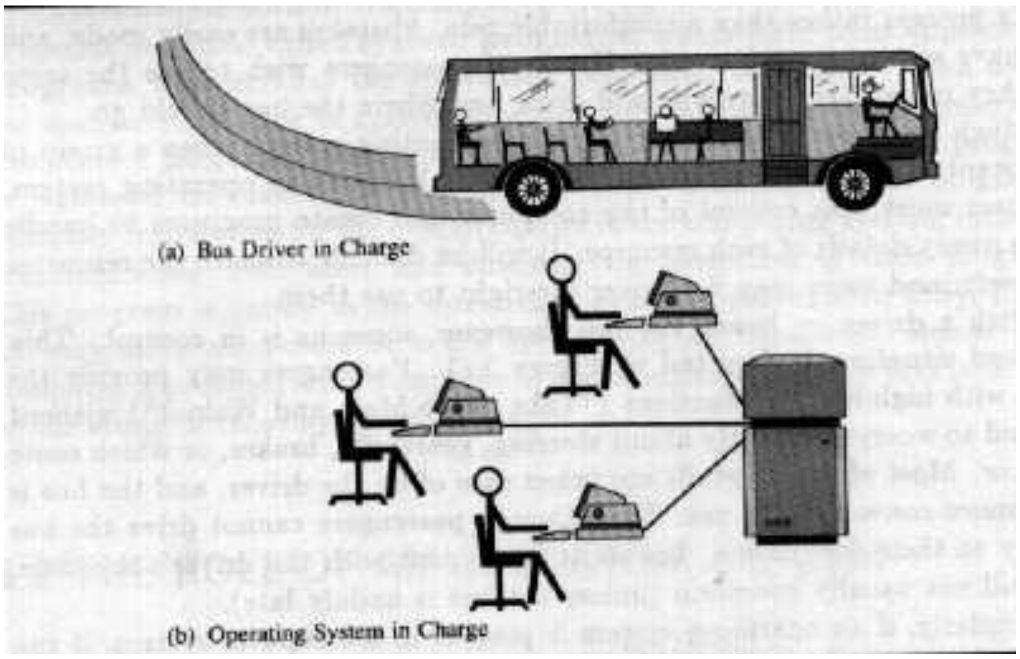

Figure 1-2: The Operating System as a Bus Driver

Similarly, if an operating system is present in a computer system, it can control the resources in detail and provide more effective sharing. Although users must defer to the decisions of the OS in some cases, everyone's work may be completed more quickly and more easily.

1.3 OPERATING SYSTEMS AS RESOURCE MANAGERS

We have said that an operating system is a manager of resources. This leads to some obvious questions:

1. What are the resources managed by the operating system?

2. What is the purpose of resource management?

The resources provided by a computing system may be grouped in two broad categories: **physical resources**, also called hardware resources, and **logical resources**, also known as software resources. As the name implies, physical resources are the permanent physical components of the computer system. The principal physical resources are:

- A processor (sometimes more than one)
- Main memory
- Input and output (I/O) devices such as displays, keyboards, printers, cameras, or network devices
- Secondary storage devices such as hard disks, CDs, or pocket drives
- Internal devices such as clocks and timers.

Logical resources are collections of information, such as data or programs. Logical resources must be stored within physical resources---for instance, within main or secondary memory. Resources of interest in this category may include:

- Units of work, such as jobs or interactive sessions
- Processes (programs in execution)
- Files (named sets of information)
- Shared programs and data objects
- Libraries that perform a variety of useful services.

The principal resources that must be managed by a typical general-purpose operating system are summarized in Figure 1-3.

PHYSICAL RESOURCES	**LOGICAL RESOURCES**
processors	jobs and sessions
main memory	processes
I/O devices and controllers	files
secondary storage	system services
clocks and timers	

Figure 1-3: Resources Managed by an Operating System

The resource management provided by an OS has two principal objectives. The first objective is to support **convenient use**. In the early days of computing, each user had to provide his or her own programs to manage each detail in the control and use of I/O devices and other common resources. Today's users of computers, by and large, are not programmers. They view the computer as a tool for saving their own valuable time, and they rightly expect such tools to be easy to use. They should not have to worry about complex details involved in resource management. The services provided to the users of today's computing systems must be easy to use, yet provide sufficient power and flexibility to meet each user's needs.

The second objective of resource management by an operating system is to support and enforce **controlled sharing**. When computer systems are shared by more than one user, programs must be prevented from interfering with each other in any way. Especially in a general-purpose computing environment, programs of any type may be run, including many which contain errors or try to do improper things. Allowing a program to have direct control of common resources without the supervision of an OS would place other programs at risk of interference or damage to their code or resources.

Controlled sharing of resources is a complex problem. Besides being properly controlled, the sharing must be efficient and fair. Resources should seldom have to wait unused when there are programs that need them. When a resource is needed by several programs, each one should get a fair turn.

1.4 EXTENSIONS OF THE HARDWARE

An operating system refines and adds to the capabilities of the computer it controls. The operating system also serves as an interface, or communication link, to the users of the computing system. Most users deal with the computer only with the help of, and subject to the control of, the operating system. Thus it is natural to think of an OS as an extension of the

-

computing hardware. This view is illustrated in Figure 1-4. The OS may be thought of as a layer on top of or surrounding the computer, a layer that separates and insulates the users and the computer from one another.

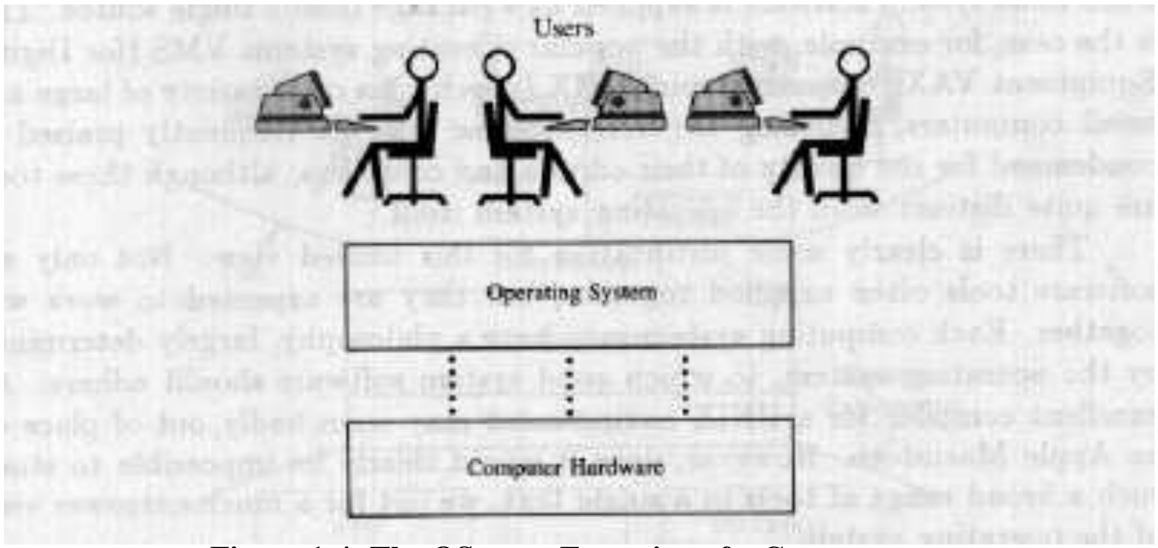

Figure 1-4: The OS as an Extension of a Computer

Although personal computer users may be somewhat aware of the differences between hardware and software, most users still find it convenient to think of their system as a single entity, and rarely distinguish the operating system from the computer hardware. If the computing system fails due to some problem, users observe only that the system is "down." They do not often care whether the problem involves hardware or software (unless it becomes necessary for them to help solve it).

Another natural reason for considering the OS to be an extension of the hardware in some systems is that part or all of the OS may actually be implemented in hardware, or at least stored permanently in a read-only memory. In such systems the OS cannot be removed from memory, and takes control whenever power is turned on. Since the OS is always available, it appears to most users that it is indeed part of the hardware.

1.5 OPERATING SYSTEMS IN PERSPECTIVE

To begin a study of operating systems, we must define the limits of system software responsibilities that we consider to fall under the proper definition of an OS. One possible view is to identify the OS with the complete computing environment. We would then consider the entire body of software supplied with a computing system, including editors, compilers, and other software tools and utilities, as parts of the OS itself. This view is often held by users, especially when most system software is supplied as a package from a single source. This is the case, for example, with most versions of UNIX, and to a lesser extent with Windows and Macintosh systems. These OSs are frequently praised or condemned for the quality of their tools and applications, although these tools are quite distinct from the operating system itself.

There is clearly some justification for this unified view. Not only are software tools often supplied together, but they are expected to work well together. Each computing system may have a philosophy, largely determined by the operating system, to which good system software should

-

adhere. An excellent compiler for a UNIX environment may seem badly out of place on a Macintosh. However, since it would clearly be impossible to study such a broad range of tools in a single text, we opt for a much narrower view of the operating system.

A different extreme taken by some texts is to view the OS as a very limited collection of essential resource management procedures, typically those that remain in main memory at all times and are entrusted with all of the privileges that the hardware can provide. These procedures are often referred to as the **kernel** of the operating system.

Our viewpoint is close, but not identical, to this second extreme. It is suggested by the discussion of the OS as extended machine, forming a layer between the computer and its users. An expanded view of this layer is shown in Figure 1-5. At each side of this layer the OS meets an interface. The interface to the computer hardware provides the OS with access to the physical resources, which form a **hardware environment** with which the OS must work. On the other side of this layer, the OS presents the system users with a **user interface**, which establishes a **user environment** that greatly determines how users perceive the system and how effectively it can meet their needs. Note that users interact with the OS both directly and through their application programs. The operations available for system communication and control, the services available to running programs, the form of terminal displays, and the philosophy established for use by software tools are all parts of this user environment.

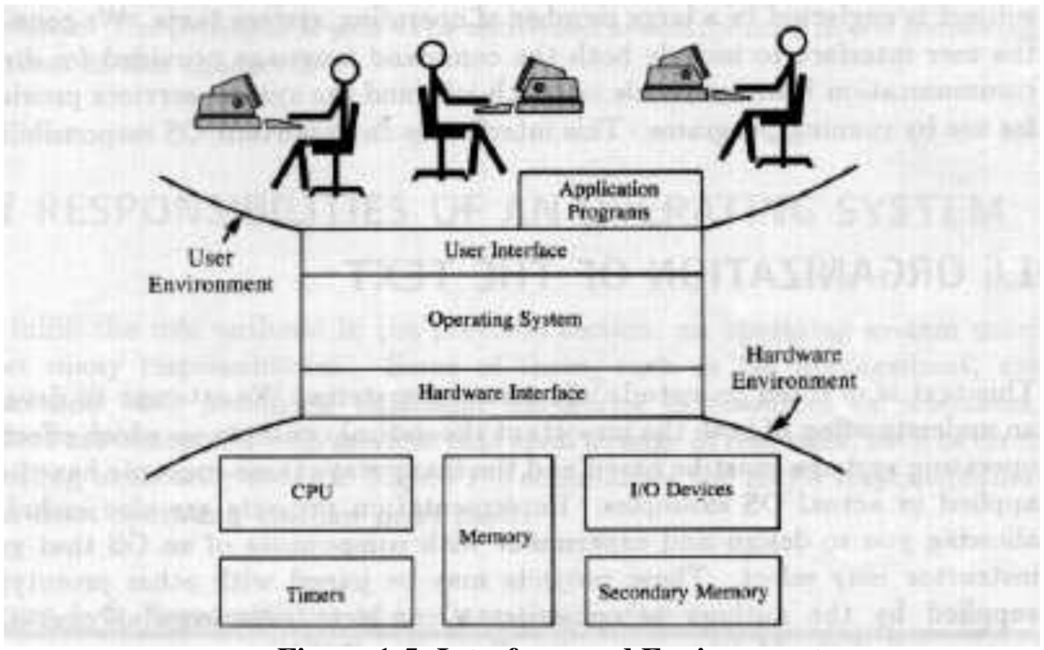

Figure 1-5: Interfaces and Environments

Our view of the OS will extend to all functions normally associated with this layer between the user environment and the hardware environment. The software in this layer extends and completes the capabilities of the computer and manages its physical resources. This management, ideally, is designed to provide the necessary control and convenience while remaining as flexible as possible. Its role is to provide a mechanism that can effectively manage each resource in accordance with a policy selected by those in charge of the specific computer system. This distinction between **mechanism**, which is a proper responsibility for operating systems, and **policy,** which generally is not, is an important distinction for effective OS design.

-

Although much of the OS software in this view may be part of the permanent, privileged kernel as described above, it can also include resource management programs and procedures which are loaded into memory as needed, or which operate with fewer privileges than the kernel itself.

In general, we restrict our view of the role of the OS to direct management of resources that must be controlled by a central authority for the benefit of all users. We deviate from this narrow view, however, in including treatment of two subjects that could arguably be excluded: **file systems** and the **user interface**.

Although files are not pure physical resources, they comprise an important category of shareable logical resources formed from physical storage devices, such as magnetic disks. The design of a file system imposes on all users the same high-level view of these storage devices. In some current operating systems, especially those designed to control multiple computers connected by a network, a file system is treated as a separate and independent component. However, files are an important system resource expected by most computer users, and we follow widely accepted practice in studying them together with operating systems.

The ultimate purpose of a computing system is to provide services to users. The operating system layer in Figure 1-5 would serve little purpose without the user interface at its top. Every OS must communicate with its users, and many have not done this job well. It is surprising that this subject is neglected in a large number of operating system texts. This interface is an important OS responsibility.

1.6 RESPONSIBILITIES OF AN OPERATING SYSTEM

To fulfill the role outlined in the previous section, an operating system must meet many responsibilities. Some of these, such as file management, are concerned with managing particular categories of resources or interfaces. Others are concerned with services that span a range of resources, such as error handling or security control. Figure 1-6 summarizes the major responsibilities that most operating systems must meet.

User Interface	Error Handling
Process Management	Reliability
Job & Session Management	Security
Device Management	Monitoring
Time Management	Accounting
Memory Management	System Management
File Management	

Figure 1-6: Responsibilities of an Operating System

-

The User Interface

The user interface provides a mechanism for direct interaction between users and the OS, or between users and other programs. This interaction may be based on typing lines of text, selecting menu items or graphic icons, speaking phrases, or some other appropriate paradigm. A closely related mechanism is the program interface, which provides services upon request to programs during execution. These services, requested by a mechanism similar to procedure calls, include such things as input and output, file access, and program termination requests.

The **user interface** establishes the "user friendliness" of the system. Commands or graphic elements that are consistent and easy to use will result in a user's more rapid acceptance of a computer system. Structures that are difficult to learn or understand quickly frustrate users and often cause them to move to other computer systems that are easier to use, whenever such systems are available. Although in some environments the user interface may be viewed as a separate, replaceable program, it is a critical element of a complete computing system.

The **program interface** defines the procedures and conventions programs must follow to request operating system services. Such things as the parameters required and the means of invoking the services of the operating system are defined by this interface. The program interface is often known as the **application program interface (API).**

The User Interface is examined in Chapter 3.

Process Management

The term **process** is used to describe a program in execution under the control of an operating system. Another widely used term for process is **task**. We will use the term process consistently throughout this text. Perhaps the most fundamental responsibility of any operating system is **process management**. When there is only one CPU, processes must either run one at a time to completion, or take turns using the CPU for a short period. Allowing processes to take turns in this manner is called **interleaved execution**; processes that share the CPU in this way are called **concurrent processes**. An OS that allows application programs to concurrently share the CPU is called a **multiprogramming (or multitasking) system**.

Controlling this interleaved execution so that work proceeds in a fair and effective way is the problem to be solved by **process scheduling**. This scheduling must be considered at several levels, from overall admission of new jobs into a system down to moment-by-moment allocation of the use of the CPU.

For various reasons, concurrent processes may interact with one another. One form of interaction occurs because resources are shared, and some resources required by one process may be in use by another. A second form of interaction occurs when processes deliberately seek to exchange information. Controlling the interaction that may occur among concurrent processes is an important responsibility for a multiprogramming OS.

We will look at basic process concepts in Chapter 4, and process scheduling in Chapter 5. Concurrency will be addressed later in the course as time permits.

Device Management

All computers have a variety of input, output and storage devices that must be controlled by the operating system. Printers, terminals, plotters, hard disks, floppy disks, magnetic tapes,

-

and communication devices are among the most common devices found in such systems today. In the past, additional devices such as punched card readers and paper tape readers were also prevalent.

In addition, many computers include internal devices that may be accessed to perform special functions. An important example is clocks or timers, which may be used to measure the timing relationships among various events. **Device management** encompasses all aspects of controlling these devices: starting operation, requesting and waiting for data transfers, responding to errors that may occur, and so on.

As might be suspected, the variety of devices that can be attached to a computer use many different commands and controls. The user cannot be expected to write the software to directly control such a wide range of devices. Furthermore, multiprogramming operating systems that support the sharing of a computer system by two or more processes cannot allow an application program to access these devices without going through the operating system program interface. Otherwise, the integrity of other users' and programs' data might be sacrificed (e.g., by a direct write request to a disk that intentionally or erroneously writes over another user's data). Device management, therefore, is a very important function of the operating system. Device support can be quite complex, and implementing such support consumes much of the time and effort in the implementation of any operating system.

A number of types of devices are covered in Chapter 9, and device management is addressed in Chapter 10.

In many cases, data transfers that have been started may continue while the CPU is performing other work. Completion of the transfer is then signaled by **interrupts**, events that literally "interrupt" the running program, requiring it to take time out to handle the event before resuming the interrupted work it was doing. Interrupts will be considered in Chapter 6.

Time Management

Another important OS responsibility is **time management**, that is, controlling the time and sequence for various events. A special category of I/O device is the **timer**, whose role is to measure time and cause events to occur at specific times. We will look at timers and time management in Chapter 8.

Memory Management

Memory management is the responsibility of controlling the use of main memory, a critical resource in any computer system. Since secondary storage devices, such as disks and tapes, are used in various ways to extend the storage available in main memory, the subject of memory management must deal with these devices as well. The performance of operating systems can be greatly affected by the way the memory is managed.

Early operating systems allowed only one program to be in progress at a time, so memory management was relatively simple. The operating system used what it needed of main memory to store its own information, and the program could have the rest. Responsibility for managing the available memory rested solely with the application program. Often the OS had no means of protecting even its own memory. If a program interfered with OS memory, it could do harm only to itself, since a failure of the operating system would harm only that single program.

-

The introduction of multiprogramming systems changed all that, because main memory had to be shared among several processes. In such systems the OS must be responsible for allocating memory and for protecting processes from one another.

Early forms of memory management were concerned primarily with allocating portions of main memory to each process when the process started. Newer strategies allow additional areas of memory to be allocated and removed as desired, and provide for temporary **swapping** of programs and their data from main memory to a disk when not immediately needed. This strategy makes the memory space available for more immediate needs.

As an evolution of swapping techniques, **virtual memory** systems have now become common. These systems can automate the swapping process by a combination of hardware and software techniques, providing powerful solutions to a full range of memory management problems.

Basic memory management will be covered in Chapter 7. Virtual memory management is examined in Chapter 13.

File Management

Information stored by a computer system is usually organized into files. File management is concerned with all aspects of reading, writing, organizing, and controlling the access to information stored in files. Another major function of such management is to provide security---that is, protection of information from improper access. Security has been one of the weakest aspects of file management in many operating systems. File management is examined in Chapter 12.

The remaining areas are also important OS responsibilities. Text chapters are available for most of these areas. Due to time limitations, however, they will not be covered in this course.

Job and Session Management

A complete unit of work performed or submitted by a user may include a series of programs, carried out by a set of processes that may proceed one at a time, or concurrently in some cases. Such a unit is called a **job** when submitted to the OS all at once, as was common in early "batch" systems. When a set of work is specified by a series of commands typed by a user at a terminal, it may more aptly be called a **session**. For various reasons, it is sometimes desirable to manage jobs and sessions as complete units. For example, the OS may need to decide when a new job or session can be started, and it may need to associate the complete job or session with a particular user for authorization or accounting purposes.

Error Handling

Operating systems must deal with a variety of errors that can occur in a computer system. Such errors include hardware errors in devices, memory, and channels, and software errors in application and system programs. Software errors can be a result of such things as arithmetic overflow, invalid instructions, or references to invalid memory.

-

Error handling is the process of detecting and recovering from such errors. Effective error handling must try to detect as many types of errors as possible. When errors are detected, reasonable attempts should be made to recover and continue, without such drastic actions as terminating the offending process or stopping the entire system. Whenever possible, failures in hardware components should result in the isolation of the faulty component without affecting the rest of the system operation. Software errors should be recognized, corrected if possible, and prevented from causing damage to other programs and resources. In many cases, error detection and recovery should be as invisible as possible to the user. However, some types of errors should be reported to the application program so that appropriate action can be taken.

Reliability and Security

An operating system may have partial responsibility for ensuring that computer systems continue to operate correctly and data and programs are protected from damage or improper access. These goals may need to be met even when errors or failures exist in hardware or software, or unauthorized persons deliberately seek to corrupt or gain access to the system. These system responsibilities are known as reliability and security. Their importance can vary greatly from one system to another. Providing extremely good reliability and security can be very costly.

The importance of reliability--keeping a computer system running correctly--is evident. The importance of ensuring the security of information stored in files has already been mentioned. In addition to this critical aspect of security, the operating system must protect one process from another, the operating system from a process, a process from the operating system, and the operating system from itself. The most obvious way an OS does this is to isolate memory areas for processes and for itself; another is to protect against the use of privileged instructions and system resources by unauthorized processes. Access to OS services must also be controlled by the OS. Other areas of security include those of session and job management, which require the appropriate identifications, passwords and other mechanisms before use of the system is granted.

Monitoring and Accounting

Most sophisticated operating systems, especially in environments where computing is considered to have significant costs, include some facilities for monitoring the behavior of the system as it proceeds and for keeping records of that behavior for various purposes. One important historical purpose of this monitoring is accounting, keeping track of resource use by each user so that users can be billed for the services of the computing system. This has become much less common, but there are other important purposes for monitoring system activities.

One additional purpose is to measure the performance of the system--that is, the effectiveness with which it completes its work within the limitations of available resources. Such monitoring may identify adjustments that should be made to keep that performance as good as possible. Another increasingly critical need is to support forensic analysis, in case of a system failure or security violation.

System Management

Some key responsibilities that fall under the heading of system management include methods for initial installation of operating systems; adjusting their characteristics to each particular environment; resuming normal operation after a shutdown or a serious error; maintaining records of system usage by each job or user; maintaining user information and other sensitive records; and providing appropriate responses to any unusual situations that may occur.

-

An important goal in real operating systems is to maintain the best possible performance. This leads to the need to measure or predict performance, and to use this information to tune the system for the best performance that is practically attainable.

This completes our overview of the usual OS activities and responsibilities. We will now move on to study a number of these areas in detail.

FOR FURTHER READING

A very large body of literature addresses operating system topics. We will cite references to work that focuses on specific issues in the appropriate chapters of this text. Here we list some books that offer fairly broad coverage.

A number of early texts provide good descriptive coverage of selected topics in operating systems. Shaw [1974] is an older but classic introduction to many important concepts; an updated version is Bic and Shaw [1988]. Another classic work is Madnick and Donovan [1974], which includes especially good treatment of memory management.

More recent texts providing good overviews of operating system concepts include Tanenbaum [2007], Stallings [2008], Silberschatz et al [2008], Davis [2004], Nutt [2003], Stuart [2008] and Schlesinger and Garrido [2007].

Several texts emphasize the study of example code and algorithms, including detailed study of specific operating systems. Tanenbaum [2006] provides detailed code for the pedagogical MINIX operating system, which was the chief inspiration for Linux.

A particularly good study of a specific real operating system, an early version of UNIX, is found in Bach [1986]. Other good treatments of important individual OSs include Sewell [1992] (VMS), Hart [2010] (Windows), Bovet and Cesati [2005] (Linux), and Neville-Neil (Free BSD, the Unix variant used in Macintosh OSX). Many of the general texts cited above also include detailed case studies.

More rigorous studies of theoretical aspects of OS design, including comparative studies of algorithms, are given in a number of advanced texts. Classics in this category include Coffman and Denning [1973], Brinch Hansen [1973], and Habermann [1976]. Significant later entries include Maekawa et al [1987] and Chow and Johnson [1998]. These texts focus on theoretical treatments of concurrent programming, scheduling, virtual memory management, performance analysis, and other selected topics.

-

REVIEW QUESTIONS

1. Why is an operating system such an important part of most computing systems?
2. State the principal purpose of an operating system in a single sentence.
3. Explain briefly the two principal objectives of resource management by operating systems.
4. List and briefly explain five categories of resources that are managed by a typical general-purpose OS.
5. Identify four types of OS responsibilities other than management of specific resources.
6. Explain two reasons why implementation and maintenance of operating system may raise difficulties not found in most other software systems.

ASSIGNMENTS

1. Are there any computing environments in which operating systems might not be necessary or even appropriate? Explain your answer.
2. Analogies are always imperfect. Discuss the limitations of the bus driver analogy given in this chapter. Propose a different analogy that might overcome some of these limitations.
3. According to the view of the OS boundaries taken by this text, which of the following would be considered part of the OS, and which would not? Justify your answers.

   a. A scheduler that chooses which process to run next
   b A program for opening files that is loaded only when needed
   c. A loader for application programs
   d. A C/C++ compiler
   e. A translator for a scripting language
   f. A text editor
   g. A database management system
   h. A login program that checks the authorization of those attempting to use the computer
   i. A mathematical subroutine library

4. In your experience with computer use, under what circumstances have you found it necessary or helpful to think of the operating system as a separate part of the total computer system? Explain.

5. For each of the following types of OS, identify some responsibilities that would probably not be important, and some that would be especially important:

   a. An average laptop or desktop computer
   b. A mobile device that runs only one program at a time
   c. A computer embedded within an aircraft
   d. A large "mainframe" computer that accepts work only in batches and returns results at a later time
   e. A timesharing system serving many users with a Unix or Linux OS.

-