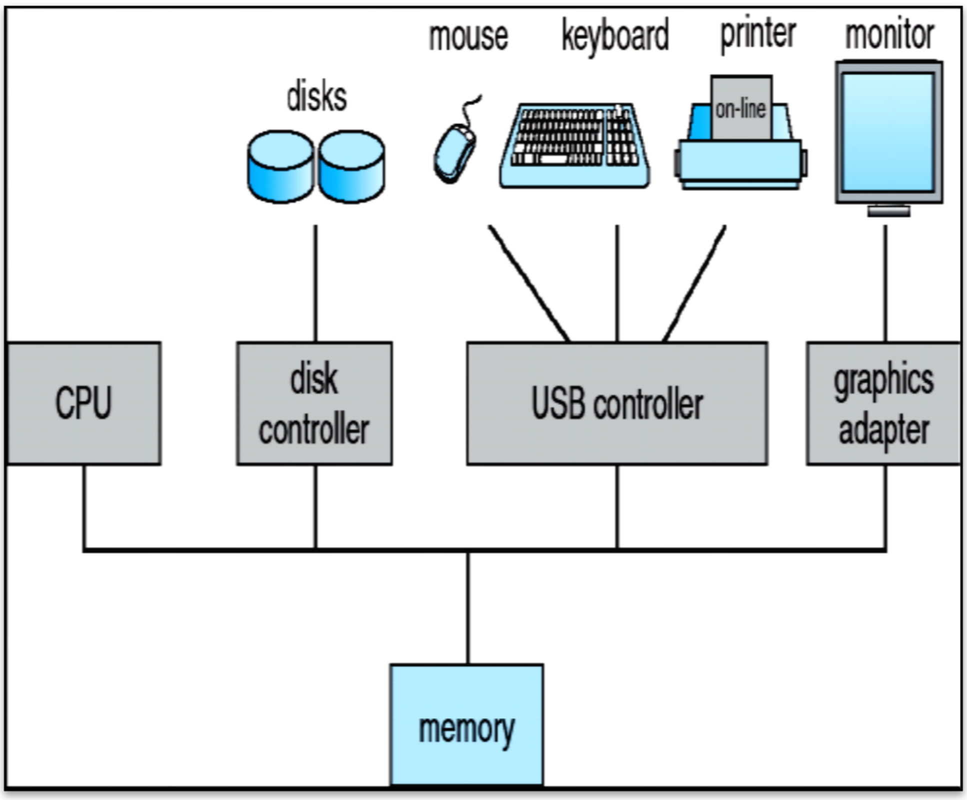
Chapter 1 *Introduction*

This chapter will discuss the following concepts:



1.1 What are the Operating Systems?

An **operating system** is a program that manages the computer hardware. It also acts as an intermediary between the computer user and the computer hardware.

1.2 What Operating Systems Do?

We begin our discussion by looking at the operating system's role in the overall computer system. A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users (Figure 1.1).

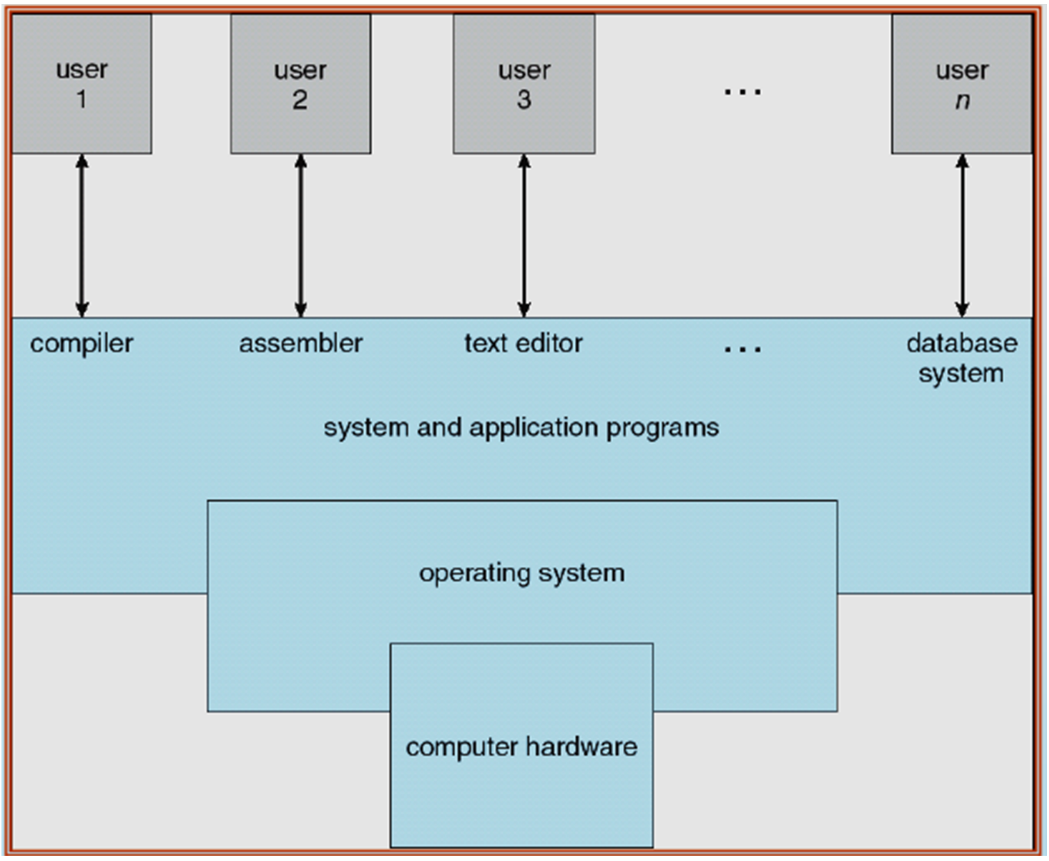


Figure 1.1 abstract views of the components of a computer system.

- **The hardware**: such as the central processing unit (**CPU**), the memory, and the input/output (**I/O**) devices—provide the basic computing resources for the system.
- **The application programs**: such as word processors, spreadsheets, compilers, and web browsers—define the ways in which these resources are used to solve users' computing problems.
- **The operating system:** controls and coordinates the use of the hardware among the various application programs for the various users.

To understand more fully the operating system's role, we next explore operating systems from two viewpoints: that of the **user** and that of the **system**.

- **User View**

The user's view of the computer varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work (or play) that the user is performing. In this case, the operating system is designed mostly for **ease of use**.

- **System View**

From the computer's point of view we can view an operating system as a **resource allocator**. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

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

Therefor operating system is the one program running at all times on the computer (usually called the **kernel**), with all else being systems programs and application programs.

1.3 Computer-System Organization

1.3.1 Computer-System Operation

A modern general-purpose computer system consists of one or more **CPUs** and a number of **device controllers** connected through a common bus that provides access to **shared memory** (Figure 1.2). Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, and video displays). To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

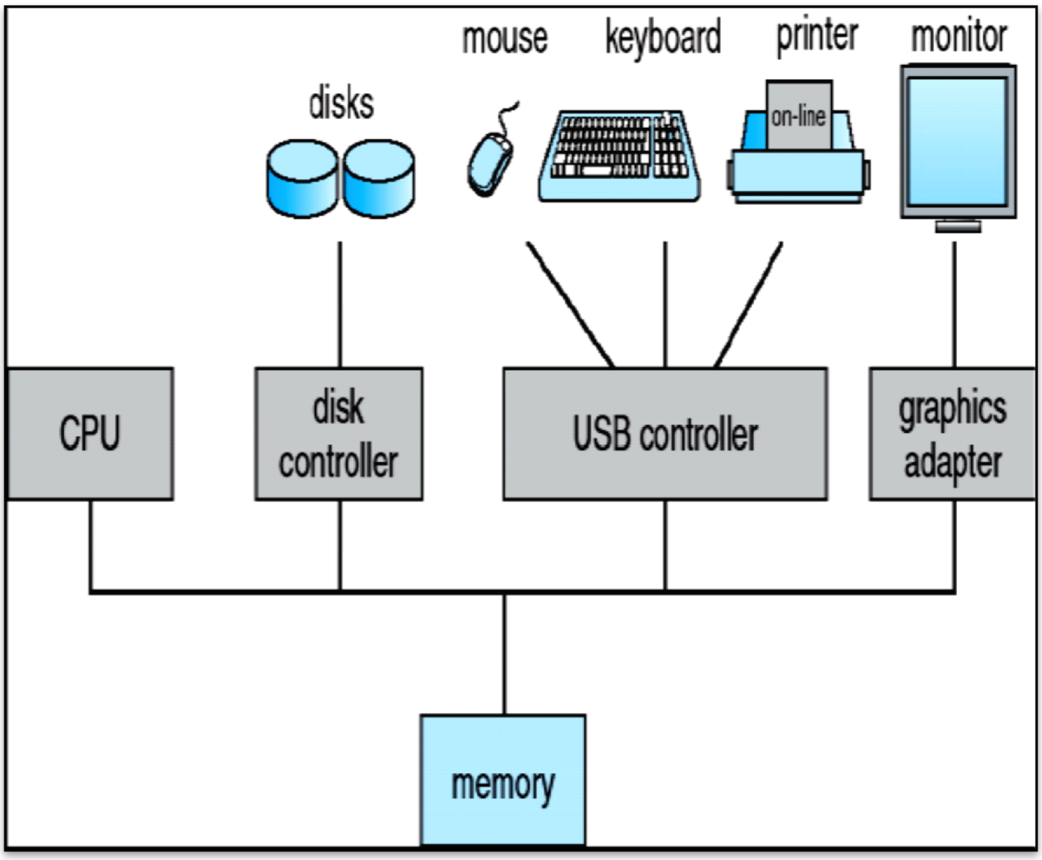


Figure 1.2 A modern computer system.

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or **bootstrap program**, tends to be simple. Typically, it is stored in read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**), known by the general term **firmware**, within the computer hardware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and to start executing that system. To accomplish this goal, the bootstrap program must locate and load into memory the operating system kernel. The operating system then starts executing the first process, such as "**init**," and waits for some event to occur.

The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a **system call** (also called a **monitor call**).

When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the **service routine** for the interrupt is located. The interrupt service routine executes; on completion, the CPU resumes the interrupted computation.

1.3.2 Storage Structure

Computer programs must be in main memory (also called random-access memory or **RAM**) to be executed. Main memory is the only large storage area (millions to billions of bytes) that the processor can access directly. It commonly is implemented in a semiconductor technology called dynamic random-access memory (**DRAM**), which forms an array of memory words. Each word has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses. Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:

1. Main memory is usually too small to store all needed programs and data permanently.
2. Main memory is a ***volatile*** storage device that loses its contents when power is turned off or otherwise lost.

Thus, most computer systems provide **secondary storage** as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently.

The most common secondary-storage device is a **magnetic disk**, which provides storage for both programs and data.

In a larger sense, however, the storage structure such as—**registers**, **main memory**, and **magnetic disks**—is only one of many possible storage systems. Others include **cache memory**, **CD-ROM**, **magnetic tapes**, and so on. Each storage system provides the basic functions of storing a datum and of holding that datum until it is retrieved at a later time. The main differences among the various storage systems lie in **speed, cost, size, and volatility**.

The wide variety of storage systems in a computer system can be organized in a hierarchy (Figure 1.3) according to **speed and cost**. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases. This trade-off is reasonable; if a given storage system were both faster and less expensive than another—other properties being the same—then there would be no reason to use the slower, more expensive memory. Now, the magnetic tape and **semiconductor memory** have become faster and cheaper. The top four levels of memory in (Figure 1.3) may be constructed using semiconductor memory.

Another form of electronic disk is **flash memory**, which is popular in cameras and personal digital assistants (**PDAs**), in robots, and increasingly as removable storage on general-purpose computers.

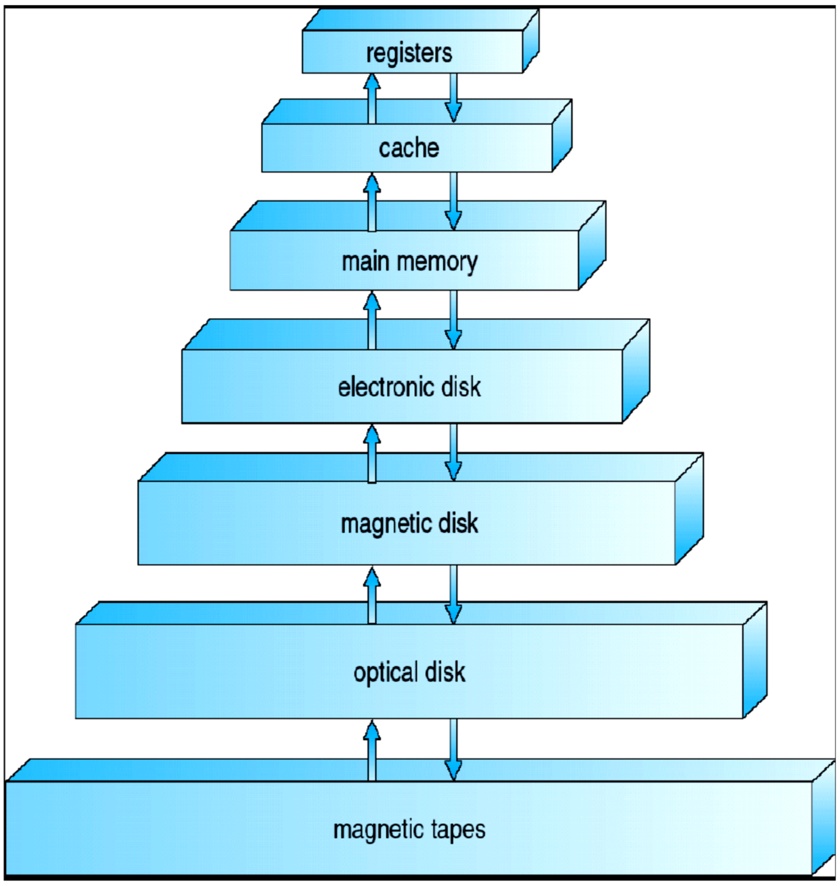


Figure 1.3 Storage-device hierarchy.

1.4 Computer-System Architecture

A computer system may be organized in a number of different ways, which we can categorize roughly according to the number of general-purpose processors used.

1.4.1 Single-Processor Systems

Most systems vise a single processor. The variety of single-processor systems may be surprising, however, since these systems range from PDAs through mainframes. On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes. Almost all systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers; or, on mainframes, they may come in the form of more general-purpose processors, such as I/O processors that move data rapidly among the components of the system.

The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor. If there is only one general-purpose CPU, then the system is a single-processor system.

1.4.2 Multiprocessor Systems

Although single-processor systems are most common, **multiprocessor systems** (also known as **parallel systems** or **tightly coupled systems**) are growing in importance. Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices. Multiprocessor systems have three main advantages:

1. **Increased throughput:** By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with *N* processors is not *N,* however; rather, it is less than N.

2. **Economy of scale**: Multiprocessor systems can cost less than equivalent multiple single processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them

than to have many computers with local disks and many copies of the data.

3. **Increased reliability:** If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

The multiple-processor systems in use today are of two types. Some systems use **asymmetric multiprocessing**, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines **a master-slave** relationship. The master processor schedules and allocates work to the slave processors.

The most common systems use **symmetric multiprocessing** (**SMP**), in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors. (Figure 1.4) illustrates a typical SMP architecture.

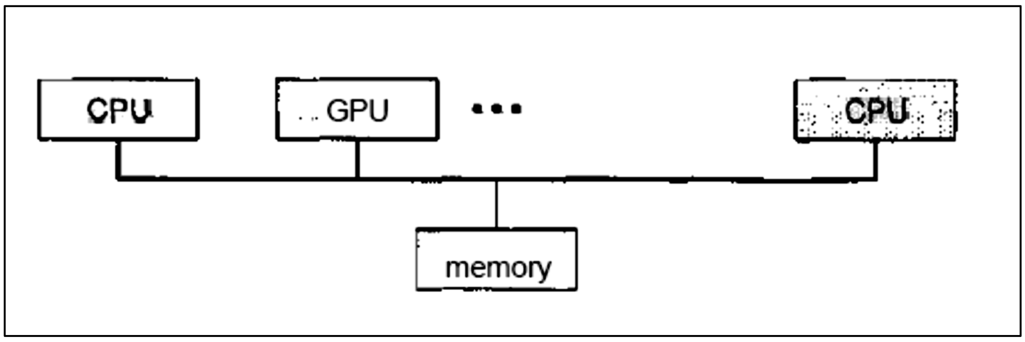


Figure 1.4 Symmetric multiprocessing architecture

1.5 Operating-System Structure

One of the most important aspects of operating systems is the ability to multi program. A single user cannot, in general, keep either the CPU or the I/O devices busy at all times. **Multiprogramming** increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.

The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure 1.5). This set of jobs can be a subset of the jobs kept in the job pool—which contains all jobs that enter the system—. The operating system picks and begins to execute one of the jobs in memory.

In a **non multiprogrammed** system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another job. When that job needs to wait, the CPU is switched to another job, and so on. As long as at least one job needs to execute, the CPU is never idle.

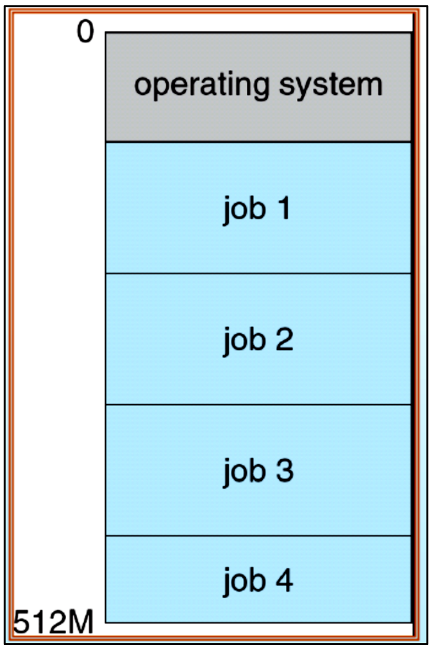


Figure 1.5 Memory layout for a multiprogramming system.

Time sharing (or **multitasking**) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

Time sharing requires an **interactive computer system**, which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using an input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the response time should be short—typically less than one second.

A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.

1.6 Operating-System Operations

Modern operating systems are interrupt driven. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are almost always signaled by the occurrence of an interrupt or a **trap**.

A **trap (or** an **exception)** is a software-generated interrupt caused either by an error (for example, **division by zero** or **invalid memory access**) or by a specific request from a user program that an operating-system service be performed.

More subtle errors can occur in a multiprogramming system, where one erroneous program might modify another program, the data of another program, or even the operating system itself.

A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

1.6.1 Dual-Mode Operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of **operating-system code** and **user defined code**. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least, we need two separate **modes** of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**). A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. This is shown in (Figure 1.6).

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as **privileged instructions**. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. Some examples of a privileged instruction include **I/O control**, **timer management**, and **interrupt management**.

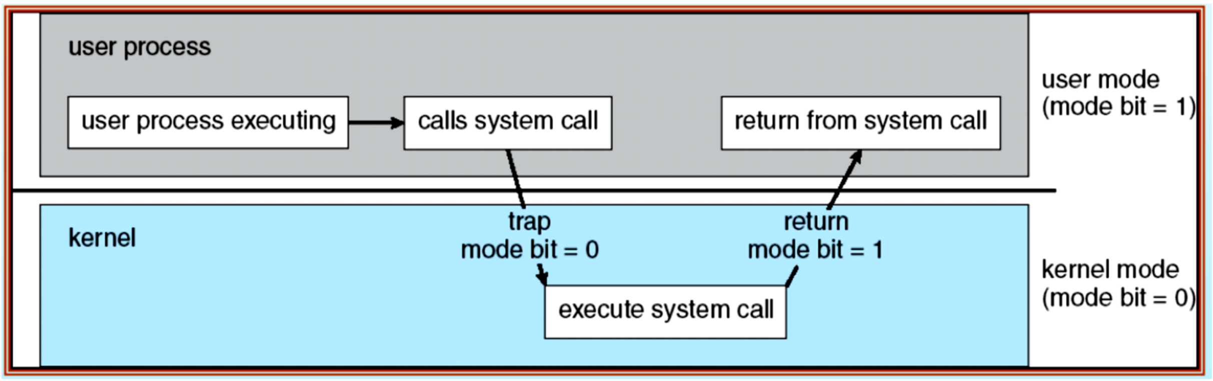


Figure 1.6 Transition from user to kernel mode.

1.6.2 Timer

We must ensure that the operating system maintains control over the CPU. We must prevent a user program from getting stuck in an infinite loop or not calling system services and never returning control to the operating system.

To accomplish this goal, we can use a **timer**. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A **variable timer** is generally implemented by a fixed-rate clock and a counter.

1.7 Computing Environments

1.7.1 Traditional Computing

Just a few years ago, At home, most users had a single computer with a slow modem connection to the office, the Internet, or both.

Today, network-connection speeds once available only at great cost are relatively inexpensive, giving home users more access to more data. These fast data connections are allowing home computers to serve up web pages and to run networks that include printers, client PCs, and servers.

1.7.2 Client-Server Computing

As PCs have become faster, more powerful, and cheaper, designers have shifted away from centralized system architecture. Terminals connected to centralized systems are now being supplanted by PCs. Correspondingly; user interface functionality once handled directly by the centralized systems is increasingly being handled by the PCs. As a result, many of today's systems act as **server systems** to satisfy requests generated by **client systems**. This form of specialized distributed system, called **client-server** system, has the general structure as in (Figure 1.7.)

Server systems can be broadly categorized as **compute servers** and **file servers**:

- The **compute-server system** provides an interface to which a client can send a request to perform an action (for example, read data); in response, the server executes the action and sends back results to the client. A server running a database that responds to client requests for data is an example of such a system.
- The **file-server system** provides a file-system interface where

clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

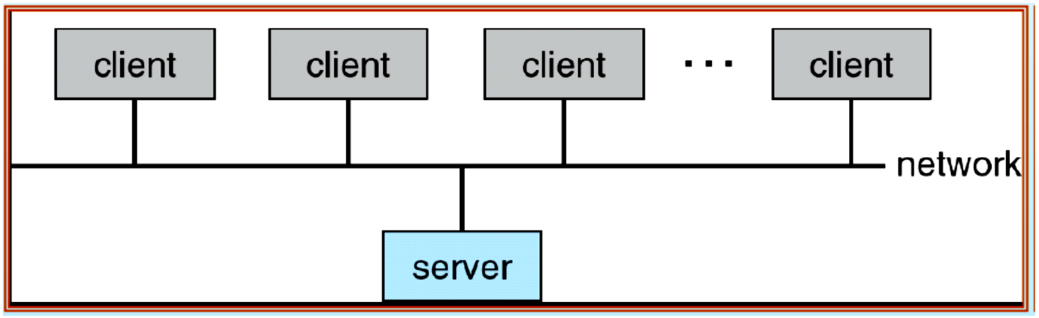


Figure 1.7 General structure of a client-server system.

1.7.3 Peer-to-Peer Computing

Another structure for a distributed system is the peer-to-peer (P2P) system model. In this model, clients and servers are not distinguished from one another; instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.

1.7.4 Web-Based Computing

The Web has become ubiquitous, leading to more access by a wider variety of devices than was dreamt of a few years ago. PCs are still the most prevalent access devices, with workstations, handheld PDAs, and even cell phones also providing access.

Web computing has increased the emphasis on networking. Devices that were not previously networked now include wired or wireless access. Devices that were networked now have faster network connectivity.

End of chapter 1