UNIT-I

COMPUTER SYSTEM AND OPERATING SYSTEM OVERVIEW

Syllabus

1. *Overview of computer operating systems*
2. *Operating system functions*
3. *Protection and Security*
4. *Distributed systems*
5. *Special purpose systems*
6. *Operating system structures*
7. *System calls*
8. *Operating system Generation*

Introduction:-

An *operating system* acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a *convenient* and *efficient* manner.

An operating system is software that manages the computer hardware.

Operating system goals:

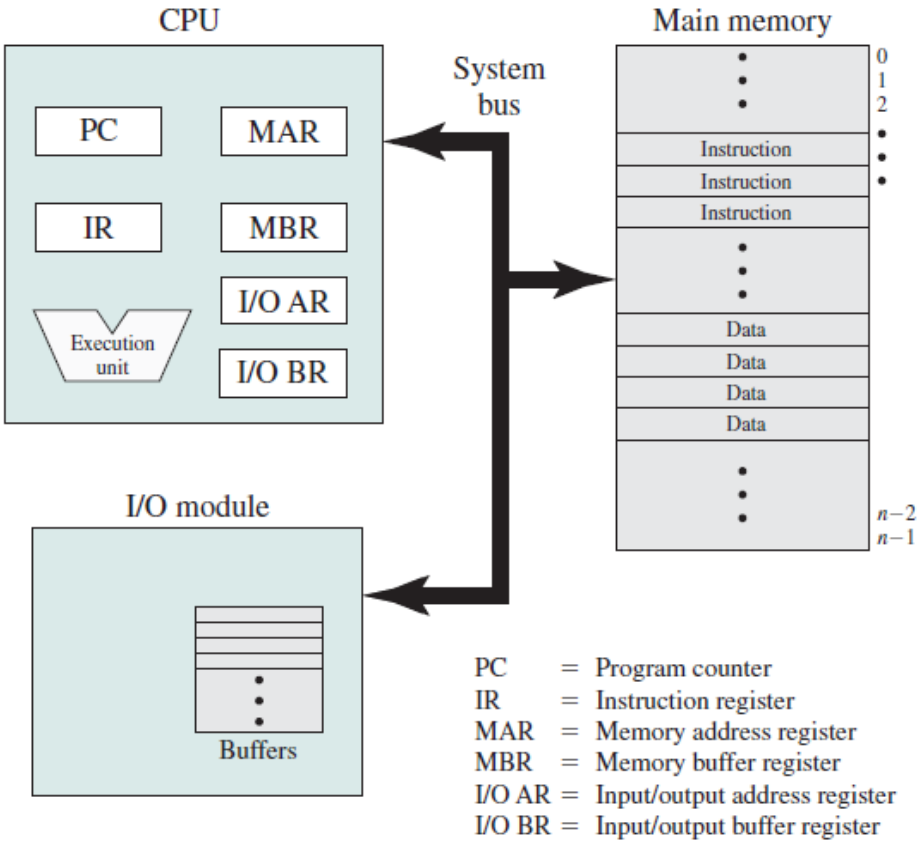
- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner

Overview of computer system hardware

At a top level, a computer consists of processor, memory, and I/O components, with one or more modules of each type. These components are interconnected in some fashion to achieve the main function of the computer. Thus, there are four main structural elements:

- **Processor:** Controls the operation of the computer and performs its data processing functions. When there is only one processor, it is often referred to as the **central processing unit** (CPU).
- **Main memory:** Stores data and programs. This memory is typically volatile; that is, when the computer is shut down, the contents of the memory are lost. Main memory is also referred to as *real memory* or *primary memory.*
- **I/O modules:** Move data between the computer and its external environment. The external environment consists of a variety of devices, including secondary memory devices (e.g., disks), communications equipment, and terminals.
- **System bus:** Provides for communication among processors, main memory, and I/O modules.

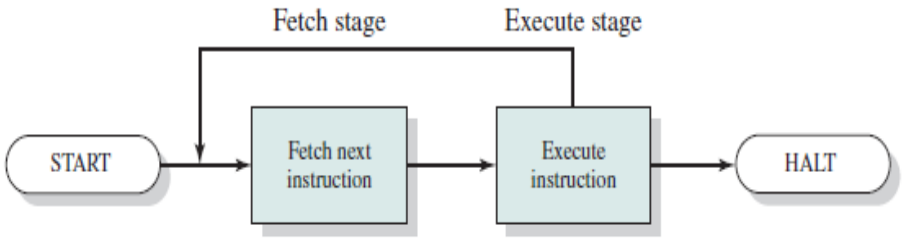
Figure depicts these top-level components.



Instruction Execution

A program to be executed by a processor consists of a set of instructions stored in memory. Instruction processing consists of two steps: The processor reads (*fetches*) instructions from memory one at a time and executes each instruction. Program execution consists of repeating the process of instruction fetch and instruction execution. Instruction execution may involve several operations and depends on the nature of the instruction.

The processing required for a single instruction is called an *instruction cycle.* Using a simplified two-step description, the instruction cycle is depicted in below Figure .



Basic Instruction Cycle

Instruction fetch and Execute

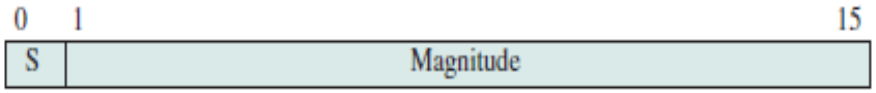
At the beginning of each instruction cycle, the processor fetches an instruction from memory. Typically, the program counter (PC) holds the address of the next

instruction to be fetched. Unless instructed otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.

.The fetched instruction is loaded into the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

• **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
• **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
• **Data processing:** The processor may perform some arithmetic or logic operation on data.
• **Control:** An instruction may specify that the sequence of execution be altered.

(a) Instruction format



(b) Integer format

(c) Internal CPU registers
Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(d) Partial list of opcodes
0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the IR and the PC is incremented.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded from memory. The remaining 12 bits (three hexadecimal digits) specify the address, which is 940.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

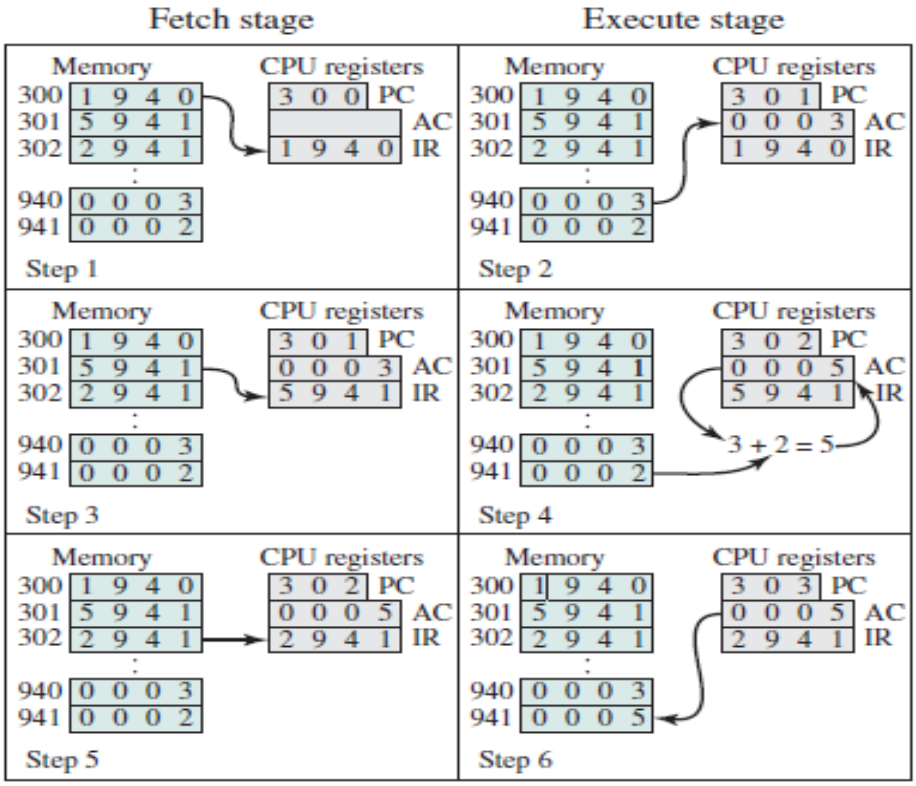


Figure 1.4 **Example of Program Execution** (contents of memory and registers in hexadecimal)

Interrupts

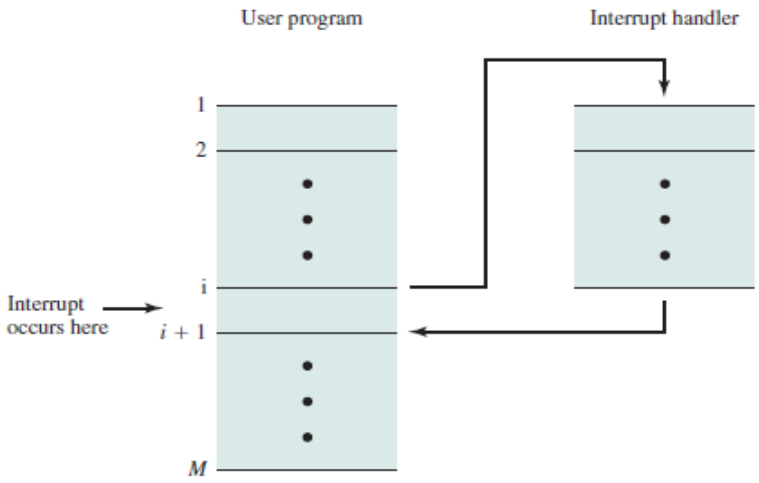
A signal that causes the control unit to branch to a specific location to execute code to service the occurrence of an external condition.

Classes of interrupts

- Hardware failure: Generated by a failure, such as power failure or memory parity error.
- I/o: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety and error conditions.
- Timer: Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
- Program: Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero.

Interrupts and the Instruction Cycle

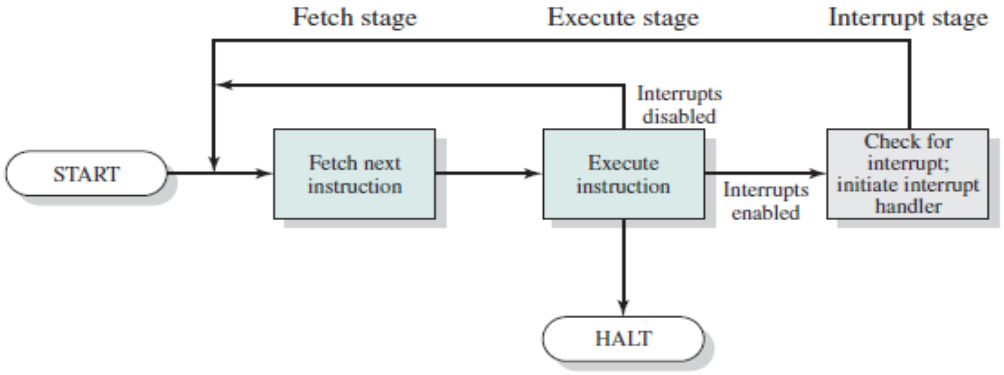
With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress. Consider the flow of control in below Fig.



Transfer of Control via Interrupts

For the user program, an interrupt suspends the normal sequence of execution. When the interrupt processing is completed, execution resumes (above Fig). Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the OS are responsible for suspending the user program and then resuming it at the same point.

To accommodate interrupts, an *interrupt stage* is added to the instruction cycle, as shown in below Figure.



Instruction Cycle with Interrupts

In the interrupt stage, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.

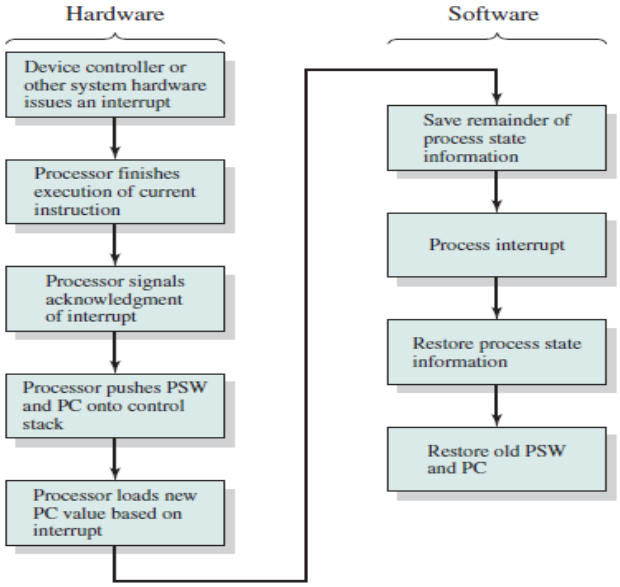
If no interrupts are pending, the processor proceeds to the fetch stage and fetches the next instruction of the current program.

If an interrupt is pending, the processor suspends execution of the current program and executes an *interrupt handler* routine. The interrupt-handler routine is generally part of the OS.

Interrupt Processing

1. The device issues an interrupt signal to the processor.
2. The processor finishes execution of the current instruction before responding to the interrupt.

3. The processor tests for a pending interrupt request, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.



Simple Interrupt Processing

4. It saves information needed to resume the current program at the point of interrupt. The minimum information required is the program status word (PSW) and the location of the next instruction (PC) to be executed.
5. The processor then loads the PC based on interrupt.
6. At this point, the program counter and PSW relating to the interrupted program have been saved on the control stack.
7. The interrupt handler may now proceed to process the interrupt.
8. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers.
9. The final act is to restore the PSW and program counter values from the stack.

1.1 Overview of computer operating system

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* in the below Figure.

- **Hardware -** Provides basic computing resources (CPU, memory, I/O devices).
  - Central processing unit (CPU) –It controls the operation of the computer and performs its data processing functions. When there is only one processor, it is often referred to as CPU.
  - Memory-Memory stores data and programs. It is also referred to as real memory or primary memory.
  - Input/output (I/O) devices- I/O devices moves data between the computer and its external environment.

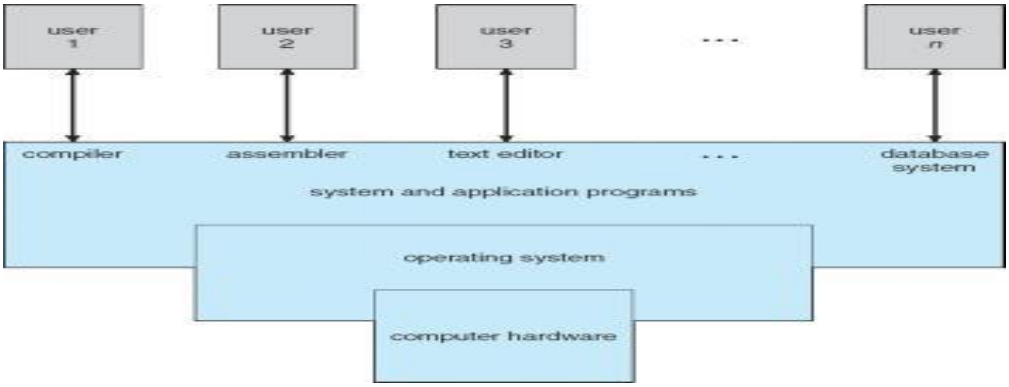


Fig. Abstract view of the components of a computer system.

.

- **Application programs - S**uch as word processors, spreadsheets, compilers, and web browsers—define the ways in which these resources are used to solve users' computing problems.
- **Operating system –** It controls and coordinates the use of the hardware among the various application programs for the various users.
- **Users -** People, machines, other computers.

We can also view a computer system as consisting of hardware, software, and data. The operating system provides the means for proper use of these resources in the operation of the computer system.

An operating system is similar to a *government.* Like a government, it performs no useful function by itself. It simply provides an *environment* within which other programs can do useful work.

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,** with some attention paid to performance and none paid to **resource utilization**—how various hardware and software resources are shared.

In other cases, a user sits at a terminal connected to a **mainframe** or **minicomputer.** Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.

In still other cases, users sit at **workstations** connected to networks of other workstations and **servers.** These users have dedicated resources at their disposal, but they also share resources such as networking and servers—file, compute, and print

servers. Therefore, their operating system is designed to compromise between individual usability and resource utilization.
Recently, many varieties of handheld computers have come into fashion.

System View

From the computer's point of view, the operating system is the program most intimately involved with the hardware. In this context, 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.

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.

Operating System Definitions

- OS is a **resource allocator**
  - Manages all resources.
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls the execution of user programs and operations of I/O devices.
- "The one program running at all times on the computer" is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program.
- **Computer startup - Bootstrap program** is loaded at power-up or reboot.
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

Operating-System Services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.

❖ One set of operating-system services provides functions that are helpful to the user.

- User interface. Almost all operating systems have a **user interface** (UI). This interface can take several forms. One is a **command-line interface (CLI),** which uses text commands and a method for entering them (say, a program to allow entering and editing of commands). Another is a **batch interface,** in which commands and

directives to control those commands are entered into files, and those files are executed. Most commonly a **graphical user interface (GUI)** is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.

• Program execution**.** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

• I/O operations**.** A running program may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired . For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

• File-system manipulation. The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership.

• Communications. There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communications may be implemented via *shared memory* or through *message passing*, in which packets of information are moved between processes by the operating system.

• Error detection. The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time).

❖ Another set of operating-system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself. Systems with multiple users can gain efficiency by sharing the computer resources among the users.

Resource allocation**.** When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There may also be routines to allocate printers, modems, USB storage drives, and other peripheral devices.

Accounting. We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting (so that users

can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and security**.** The owners of information stored in a multiuser or networked computer system may want to control use of that information. When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources. It extends to defending external I/O devices, including modems and network adapters, from invalid access attempts and to recording all such connections for detection of break-ins.

Computer System Organization

➢ Computer-system operation

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 the operating system kernel into memory.
- ❖ 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.
- ❖ Interrupts are an important part of computer architecture. Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine (ISR).

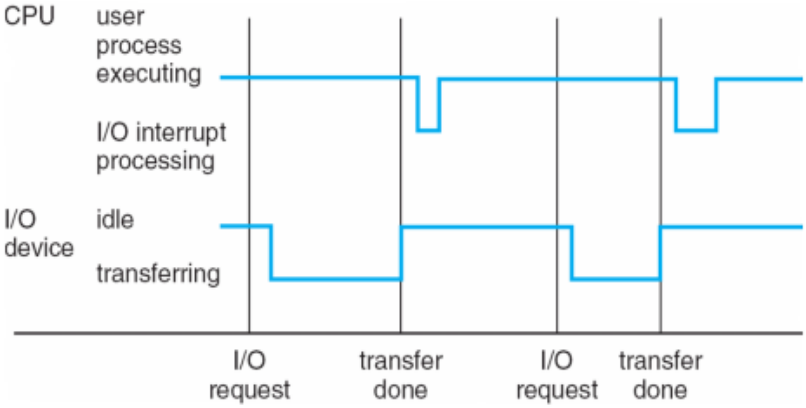


Fig: Interrupt time line for a single process doing output

- **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.

❖ The **load** instruction moves a word from main memory to an internal register within the CPU, whereas the **store** instruction moves the content of a register to main memory. Aside from explicit loads and stores, the CPU automatically loads instructions from main memory for execution.

❖ A typical instruction-execution cycle, as executed on a system with a **von Neumann** architecture, first fetches an instruction from memory and stores that instruction in the **instruction register.**

❖ The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory.

❖ 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** (*Nonvolatile)* as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently.

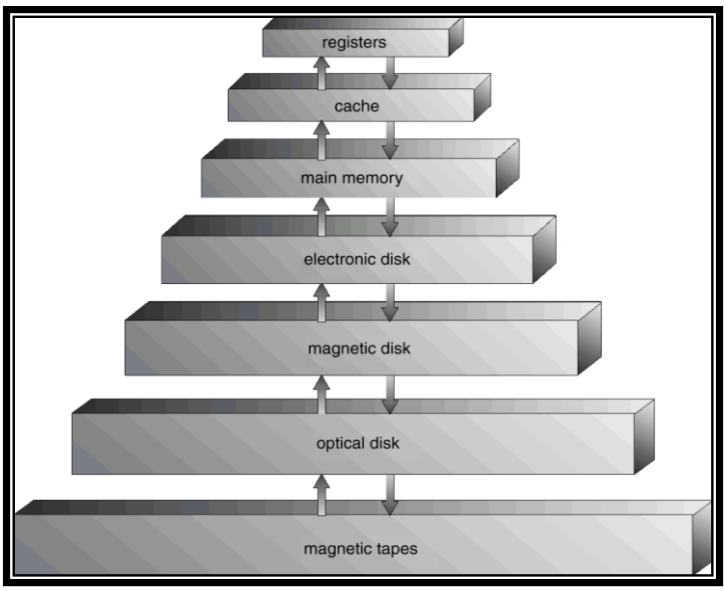
The most common secondary-storage device is a **magnetic disk,** which provides storage for both programs and data. Most programs (web browsers, compilers, word processors, spreadsheets, and so on) are stored on a disk until they are loaded into memory.

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 (fig given below) according to

- Speed
- Cost
- Volatility

Storage-Device Hierarchy



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.

Now many early storage devices, including paper tape and core memories, are relegated to museums now that magnetic tape and **semiconductor memory** have become faster and cheaper. The top four levels of memory in Figure may be constructed using semiconductor memory.

In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile. As mentioned earlier, **volatile storage** loses its contents when the power to the device is removed. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage** for safekeeping.

In the hierarchy shown in Figure, the storage systems above the electronic disk are *volatile,* whereas those below **disk** can be designed to be either *volatile or nonvolatile.*

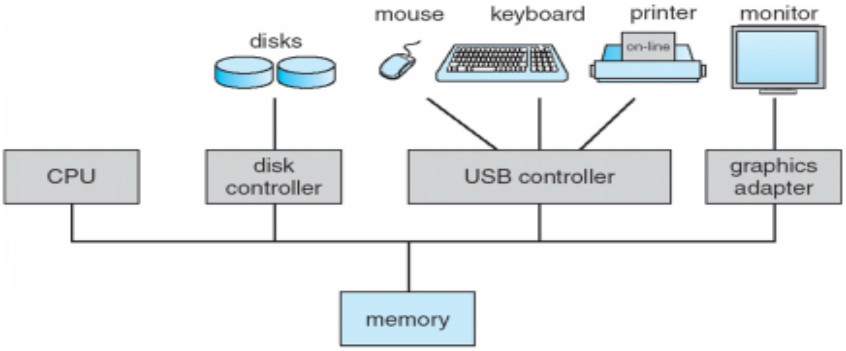
➢ I/O Structure:-

Storage is only one of many types of I/O devices within a computer. A large portion of operating system code is dedicated to managing I/O.

A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. Depending on the controller, there may be more than one attached device.

A device controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.

Typically, operating systems have a **device driver** for each device controller. This device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system.



A modern computer system.

To start an I/O operation, the device driver loads the appropriate registers within the device controller.

- The device controller, in turn, examines the contents of these registers to determine what action to take (such as "read a character from the keyboard").
- The controller starts the transfer of data from the device to its local buffer.
- Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
- The device driver then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read.
- For other operations, the device driver returns status information.

This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O.

To solve this problem, **direct memory access (DMA)** is used. After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. While the device controller is performing these operations, the CPU is available to accomplish other work.

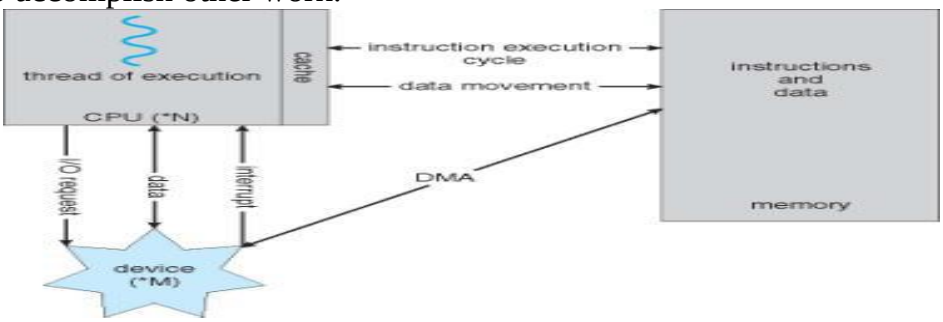


Fig: How a modern computer system works

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. Single-Processor System
2. Multiprocessor System
3. Clustered System

1. Single-Processor System:-

Most systems vise a single processor.

- 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;
- 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.
- All of these special-purpose processors run a limited instruction set and do not run user processes.
- Sometimes they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status.
- For example, a disk-controller microprocessor receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm.
- This arrangement relieves the main CPU of the overhead of disk scheduling.
- PCs contain a microprocessor in the keyboard to convert the keystrokes into codes to be sent to the CPU. In other systems or circumstances, special-purpose processors are low-level components built into the hardware.
- The operating system cannot communicate with these processors; they do their jobs autonomously.
- 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.

2. Multiprocessor System:-

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. Throughput is increased 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. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly.

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. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

The multiple-processor systems in use today are of two types.

i. **Asymmetric multiprocessing**
ii. **Symmetric multiprocessing (SMP)**

Asymmetric multiprocessing:-

- In **asymmetric multiprocessing** 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.
- Each processor has its own memory address space.

Symmetric multiprocessing (SMP)

- Each processor runs an identical copy of OS. These copies communicate with each other.
- Each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors.
- The benefit of this model is that many processes can run simultaneously—*N* processes can run if there are *N* CPUs—without causing a significant deterioration of performance.

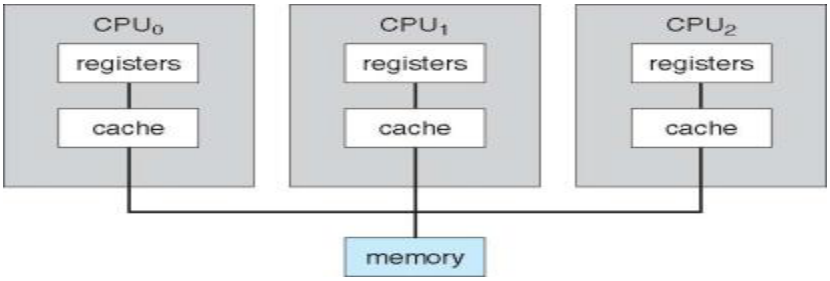


Fig: Symmetric multiprocessing architecture

A multiprocessor system will allow processes and resources—such as memory—to be shared dynamically among the various processors.

The difference between symmetric and asymmetric multiprocessing may result from either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master and multiple slaves

3. Clustered System;-

Another type of multiple-CPU system is the **clustered system.** Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.

Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems coupled together

The generally accepted definition is that clustered computers share storage and are closely linked via a **local-area network (LAN)** or a faster interconnect such as InfiniBand.

Clustering is usually used to provide **high-availability** service; that is, service will continue even if one or more systems in the cluster fail. High availability is generally obtained by adding a level of redundancy in the system. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the LAN). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. The users and clients of the applications see only a brief interruption of service.

❖ Clustering can be structured asymmetrically or symmetrically.

In **asymmetric clustering,** one machine is in **hot-standby mode** while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.

In **symmetric mode,** two or more hosts are running applications, and are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware. It does require that more than one application be available to run.

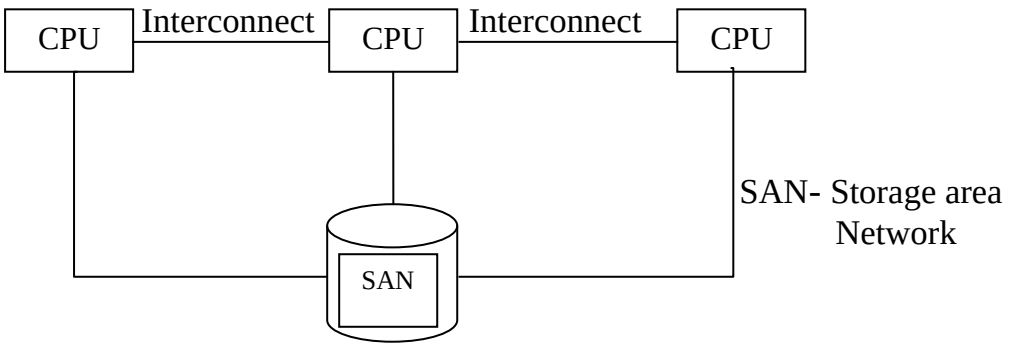


Fig: General Structure of a clustered system

4. Multiprogramming System

An operating system provides the environment within which multiple programs are executed.

- One of the most important aspects of operating systems is the ability to multiprogram. 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 shown in Figure 1.7. This set of jobs can be a subset of the jobs kept in the job pool—which contains all jobs that enter the system—since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool.

The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. 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. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

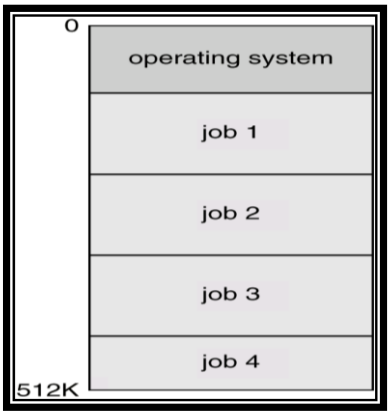


Fig: Memory layout for a multiprogramming system

Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

4. Time sharing (or multitasking)

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** (or **hands-on) 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 a 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.

A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program loaded into memory and executing is called a **process.**

Time-sharing and multiprogramming require several jobs to be kept simultaneously in memory. Since in general main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the **job pool.** This pool consists of all processes residing on disk awaiting allocation of main memory.

If several jobs are ready to be brought into memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is **job scheduling.** When the operating system selects a job from the job pool, it loads that job into memory for execution.

In addition, if several jobs are ready to run at the same time, the system must choose among them. Making this decision is **CPU scheduling.**

1.2 Operating-System Operations (functions)

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. The interrupt-driven nature of an operating system defines that system's general structure. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

Since the operating system and the users share the hardware and software resources of the computer system, we need to make sure that an error in a user program could cause problems only for the one program that was running. With sharing, many processes could be adversely affected by a bug in one program. For example, if a process gets stuck in an infinite loop, this loop could prevent the correct operation of many other processes. 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.

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. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request. This is shown in Figure 1.8. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.

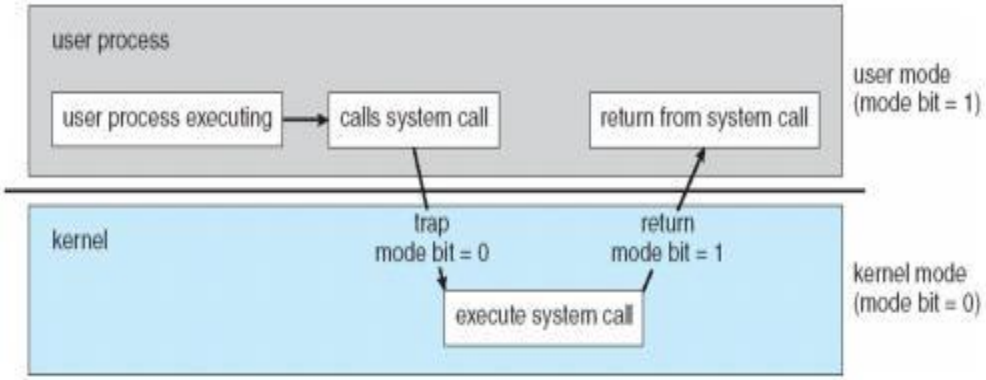


Fig: Transition from user to kernel mode

❖ 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 system always switches to user mode (by setting the mode bit to 1) before passing control to a user program. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

❖ System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf. A system call is invoked in a variety of ways, depending on the functionality provided by the underlying processor. In all forms, it is the method used by a process to request action by the operating system.

❖ When a system call is executed, it is treated by the hardware as software interrupt. Control passes through the interrupt vector to a service routine in the operating system, and the mode bit is set to kernel mode. The system call service

routine is a part of the operating system. The kernel examines the interrupting instruction to determine what system call has occurred; a parameter indicates what type of service the user program is requesting.

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. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

❖ we can use the timer to prevent a user program from running too long. A simple technique is to initialize a counter with the amount of time that a program is allowed to run. A program with a 7-minute time limit, for example, would have its counter initialized to 420. Every second, the timer interrupts and the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

3. Process Management

❖ A program does nothing unless its instructions are executed by a CPU. A program in execution, as mentioned, is a process. A time-shared user program such as a compiler is a process. A word-processing program being run by an individual user on a PC is a process. A system task, such as sending output to a printer, can also be a process (or at least part of one). For now, you can consider a process to be a job or a time-shared program.

❖ A process needs certain resources—including CPU time, memory, files, and I/O devices—to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running.

❖ We emphasize that a program by itself is not a process; a program is a *passive* entity, such as the contents of a file stored on disk, whereas a process is an *active* entity. A single-threaded process has one **program counter** specifying the next instruction to execute. The execution of such a process must be sequential. The CPU executes one instruction of the process after another, until the process completes. Further, at any time, one instruction at most is executed on behalf of the process. Thus, although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread.

❖ A process is the unit of work in a system. Such a system consists of a collection of processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes (those that execute user code). All these processes can potentially execute concurrently— by multiplexing on a single CPU.

❖ The operating system is responsible for the following activities in connection with process management:

• Creating and deleting both user and system processes
• Suspending and resuming processes
• Providing mechanisms for process synchronization
• Providing mechanisms for process communication
• Providing mechanisms for deadlock handling

4. Memory Management

❖ The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The central processor reads instructions from main memory during the instruction-fetch cycle and both reads and writes data from main memory during the data-fetch cycle. The main memory is generally the only large storage device that the CPU is able to address and access directly.

❖ For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls. In the same way, instructions must be in memory for the CPU to execute them.

❖ For a program to be executed, it must be mapped to absolute addresses and loaded into memory. As the program executes, it accesses program instructions and data from memory by generating these absolute addresses. Eventually, the program terminates, its memory space is declared available, and the next program can be loaded and executed.

❖ To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management. Many different memory management schemes are used. These schemes reflect various approaches, and the effectiveness of any given algorithm depends on the situation. In selecting a memory-management scheme for a specific system, we must take into account many factors—especially on the *hardware* design of the system. Each algorithm requires its own hardware support.

❖ The operating system is responsible for the following activities in connection with memory management:

• Keeping track of which parts of memory are currently being used and by whom
• Deciding which processes and data to move into and out of memory
• Allocating and deallocating memory space as needed

4.1 Mass(Secondary)-Storage Management

❖ Main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the principal on-line storage medium for both programs and data. Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing.

❖ The operating system is responsible for the following activities in connection with disk management:

• Free-space management
• Storage allocation
• Disk scheduling

❖ Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may hinge on the speeds of the disk subsystem and of the algorithms that manipulate that subsystem.

4.2 Caching

❖ **Caching** is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system—the cache—on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache; if it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

❖ In addition, internal programmable registers, such as index registers, provide a high-speed cache for main memory. The programmer (or compiler) implements the register-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory. There are also caches that are implemented totally in hardware.

❖ Main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use, and data must be in main memory before being moved to secondary storage for safekeeping. The file-system data, which resides permanently on secondary storage, may appear on several levels in the storage hierarchy. At the highest level, the operating system may maintain a cache of file-system data in main memory Also; electronic RAM disks may be used for high-speed storage that is accessed through the file-system interface.

6. File-System Management

❖ File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common. Each of these

media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).

❖ A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).

❖. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways(for example :read, write, append) files may be accessed.

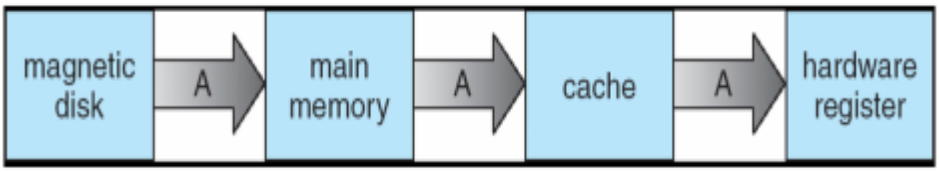
❖ The operating system is responsible for the following activities in connection with file management:

• Creating and deleting files
• Creating and deleting directories to organize files
• Supporting primitives for manipulating files and directories
• Mapping files onto secondary storage
• Backing up files on stable (nonvolatile) storage media

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Performance of various levels of storage.

The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software. For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention. In contrast, transfer of data from disk to memory is usually controlled by the operating system.'



Migration of integer A from disk to register.

I/O Systems

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O **subsystem.**

The I/O subsystem consists of several components:
• A memory-management component that includes buffering, caching, and spooling
• A general device-driver interface
• Drivers for specific hardware devices
Only the device driver knows the peculiarities of the specific device to which it is assigned.

1.3. Protection and Security

If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated. For that purpose, mechanisms ensure that files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system. For example, memory-addressing hardware ensures that a process can execute only within its own address space. The timer ensures that no process can gain control of the CPU without eventually relinquishing control. Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.

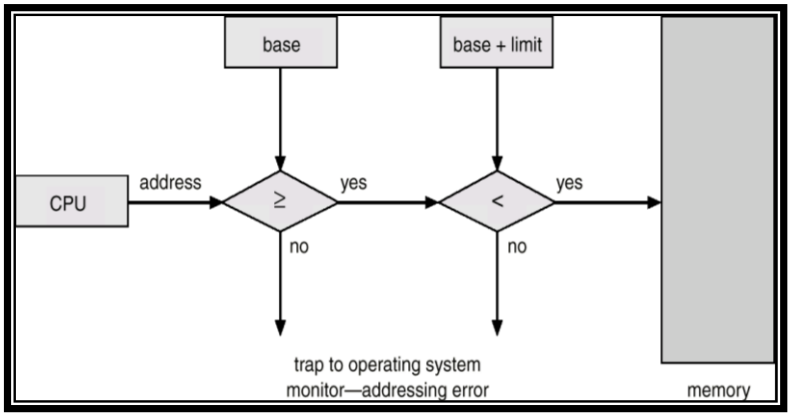
Protection, then, is any mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement. It is the job of **security** to a system from external and internal attacks.

I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

Hardware Address Protection

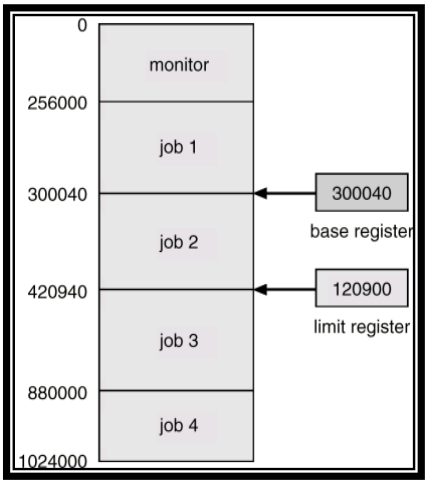
- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
- The load instructions for the *base* and *limit* registers are privileged instructions



Hardware Address Protection

Memory Protection

- Os must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - **Base register** – holds the smallest legal physical memory address.
  - **Limit register** – contains the size of the range
- Memory outside the defined range is protected



Use of a Base and Limit Register

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.

1.4 Distributed Systems

➢ A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide the users with access to the various resources that the system maintains. Access to a shared resource increases computation speed, functionality, data availability, and reliability.

➢ A **network,** in the simplest terms, is a communication path between two or more systems. Distributed systems depend on networking for their functionality. Networks vary by the protocols used, the distances between nodes, and the transport media. TCP/IP is the most common network protocol, although ATM and other protocols are in widespread use.

➢ Networks are characterized based on the distances between their nodes. A **local-area network (LAN)** connects computers within a room, a floor, or a building. A **wide-area network (WAN)** usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide. These networks may run one protocol or several protocols. The continuing advent of new technologies brings about new forms of networks. For example, a **metropolitan-area network** (MAN) could link buildings within a city. Bluetooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a **small-area network** such as might be found in a home.

➢ The media to carry networks are equally varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes, and radios. When computing devices are connected to cellular phones, they create a network. Even very short-range infrared communication can be used for networking.

➢ Some operating systems have taken the concept of networks and distributed systems further than the notion of providing network connectivity. A **network operating system** is an operating system that provides features such as file sharing across the network and that includes a communication scheme that allows different processes on different computers to exchange messages. A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers. A distributed operating system provides a less autonomous environment: The different operating systems communicate closely enough to provide the illusion that only a single operating system controls the network.

1.5 Special-Purpose Systems

Real-Time Embedded Systems

➢ Embedded computers are the most prevalent form of computers in existence. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks. The systems they

run on are usually primitive, and so the operating systems provide limited features. Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

➢ Embedded systems almost always run **real-time operating systems.** A real-time system has well-defined, fixed time constraints. Processing *mustbe* done within the defined constraints, or the system will fail, these type of systems are called hard ***real time systems.*** A real-time system functions correctly only if it returns the correct result within its time constraints.

➢ Contrast this system with a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or a batch system, which may have no time constraints at all, these type of systems are called ***soft real time systems.***

Multimedia Systems

➢ Most operating systems are designed to handle conventional data such as text files, programs, word-processing documents, and spreadsheets. However, a recent trend in technology is the incorporation of **multimedia data** into computer systems. Multimedia data consist of audio and video files as well as conventional files. These data differ from conventional data in that multimedia data—such as frames of video—must be delivered (streamed) according to certain time restrictions (for example, 30 frames per second).

➢ Multimedia describes a wide range of applications that are in popular use today. These include audio files such as MP3 DVD movies, video conferencing, and short video clips of movie previews or news stories downloaded over the Internet. Multimedia applications may also include live webcasts (broadcasting over the World Wide Web) of speeches or sporting events and even live webcams that allow a viewer in Manhattan to observe customers at a café in Paris. Multimedia applications need not be either audio or video; rather, a multimedia application often includes a combination of both. For example, a movie may consist of separate audio and video tracks. Nor must multimedia applications be delivered only to desktop personal computers. Increasingly, they are being directed toward smaller devices, including PDAs and cellular telephones.

Handheld Systems

➢ **Handheld systems** include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones, many of which use special-purpose embedded operating systems.

➢ Developers of handheld systems and applications face many challenges, those are

➢ Most of which are due to the limited size of such devices. For example, a PDA is typically about 5 inches in height and 3 inches in width, and it weighs less than one-half pound. Because of their size, most handheld devices have a small amount of memory, slow processors, and small display screens. We will take a look now at each of these limitations.

➢ The first issue is the amount of physical memory in a handheld depends upon the device, but typically is is somewhere between 512 KB and 128 MB. As a result, the operating system and applications must manage memory efficiently..

➢ A second issue of concern to developers of handheld devices is the speed of the processor used in the devices. Processors for most handheld devices run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery, which would take up more space and would have to be replaced (or recharged) more frequently. Most handheld devices use smaller, slower processors that consume less power.

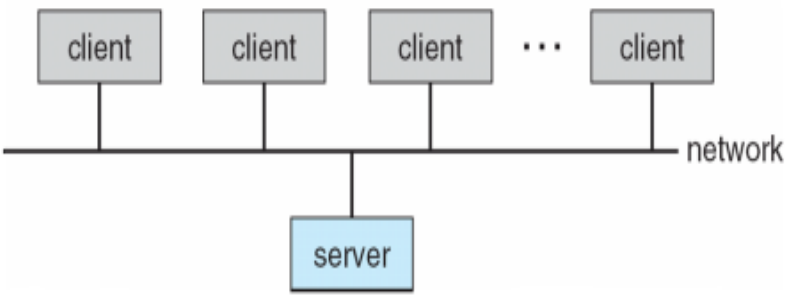
➢ The last issue confronting program designers for display size of monitor. A lack of physical space limits input methods to small keyboards, handwriting recognition, or small screen-based keyboards. The small display screens limit output options. Whereas a monitor for a home computer may measure up to 30 inches, the display for a handheld device is often no more than 3 inches square. Familiar tasks, such as reading e-mail and browsing web pages, must be condensed into smaller displays. One approach for displaying the content in web pages is **web clipping,** where only a small subset of a web page is delivered and displayed on the handheld device.

➢ Some handheld devices use wireless technology, such as BlueTooth or 802.11, allowing remote access to e-mail and web browsing. Cellular telephones with connectivity to the Internet fall into this category.

Computing Environments

Client-Server Computing

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.



General structure of a client-server system

- 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.

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.
- In a client-server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.
- To participate in a peer-to-peer system, a node must first join the network of peers.
- Once a node has joined the network, it can begin providing services to - and requesting services from—other nodes in the network.

1.6 Operating-System Structure

A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

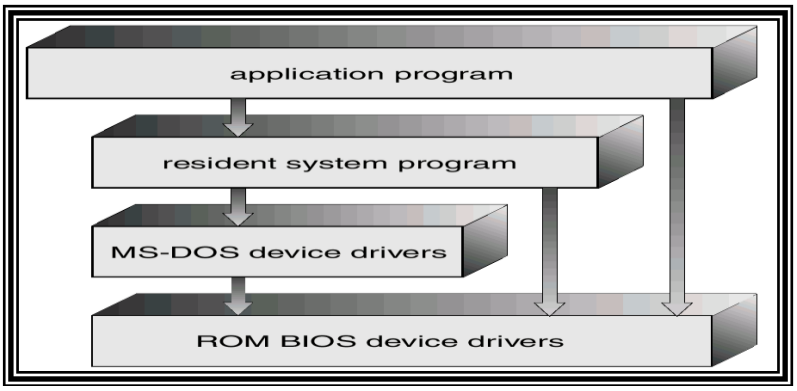
Simple Structure

Many commercial systems do not have well-defined structures. Frequently, such operating systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system.

- MS-DOS – written to provide the most functionality in the least space
  - not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail. Because the Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

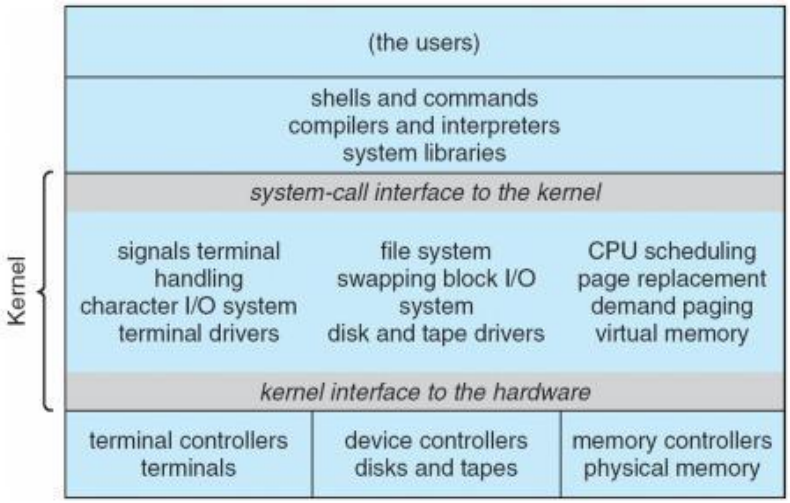
MS-DOS System Structure



Another example of limited structuring is the original UNIX operating system.

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts.
  - Systems programs
  - The kernel - kernel is further separated into a series of interfaces and device drivers.
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

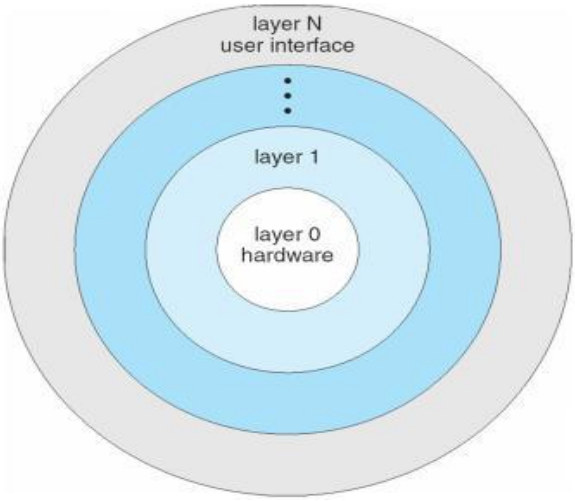
UNIX System Structure



Layered Approach

- A operating system can be made modular in many ways. One method is the **layered approach,** in which the operating system is broken up into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS or UNIX systems. The operating system can then retain much greater control over the computer and over the applications that make use of that computer. Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems. Under the top down approach, the overall functionality and features are determined and are separated into components. Information hiding is also important, because it leaves programmers free to implement the low-level routines as they see fit, provided that the external interface of the routine stays unchanged and that the routine itself performs the advertised task.



A layered operating system

An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data. A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers. Layer *M*, in turn, can invoke operations on lower-level layers.

The main advantage of the layered approach is simplicity of construction and debugging. This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system,. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system is simplified.

Each layer is implemented with only those operations provided by lower level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

Microkernels

- Moves as much from the kernel into ―*user*‖ space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

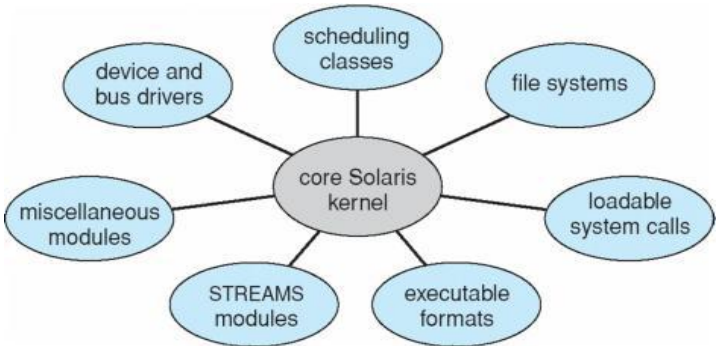
Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Here, the kernel has a set of core components and dynamically links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X.

For example, the Solaris operating system structure, shown in below Fig., is organized around a core kernel with seven types of loadable kernel modules:

1. Scheduling classes
2. File systems
3. Loadable system calls
4. Executable formats
5. STREAMS modules
6. Miscellaneous
7. Device and bus drivers

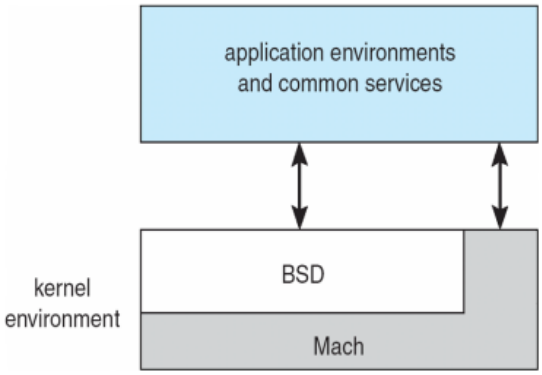


Solaris Modular Approach

Such a design allows the kernel to provide core services yet also allows certain features to be implemented dynamically. For example, device and bus drivers for specific hardware can be added to the kernel, and support for different file systems can be added as loadable modules.

The Apple Macintosh Mac OS X operating system uses a hybrid structure. Mac OS X (also known as *Danvin)* structures the operating system using a layered technique where one layer consists of the Mach microkernel. The structure of Mac OS X appears in below Figure.

Mac OS X Structure



The top layers include application environments and a set of services providing a graphical interface to applications. Below these layers is the kernel environment, which consists primarily of the Mach microkernel and the BSD kernel.

Mach provides memory management; support for remote procedure calls (RPCs) and interprocess communication (IPC) facilities, including message passing; and thread scheduling.

The BSD component provides a BSD command line interface, support for networking and file systems, and an implementation of POSIX APIs, including Pthreads.

In addition to Mach and BSD, the kernel environment provides an I/O kit for development of device drivers and dynamically loadable modules (which Mac OS X refers to as **kernel extensions).**

1.7 System Calls

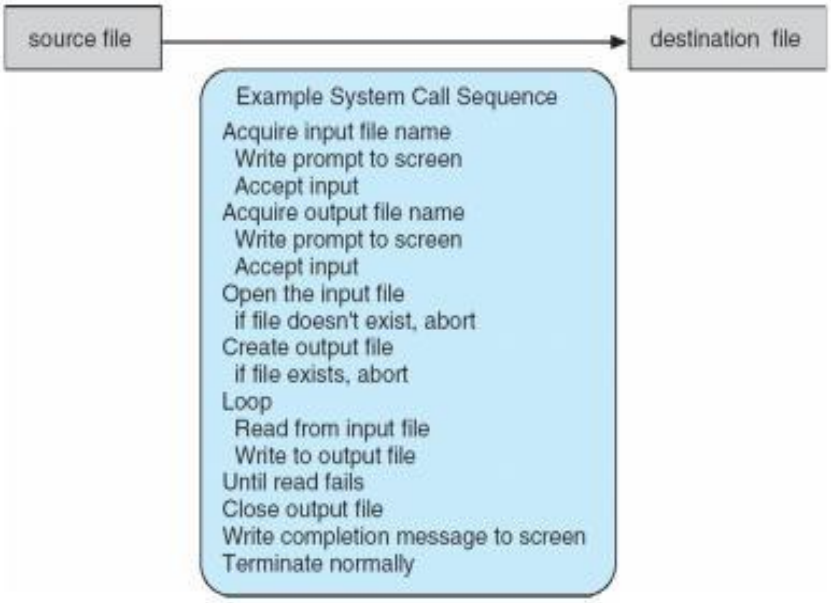
- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example

Let's first use an example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file. The first input that the program will need is the names of the two files: the input file and the output file. These names can be specified in many ways, depending on the operating-system design. One approach is for the program to ask the user for the names of the two files.

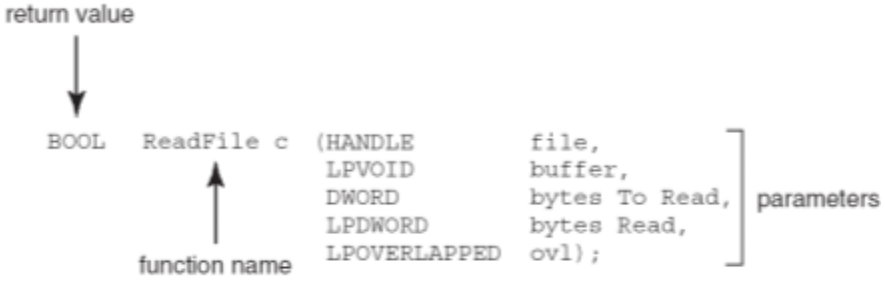
Example of System Calls

- System call sequence to copy the contents of one file to another file.



Example of Standard API

As an example of a standard API, consider the ReadFile( ) function in the Win32 API—a function for reading from a file. The API for this function appears in the below Figure.



A description of the parameters passed to ReadFile ( )

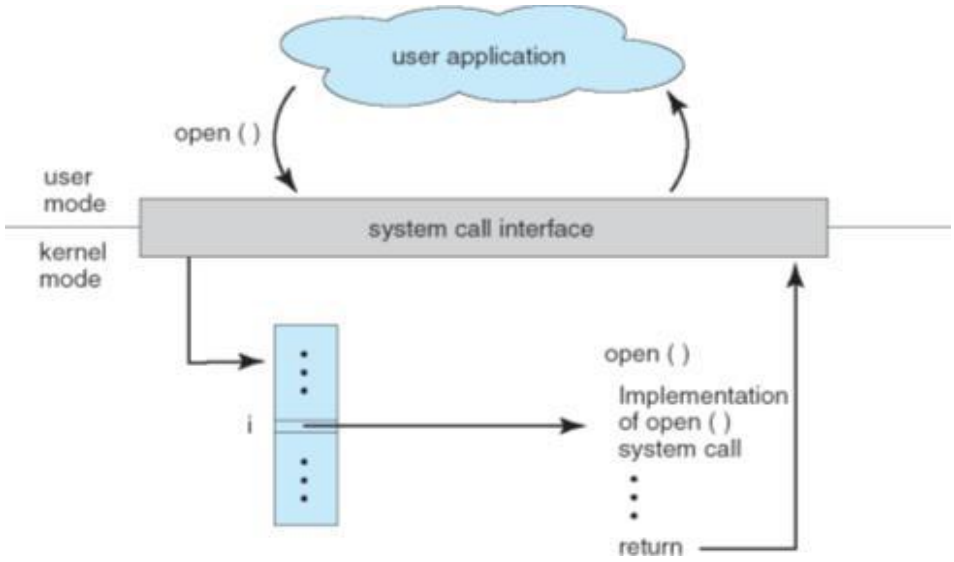
- HANDLE file—the file to be read
- LPVOID buffer—a buffer where the data will be read into and written from
- DWORD bytesToRead—the number of bytes to be read into the buffer
- LPDWORD bytesRead—the number of bytes read during the last read
- LPOVERLAPPED ovl—indicates if overlapped I/O is being used

The run-time support system (a set of functions built into libraries included with a compiler) for most programming languages provides a **system-call interface** that serves as the link to system calls made available by the operating system. The system-call interface intercepts function calls in the API and invokes the necessary system call within the operating system.

- Typically, a number associated with each system call
- System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API –System Call –OS Relationship

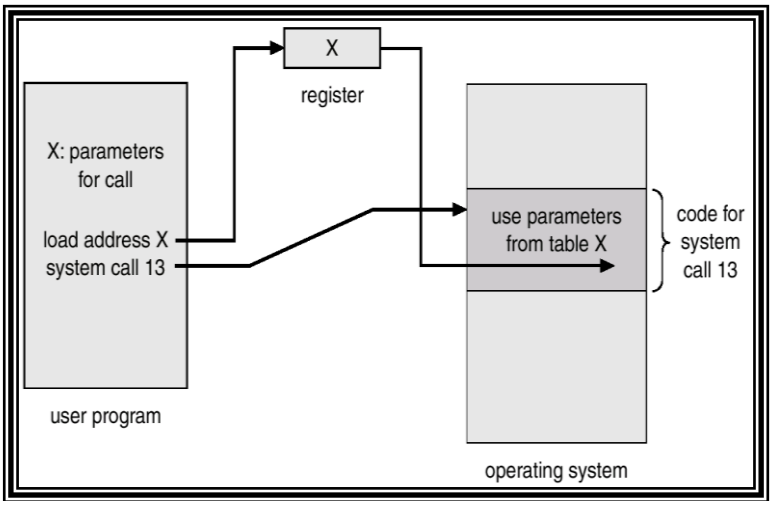
The handling of a user application invoking the open( ) system call.



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.(This approach taken by Linux and Solaris)
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

Parameter Passing via Table



Types of System Calls

System calls can be grouped roughly into five major categories:

- Process control
- File management
- Device management
- Information maintenance
- Communications

Process Control

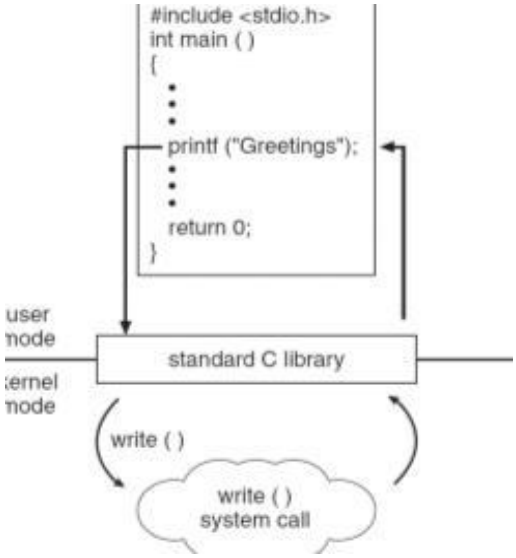
A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.

Types of system calls of Process control

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command.

Example Of Standard C Library



The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, lot's assume a C program invokes the printf () statement. The C library intercepts this call and invokes the necessary system call(s) in the operating system—in this instance, the write () system call. The C library takes the value returned by write () and passes it back to the user program. This is shown in above Figure.

File Management

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it.

System calla in File management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

File attributes include the file name, a file type, protection codes, accounting information, and so on. At least two system calls, get file attribute and set file attribute, are required for this function.

Device Management

A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files). If there are multiple users of the system, the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and close system calls for files.

Some System calls are:

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Once the device has been requested (and allocated to us), we can read, write, and (possibly) reposition the device, just as we can with files.

Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date . Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

In addition, the operating system keeps information about all its processes, and system calls are used to access this information.

Some System calls are

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

Communication

There are two common models of interprocess communication: the message passing model and the shared-memory model.

In the message-passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened.

In the shared-memory model, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.

Some System calls are

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

System Programs

Another aspect of a modern system is the collection of system programs. Recall Figure 1.1, which depicted the logical computer hierarchy. At the lowest level is hardware. Next is the operating system, then the system programs, and finally the application programs. System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex. They can be divided into these categories:

- **File management.** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information.** Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a registry, which is used to store and retrieve configuration information.
- **File modification.** Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
- **Programming-language support.** Compilers, assemblers, debuggers and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.
- **Program loading and execution.** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
- **Communications.** These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such programs include web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as **system utilities** or **application programs.**

1.8 Operating-System Generation

It is possible to design, code, and implement an operating system specifically for one machine at one site. More commonly, however, operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations. The system must then be configured or generated for each specific computer site, a process sometimes known as **system generation** (SYSGEN).

The operating system is normally distributed on disk or CD-ROM. To generate a system, we use a special program. The SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there. The following kinds of information must be determined.

- What CPU is to be used? What options (extended instruction sets, floating point arithmetic, and so on) are installed? For multiple CPU systems, each CPU must be described.

- How much memory is available? Some systems will determine this value themselves by referencing memory location after memory location until an "illegal address" fault is generated. This procedure defines the final legal address and hence the amount of available memory.

- What devices are available? The system will need to know how to address each device (the device number), the device interrupt number, the device's type and model, and any special device characteristics.

- What operating-system options are desired, or what parameter values are to be used? These options or values might include how many buffers of which sizes should be used, what type of CPU-scheduling algorithm is desired, what the maximum number of processes to be supported is, and so on.

Once this information is determined, it can be used in several ways. At one extreme, a system administrator can use it to modify a copy of the source code of the operating system.

System Boot

After an operating system is generated, it must be made available for use by the hardware. But how does the hardware know where the kernel is or how to load that kernel? The procedure of starting a computer by loading the kernel is known as *booting* the system. On most computer systems, a small piece of code known as the **bootstrap program** or **bootstrap loader** locates the kernel, loads it into main memory, and starts its execution.

When a CPU receives a reset event—for instance, when it is powered up or rebooted—the instruction register is loaded with a predefined memory location, and execution starts there. At that location is the initial bootstrap program. This program is in the form of **read-only memory (ROM),** because the RAM is in an unknown state at system startup.

The bootstrap program can perform a variety of tasks. Usually, one task is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps. It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory.

UNIT –I: Revised Questions

NOV-2012

1. (a) Distinguish between the client-server and peer-to-peer models of distributed systems.
(b) Explain briefly system calls and their types.

1. (a) Define the essential properties of the following types of operating system:
(i) Batch
(ii) Real Time
(iii) Time sharing
(iv) Clustered
(b) In a multiprogramming and time sharing environment several users share the system simultaneously. This situation can result in various security problems. Discuss these problems.

1. (a) What is an operating system? Explain briefly the operating systems role in the overall computer system.
(b) Under what circumstances would a user be better off using a time sharing system rather than a PC or single-user workstation?

1. (a) How are network computers different from traditional personal computers? Describe some usage scenarios in which it is advantageous to use network computers.
(b) Explain briefly about the services and functions provided by the operating system.

MAY-2012

1. List and explain the different CPU scheduling algorithms with example. [16]

1. a) Discuss about the operating system components.
b) What is semaphore? What is meant by counting semaphore and binary semaphore? Discuss mutual exclusion implementation using semaphore. [8+8]

1. a) Explain the terms multiprogramming and multitasking.
b) Explain the different file access control methods. [8+8]

1. a) Explain the layered approach and microkernel approach to system design.
b) Explain the linked method of allocating disk space with its advantages and disadvantages. [8+8]

MAY-2011

1. (a) Define an operating system. What is its purpose? Explain the various functions of an operating system.
(b) What is meant by system call? Discuss about types of system calls. [8+8]

1. (a) Discuss about the operating system components.
(b) What is microkernel? Discuss the microkernel approach to system design. [8+8]

1. (a) Explain the terms multiprogramming and multitasking. [8+8]
(b) What is a system call? Explain in brief the types of system calls provided by OS.

1. (a) List and explain the different operating system services.
(b) What is the dual mode of operation for protecting an operating system? How the hardware support is provided for the dual mode of operation? [8+8]

MAY-2010

1. (a) With the help of a diagram, explain the abstract view of the components of a computer system.
(b) What are the memory devices that can be connected to a computer? Organize them in a hierarchy. [8+8]

1. (a) Discuss operating system from user point of view and system view
(b) Compare the computer systems based on the number of processors. [8+8]

1. (a) What is the difference between symmetric and asymmetric multiprocessing.
(b) How multiprogramming improves CPU utilization. [8+8]

1. (a) What is Dual mode operation of operating system?
(b) How Caching improves performance of a system. [8+8]