Module I

INTRODUCTION TO OPERATING SYSTEM

What is an Operating System?

An operating system is a system software that acts as an intermediary between a user of a computer and the computer hardware.
It is a software that manages the computer hardware.
Os allows the user to execute programs in a convenient and efficient manner.
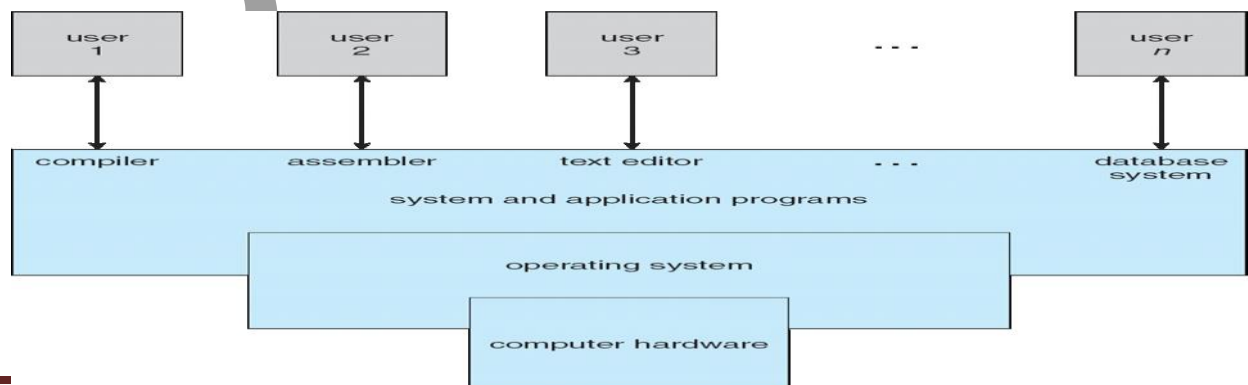
Operating system goals:
- Make the computer system convenient to use. It hides the difficulty in managing the hardware.
- Use the computer hardware in an efficient manner
- Provide and environment in which user can easily interface with computer.
- It is a **resource allocator**

Computer System Structure (Components of Computer System)

Computer system mainly consists of four components-

- Hardware – provides basic computing resources
  - ✓ CPU, memory, I/O devices
- Operating system
  - ✓ Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - ✓ Word processors, compilers, web browsers, database systems, video games
- Users
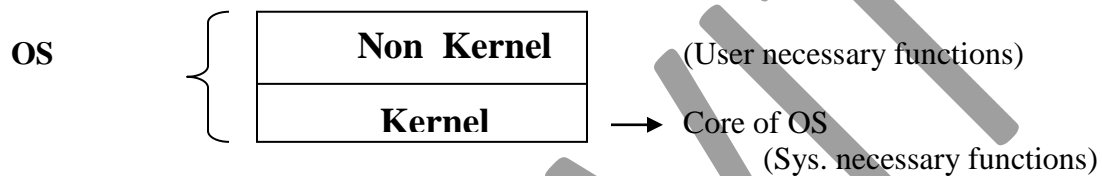 - ✓ People, machines, other computers

The basic hardware components comprises of CPU, memory, I/O devices. The application program uses these components. The OS controls and co-ordinates the use of hardware, among various application programs (like compiler, word processor etc.) for various users.

The OS allocates the resources among the programs such that the hardware is efficiently used.

The operating system is the program running at all the times on the computer. It is usually called as the kernel.

OS { | **Non Kernel** | (User necessary functions) |
| --- |
| **Kernel** | → Core of OS |

(Sys. necessary functions)

Kernel functions are used always in system, so always stored in memory. Non kernel functions are stored in hard disk, and it is retrieved whenever required.

Views of OS

Operating System can be viewed from two viewpoints–
   User views & System views

1. User Views:-
   The user's view of the operating system depends on the type of user.
   i. If the user is using **standalone** system, then OS is designed for ease of use and high performances. Here resource utilization is not given importance.

   ii. If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to maximize resource utilization. OS is designed such that the CPU time, memory and i/o are used efficiently and no single user takes more than the resource allotted to them.

   iii. If the users are in **workstations**, connected to networks and servers, then the user have a system unit of their own and shares resources and files with other systems. Here the OS is designed for both ease of use and resource availability (files).

   iv. Users of **hand held** systems, expects the OS to be designed for ease of use and performance per amount of battery life.

v. Other systems like embedded systems used in home devies (like washing m/c) & automobiles do not have any user interaction. There are some LEDs to show the status of its work.

2. System Views:-
   Operating system can be viewed as a **resource allocator** and **control program**.
   i. **Resource allocator** - The OS acts as a manager of hardware and software resources. CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program. There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority.
   ii. **Control Program** – The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer.

Computer System Organization

Computer-system operation
One or more CPUs, device controllers connect through common bus providing access to shared memory. Each device controller is in-charge of a specific type of device. To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory. The CPU and other devices execute concurrently competing for memory cycles. Concurrent execution of CPUs and devices competing for memory cycles
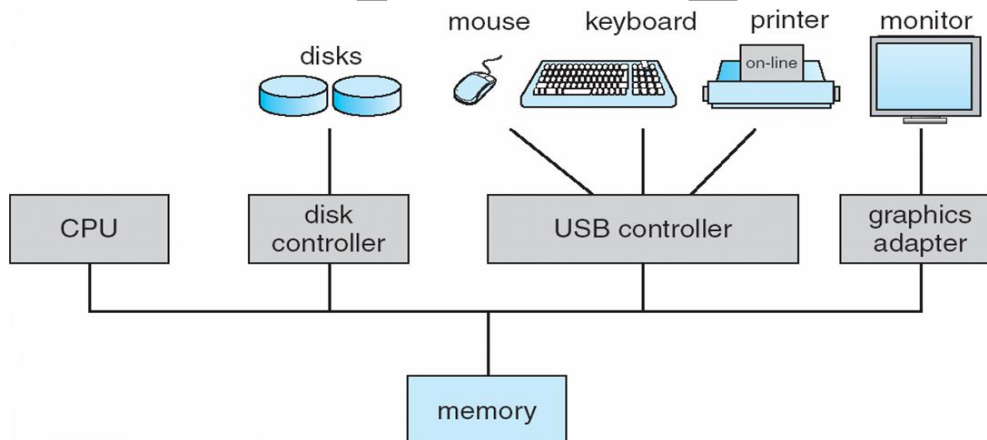
When system is switched on, '**Bootstrap'** program is executed. It is the initial program to run in the system. This program is stored in read-only memory (ROM) or in electrically erasable programmable read-only memory(EEPROM). It initializes the CPU registers, memory, device controllers and other initial setups. The program also locates and loads, the OS kernel to the

memory. Then the OS starts with the first process to be executed (ie. 'init' process) and then wait for the interrupt from the user.

Switch on ⟶ 'Bootstrap' program
  - Initializes the registers, memory and I/O devices
  - Locates & loads kernel into memory
  - Starts with 'init' process
  - Waits for interrupt from user.

Interrupt handling –

The occurrence of an event is usually signaled by an **interrupt.** The interrupt can either be from the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU. Software triggers 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 (Interrupt Vector Table) contains the starting address where the service routine for the interrupt is located. After the execution of interrupt service routine, 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
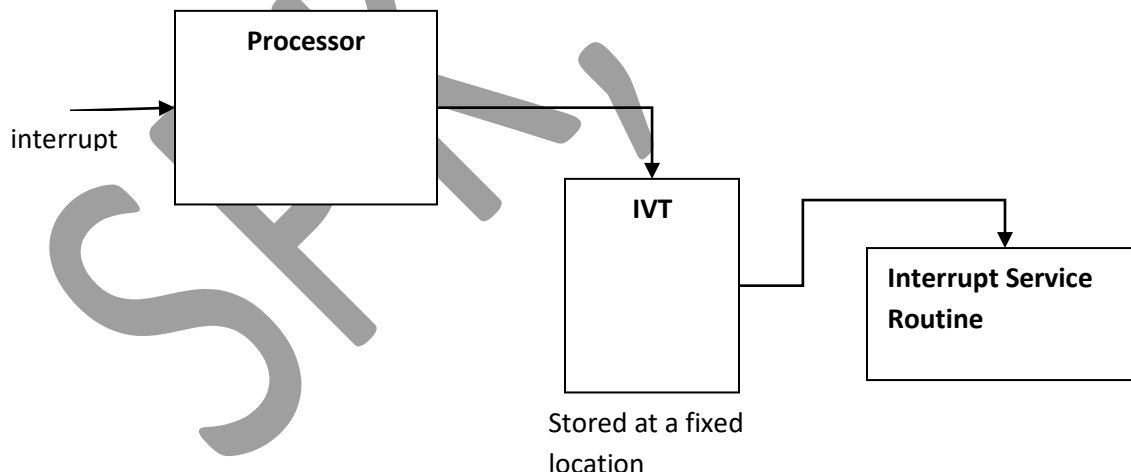
Storage Structure

Computer programs must be in main memory (**RAM)** to be executed. Main memory is the large memory that the processor can access directly. It commonly is implemented in a semiconductor technology called **dynamic random-access memory (DRAM).** Computers provide Read Only Memory(ROM), whose data cannot be changed.

All forms of memory provide 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.

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.

Thus, most computer systems provide **secondary storage** as an extension of main memory. The main requirement for secondary storage is that it will 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 are stored on a disk until they are loaded into memory. Many programs then use the disk as both a source and a destination of the information for their processing.
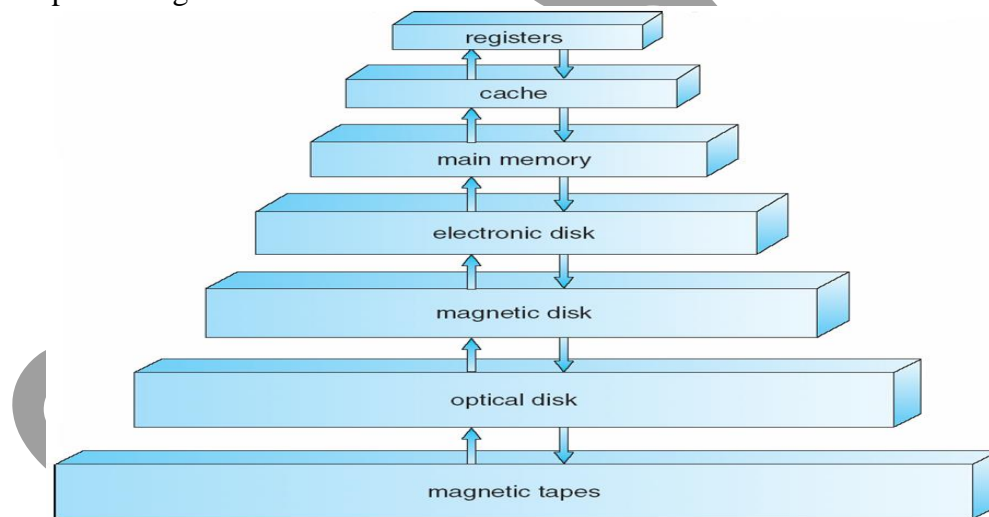
The wide variety of storage systems in a computer system can be organized in a hierarchy as shown in the figure, according to speed, cost and capacity. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time and the capacity of storage generally increases.

In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile. **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 are nonvolatile.

An **electronic disk** can be designed to be either volatile or nonvolatile. During normal operation, the electronic disk stores data in a large DRAM array, which is volatile. But many electronic-disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic-disk controller copies the data from RAM to the magnetic disk. Another form of electronic disk is flash memory.

I/O Structure

A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

Every device have a device controller, maintains some local buffer and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices. The operating systems have a **device driver** for each device controller.

DEVICE
CONTROLLER

BUFFER   Registers
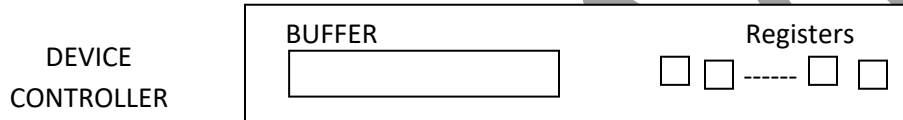
To start an I/O operation, the device driver loads the registers within the device controller. The device controller, examines the contents of these registers to determine what action to take (such as "read a character from the keyboard"). The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver(OS) via an interrupt that it has finished its operation. The device driver then returns control to the operating system, and also returns the data. 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 very difficult for bulk data movement. To solve this problem, **direct memory access (DMA)** is used.
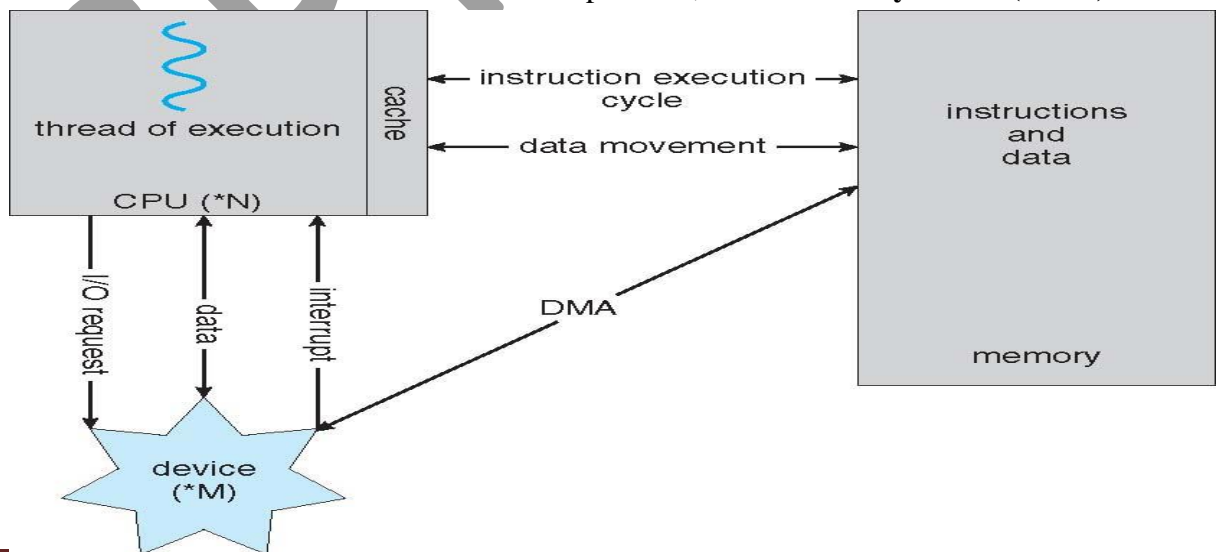
- DMA is used for high-speed I/O devices, able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Computer System Architecture

Categorized roughly according to the number of general-purpose processors used –

Single-Processor Systems –

Most systems use a single processor. The variety of single-processor systems range from PDAs through mainframes. On a single-processor system, there is one main CPU capable of executing instructions from user processes. It contains special-purpose processors, in the form of device-specific processors, for devices such as disk, keyboard, and graphics controllers.

All special-purpose processors run limited instructions and do not run user processes. These are managed by the operating system, the operating system sends them information about their next task and monitors their status.

For example, a disk-controller processor, implements its own disk queue and scheduling algorithm, thus reducing the task of main CPU. Special processors in the keyboard, converts the keystrokes into codes to be sent to the CPU.

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.

Multiprocessor Systems (parallel systems or **tightly coupled systems) –**

Systems that have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices are the multiprocessor systems.

Multiprocessor systems have three main advantages:
1. **Increased throughput -** In multiprocessor system, as there are multiple processors execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly and also due to the competation for the shared resources. The speed-up ratio with *N* processors is not *N,* rather, it is less than N. Thus the speed of the system is not has expected.

2. **Economy of scale -** Multiprocessor systems can cost less than equivalent number of many single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several processes are working on the same data, the data can also be shared among them.

3. **Increased reliability-** In multiprocessor systems functions are shared among several processors. If one processor fails, the system is not halted, it only slows down. The job of the failed processor is taken up, by other processors.

    Two techniques to maintain 'Increased Reliability' - graceful degradation & fault tolerant

    Graceful degradation – As there are multiple processors when one processor fails other process will take up its work and the system goes down slowly.

    Fault tolerant – When one processor fails, its operations are stopped, the system failure is then detected, diagnosed, and corrected.

    The HP NonStop system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of multiple pairs of CPUs. Both processors in the pair execute same instruction and compare the results. If the results differ, then one CPU of the pair is at fault, and both are halted. The process that was being executed is then moved to another pair of CPUs, and the instruction that failed is restarted. This solution is expensive, since it involves special hardware and considerable hardware duplication.

There are two types of multiprocessor systems –
*   **Asymmetric multiprocessing**
*   **Symmetric multiprocessing**

1) **Asymmetric multiprocessing – (Master/Slave architecture)** Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors.
2) **Symmetric multiprocessing (SMP) –** All the processors are considered as peers. There is no master-slave relationship. All the processors have its own registers and CPU, only memory is shared.
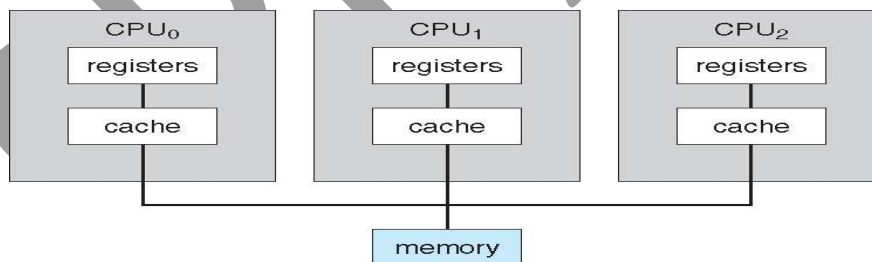
The benefit of this model is that many processes can run simultaneously. *N* processes can run if there are *N* CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP.

A recent trend in CPU design is to include multiple compute **cores** on a single chip. The communication between processors within a chip is more faster than communication between two single processors.
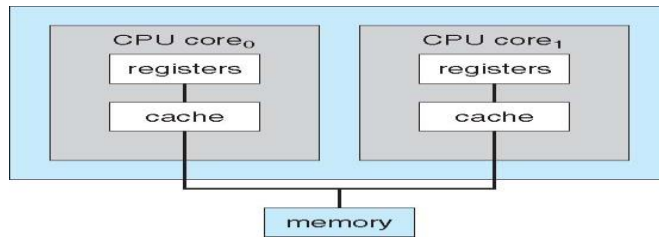
Clustered Systems

Clustered systems are two or more individual systems connected together via network and sharing software resources. Clustering provides **high-availability** of resources and services. The service will continue even if one or more systems in the cluster fail. High availability is generally obtained by storing a copy of files (s/w resources) in the system.

There are two types of Clustered systems – **asymmetric** and **symmetric**

In **asymmetric clustering –** one system is in **hot-stand by mode** while the others are 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 clustering –** two or more systems are running applications, and are monitoring each other. This mode is more efficient, as it uses all of the available hardware. If any system fails, its job is taken up by the monitoring system.

Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN). Parallel clusters allow multiple hosts to access the same data on the shared storage. Cluster technology is changing rapidly with the help of **SAN(storage-area networks)**. Using SAN resources can be shared with dozens of systems in a cluster, that are separated by miles.

Operating-System Structure

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


The operating system keeps several jobs in memory simultaneously as shown in figure. This set of jobs is a subset of the jobs kept in the job pool. 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(in secondary memory). 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. Thus the CPU is never idle.

Secondary mem.   Primary memory   CPU

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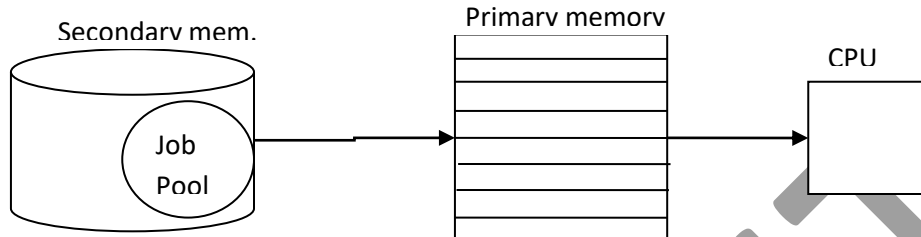
Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

In **Time sharing** (or **multitasking) systems,** a single 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. The user feels that all the programs are being executed at the same time. 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. 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 only, even though it is being shared among many users.

A **multiprocessor system** is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.

CPU   GPU   • • •   CPU

memory
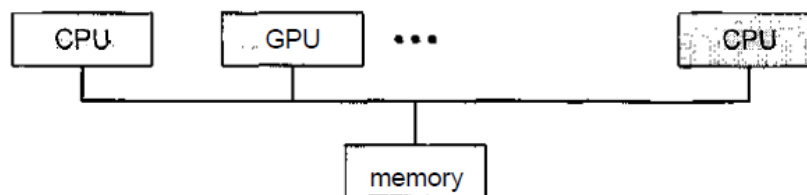
Distributed Systems

Individual systems that are connected and share the resource available in network is called Distributed system. Access to a shared resource increases computation speed, functionality, data availability, and reliability.

A **network** 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. Most operating systems support TCP/IP.

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. A **metropolitan-area network** (MAN) links buildings within a city. A **small-area network** connects systems within a several feet using wireless technology. Eg. BlueTooth and 802.11.

The media to carry networks also vary - copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes, and radios.

A **network operating system** is an operating system that provides features such as file sharing across the network and 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.

Operating-System Operations

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

a) Dual-Mode Operation

Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system.

The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.

The system can be assumed to work in two separate **modes** of operation:
- **user mode** and
- **kernel mode** (**supervisor mode, system mode, or privileged mode).**

A hardware bit of the computer, called the **mode bit**, is used 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 by the operating system and one that is executed by the user.

When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.
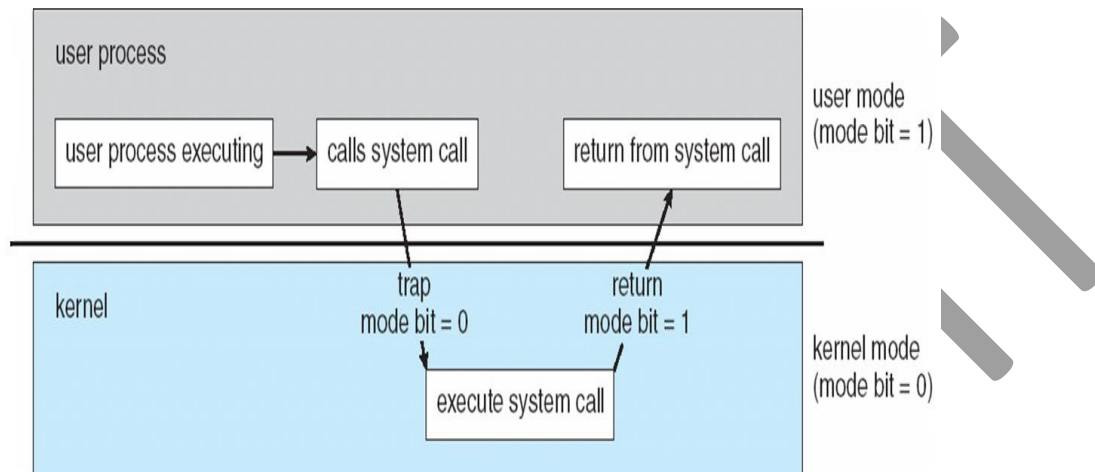
At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the mode bit from 1 to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.

The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to user mode is an example of a privileged instruction.

Initial control is within the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

b) Timer

Operating system uses timer to control the CPU. A user program cannot hold CPU for a long time, this is prevented with the help of 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).

Fixed timer – After a fixed time, the process under execution is interrupted.

Variable timer – Interrupt occurs after varying interval. This is implemented using 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.

Before changing to the user mode, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.

Process Management

A program under execution is a process. A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time. When the process terminates, the operating system reclaims the resources.

The program stored on a disk is a **passive entity** and the program under execution is an **active entity**. A single-threaded process has one **program counter** specifying the next instruction to execute. The CPU executes one instruction of the process after another, until the process completes. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread.

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

- Scheduling process and threads on the CPU
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

Memory Management

Main memory is a large array of words or bytes. Each word or byte has its own address. Main memory is the storage device which can be easily and directly accessed by the CPU. As the program executes, the central processor reads instructions and also reads and writes data from main memory.

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.

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 by user.
- Deciding which processes and data to move into and out of memory.
- Allocating and deallocating memory space as needed.

Storage Management

There are three types of storage management i) File system management ii) Mass-storage management iii) Cache management.

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.

A file is a collection of related information defined by its creator. Commonly, files represent programs 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).

The operating system implements the abstract concept of a file by managing mass storage media. Files are normally organized into directories to make them easier to use. When multiple users have access to files, it may be desirable to control by whom and in what ways (read, write, execute) 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

Mass-Storage Management

As the main memory is too small to accommodate all data and programs, and as the data that it holds are erased when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the 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. Hence, the proper management of disk storage is of central importance to a computer system. The operating system is responsible for the following activities in connection with disk management:

- Free-space management
- Storage allocation
- Disk scheduling

As the secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may depend on the speeds of the disk. Magnetic tape drives and their tapes, CD, DVD drives and platters are **tertiary storage** devices. The functions that operating systems provides include mounting and unmounting media in devices, allocating and freeing the devices for exclusive use by processes, and migrating data from secondary to tertiary storage.

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—as temporary data. When a particular piece of information is required, first we check whether it is in the cache. If it is, we use the information directly from the cache; if it is not in cache, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

Because caches have limited size, **cache management** is an important design problem. Careful selection of the cache size and page replacement policy can result in greatly increased performance.

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.

In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose to retrieve an integer A from magnetic disk to the processing program. The operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory. This operation is followed by copying A to the cache and to an internal register. Thus, the copy of A appears in several places: on the magnetic disk, in main memory, in the cache, and in an internal register.
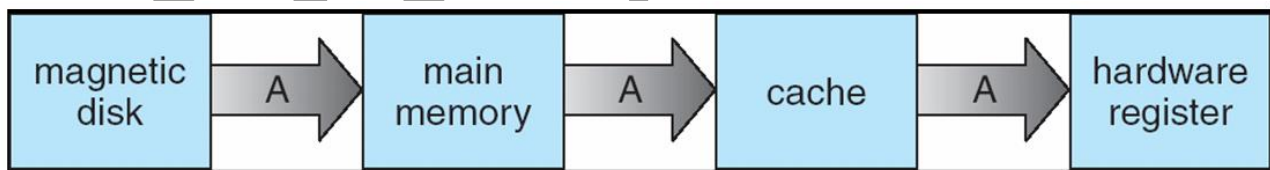
In a multiprocessor environment, in addition to maintaining internal registers, each of the CPUs also contains a local cache. In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute concurrently, any update done to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called **cache coherency,** and it is usually a hardware problem (handled below the operating-system level).

I/O Systems

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. 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.

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.

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, there are mechanisms which 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 for a long time. Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.

Protection is a 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.

Protection improves reliability. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage. A system can have adequate protection but still be prone to failure and allow inappropriate access.

Consider a user whose authentication information is stolen. Her data could be copied or deleted, even though file and memory protection are working. It is the job of **security** to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of service attacks etc.

Protection and security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated **user identifiers (user IDs).** When a user logs in to the system, the authentication stage determines the appropriate user ID for the user.

Distributed Systems
A distributed system is a collection of systems that are networked to provide the users with access to the various resources in the network. Access to a shared resource increases computation speed, functionality, data availability, and reliability.

A **network** is a communication path between two or more systems. Networks vary by the protocols used(TCP/IP,UDP,FTP etc.), the distances between nodes, and the transport media(copper wires, fiber-optic,wireless).
TCP/IP is the most common network protocol. The operating systems support of protocols also varies. Most operating systems support TCP/IP, including the Windows and UNIX operating systems.

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. A **metropolitan-area network** (MAN) connects 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 transportation media to carry networks are also 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.

Special-Purpose Systems

There are different classes of computer systems, whose functions are more limited and specific and it deal with limited computation domains. The systems can be classified as Real-Time Embedded Systems, Multimedia Systems and Handheld 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. Usually, they have little user interface, and more time is spent for monitoring and managing hardware devices, eg. automobile engines and robotic arms.

The Operating Systems, in these embedded systems vary considerably. Some systems have standard operating systems—such as UNIX—with special-purpose applications. Others have special-purpose embedded operating system providing just the functionality desired.

Embedded systems always run **real-time operating systems.** A real-time system is used when there is restricted time for an operation or for the flow of data. A real-time system functions

correctly only if it returns the correct result within its time constraints. Sensors bring data to the computer. The computer must analyze the data and perform certain action.

Some medical imaging systems, automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are real-time systems. A real-time system has well-defined, fixed time constraints. Processing **mustbe** done within the defined constraints, or the system will fail. For instance, the robot arm should be halted before it has smashed into the car, it was building.

Entire houses can be computerized, so that a computer —can control heating and lighting, alarm systems, and even coffee makers. Web access can enable a home owner to tell the house to heat up before she arrives home.

Multimedia 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 like audio files - MP3, DVD movies, video conferencing, and short video clips of movie previews or news. Multimedia applications may also include live webcasts of speeches or sporting events and even live webcams. Multimedia applications can be either audio or video or combination of both. For example, a movie may consist of separate audio and video tracks.

Handheld Systems

Handheld systems include personal digital assistants (PDAs), such as Palm and Pocket-PCs, and cellular telephones. Developers of these systems face many challenges, due to the limited memory, slow processors and small screens in such devices.

The amount of physical memory in a handheld depends upon the device, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager when the memory is not being used. 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 faster speed than the processor in a PC. Faster processors require more power and so, a larger battery is required. Another issue is the usage of I/O devices.

Generally, the limitations in the functionality of PDAs are balanced by their convenience and portability. Their use continues to expand as network connections become more available and other options, such as digital cameras and MP3 players, expand their utility.

Computing Environments

The different computing environments are -

Traditional Computing

The current trend is toward providing more ways to access these computing environments. Web technologies are stretching the boundaries of traditional computing. Companies establish **portals,** which provide web accessibility to their internal servers. **Network computers** are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. Handheld PDAs can also connect to **wireless networks** to use the company's web portal. The fast data connections are allowing home computers to serve up web pages and to use networks. Some homes even have **firewalls** to protect their networks.

In the latter half of the previous century, computing resources were scarce. Years before, systems were either batch or interactive. Batch system processed jobs in bulk, with predetermined input (from files or other sources of data). Interactive systems waited for input from users. To optimize the use of the computing resources, multiple users shared time on these systems. Time-sharing systems used a timer and scheduling algorithms to rapidly cycle processes through the CPU, giving each user a share of the resources.

Today, traditional time-sharing systems are used everywhere. The same scheduling technique is still in use on workstations and servers, but frequently the processes are all owned by the same user (or a single user and the operating system). User processes, and system processes that provide services to the user, are managed so that each frequently gets a slice of computer time.

Client-Server Computing

Designers shifted away from centralized system architecture to - terminals connected to centralized systems. As a result, many of today's systems act as **server systems to** satisfy requests generated by **client systems.** This form of specialized distributed system, called **client-server** system.
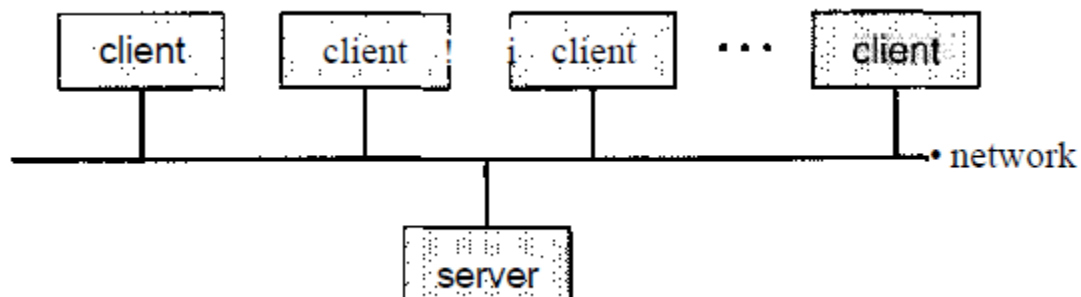
General Structure of Client – Server System

Server systems can be broadly categorized as compute servers and file servers:
- The **compute-server system** provides an interface to which a client can send a request to perform an action (for example, read data); in response, the server executes the action and sends back results to the client. A server running a database that responds to client requests for data is an example of such a svstem.
- 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 the web browsers.

Peer-to-Peer Computing

In this model, clients and servers are not distinguished from one another; here, 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, because all the services must be served by the server. 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. Determining what services are available is accomplished in one of two general ways:
- When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
- A peer acting as a client must know, which node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request. To support this approach, a *discovery protocol* must be provided that allows peers to discover services provided by other peers in the network.

Web-Based Computing

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

The implementation of web-based computing has given rise to new categories of devices, such as **load balancers,** which distribute network connections among a pool of similar servers. Operating systems like Windows 95, which acted as web clients, have evolved into Linux and Windows XP, which can act as web servers as well as clients. Generally, the Web has increased the complexity of devices, because their users require them to be web-enabled.

The design of an operating system is a major task. It is important that the goals of the new system be well defined before the design of OS begins. These goals form the basis for choices among various algorithms and strategies.
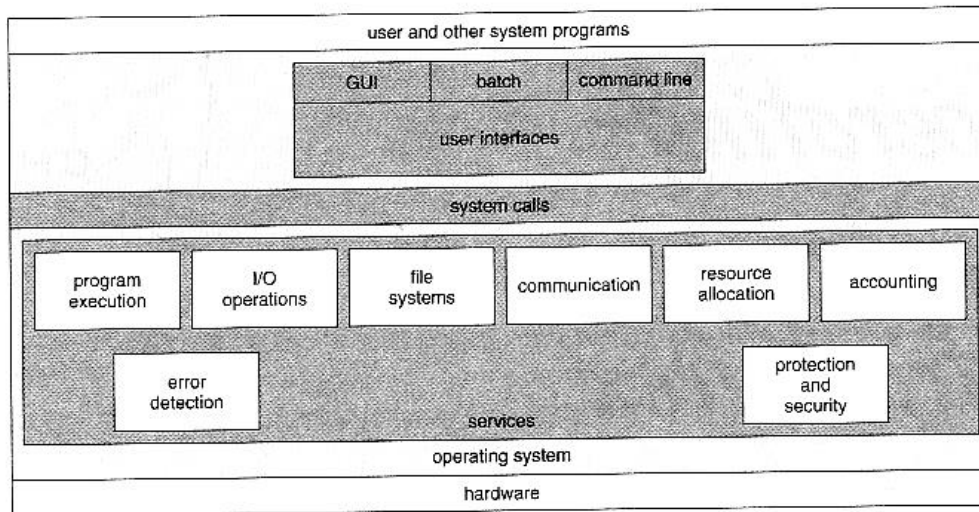
2.1 Operating-System Services

Figure 2.1 A view of 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.

OS provide services for the users of the system, including:

- **User Interfaces** - Means by which users can issue commands to the system. Depending on the operating system these may be a **command-line interface** (e.g. sh, csh, ksh, tcsh, etc.), a **Graphical User Interface** (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a **batch command systems**. In Command Line Interface(CLI)- commands are given to the system. In Batch interface – commands and directives to control these commands are put in a file and then the file is executed. In GUI systems- windows with pointing device to get inputs and keyboard to enter the text.
- **Program Execution** - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions are provided(device drivers) by OS.

- **File-System Manipulation** – Programs need to read and write files or directories. The services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS.
- **Communications** - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented by using the service of OS- like shared memory or message passing.
- **Error Detection** - Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the CPU and memory hardware (such as power failure and memory error), 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).

OS provide services for the efficient operation of the system, including:

- **Resource Allocation** – Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time.
- **Accounting** – There are services in OS to keep track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.
- **Protection and Security** – The owners of information(file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders must also be done, by means of a password.

2.2 User Operating-System Interface

There are several ways for users to interface with the operating system.

1) Command-line interface, or command interpreter, allows users to directly enter commands to be performed by the operating system.
2) Graphical user interface(GUI), allows users to interface with the operating system using pointer device and menu system.
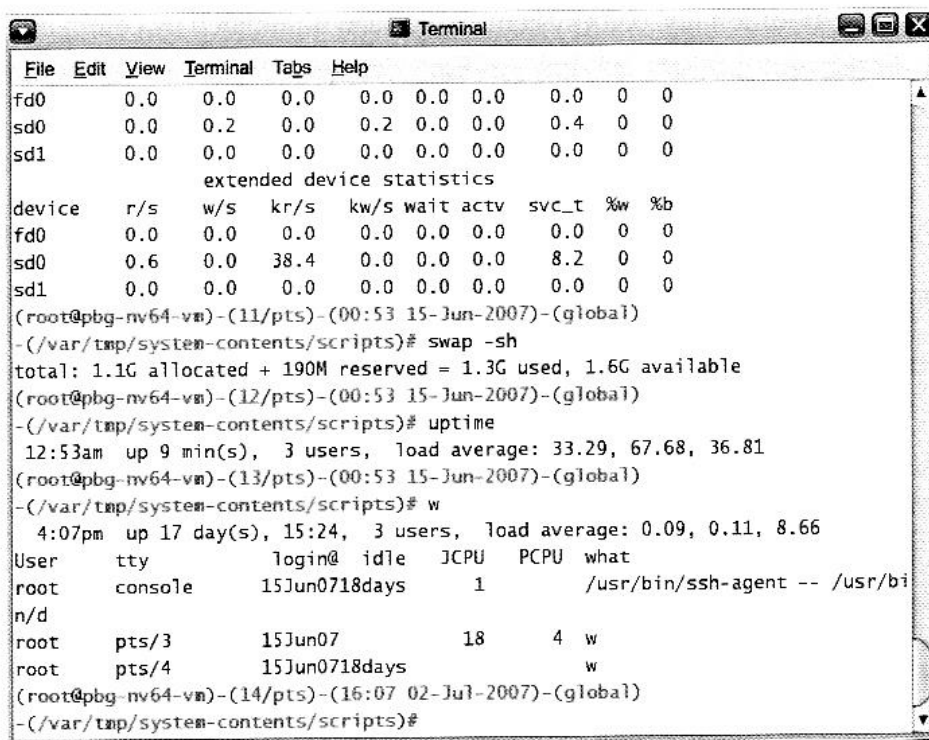
Command Interpreter

Command Interpreters are used to give commands to the OS. There are multiple command interpreters known as shells. In UNIX and Linux systems, there are several different shells, like the *Bourne shell, C shell, Bourne-Again shell, Korn shell,* and others.

The main function of the command interpreter is to get and execute the user-specified command. Many of the commands manipulate files: create, delete, list, print, copy, execute, and so on.

The commands can be implemented in two general ways-

1) The command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a particular section of its code that sets up the parameters and makes the appropriate system call.

2) The code to implement the command is in a function in a separate file. The interpreter searches for the file and loads it into the memory and executes it by passing the parameter. Thus by adding new functions new commands can be added easily to the interpreter without disturbing it.

Figure 2.2 The Bourne shell command interpreter in Solaris 10.

Graphical User Interface, GUI

Another way of interfacing with the operating system is through a user friendly graphical user interface, or GUI. Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window and menu system. The user moves the mouse to position its

pointer on images, or icons on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory-known as a folder-or pull down a menu that contains commands.

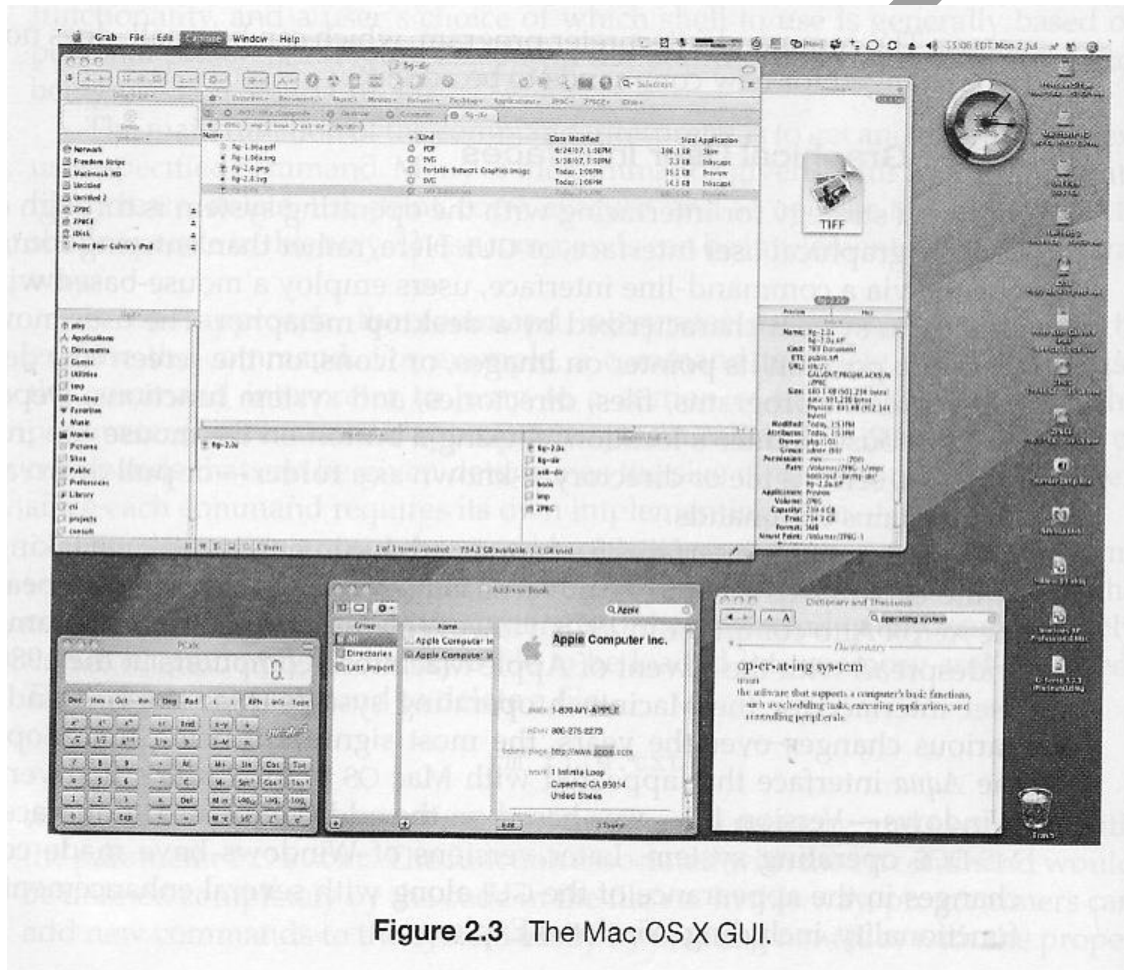Graphical user interfaces first appeared on the Xerox Alto computer in 1973.
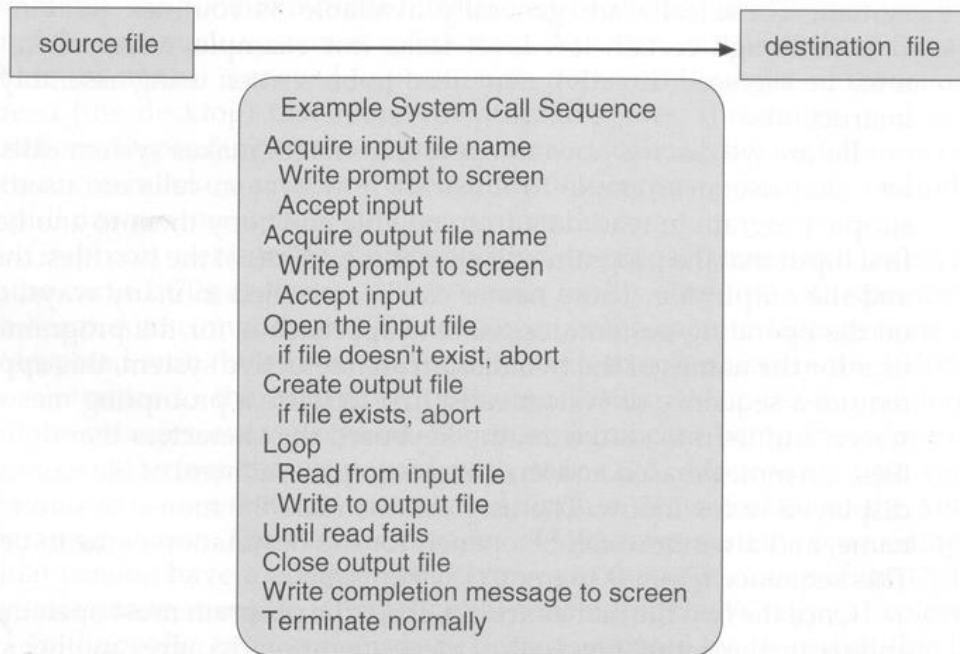

Figure 2.3 The Mac OS X GUI.

Most modern systems allow individual users to select their desired interface, and to customize its operation, as well as the ability to switch between different interfaces as needed.

2.3 System Calls

- System calls is a means to access the services of the operating system.

- Generally written in C or C++, although some are written in assembly for optimal performance.

- The below figure illustrates the sequence of system calls required to copy a file content from one file(input file) to another file (output file).

```
source file  ───────────────────►  destination  file

        Example System Call Sequence
     Acquire input file name
        Write prompt to screen
        Accept input
     Acquire output file name
        Write prompt to screen
        Accept input
     Open the input file
        if file doesn't exist, abort
     Create output file
        if file exists, abort
     Loop
        Read from input file
        Write to output file
     Until read fails
     Close output file
     Write completion message to screen
     Terminate normally
```

There are number of system calls used to finish this task. The first system call is to write a message on the screen (monitor). Then to accept the input filename. Then another system call to write message on the screen, then to accept the output filename. When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console(another system call) and then terminate abnormally (another system call) and create a new one (another system call).

Now that both the fileas are opened, we enter a loop that reads from the input file(another system call) and writes to output file (another system call).

Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window(system call), and finally terminate normally (final system call).

- Most programmers do not use the low-level system calls directly, but instead use an "Application Programming Interface", API.

- The APIs instead of direct system calls provides for greater program portability between different systems. The API then makes the appropriate system calls through the system call interface, using a system call table to access specific numbered system calls, as shown in Figure 2.6.
- Each system call has a specific numbered system call. The system call table (consisting of system call number and address of the particular service) invokes a particular service routine for a specific system call.
- The caller need know nothing about how the system call is implemented or what it does during execution.
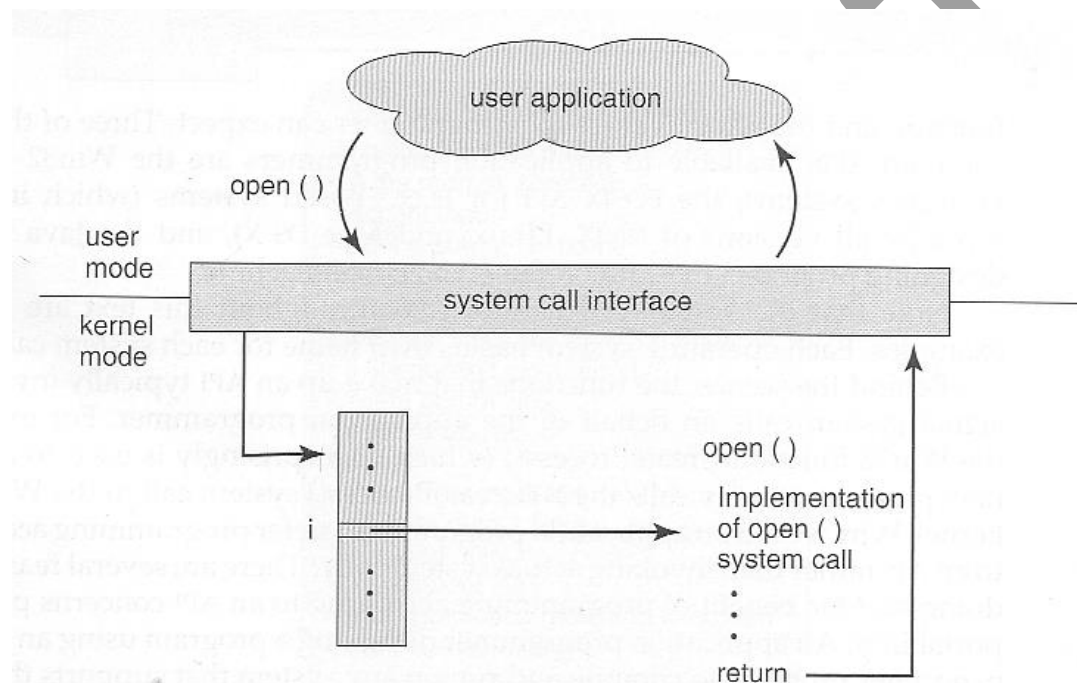

Figure 2.6   The handling of a user application invoking the open() system call.
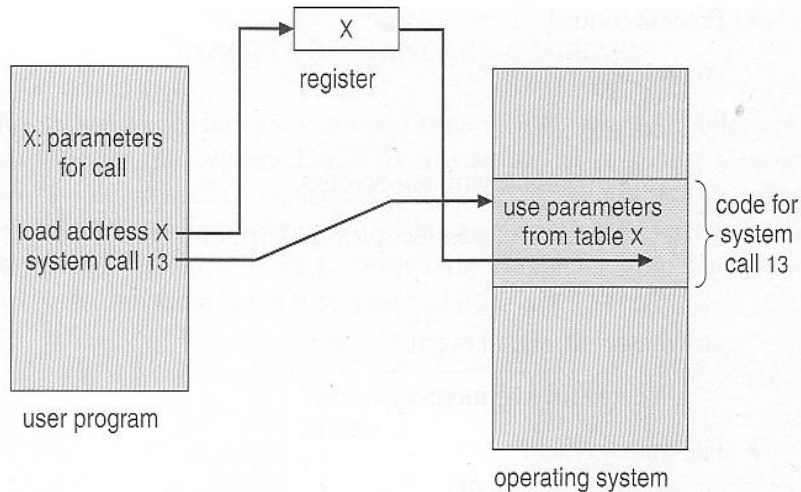
Figure 2.4 Passing of parameters as a table.

Three general methods used to pass parameters to OS are –

- To pass parameters in registers
- If parameters are large blocks, address of block (where parameters are stored in memory) is sent to OS in the register. (Linux & Solaris).
- Parameters can be pushed onto the stack by program and popped off the stack by OS.

2.3.1 Types of System Calls

The system calls can be categorized into six major categories:

- Process Control
- File management
- Device management
- Information management
- Communications
- Protection

- 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
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - 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
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
 - attach or detach remote devices
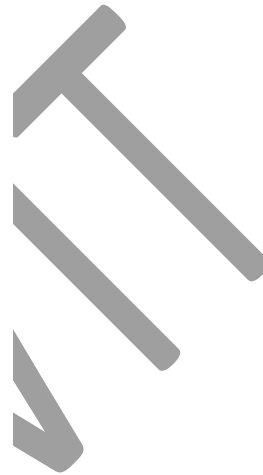
EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

a) Process Control

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed, and eventually stopped.
- When one process pauses or stops, then another must be launched or resumed
- Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
- After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur(wait event). The process sends back a signal when the event has occurred (signal event).

o In DOS, the command interpreter loaded first.Then loads the process and transfers control to it. The interpreter does not resume until the process has completed, as shown in Figure 2.10:
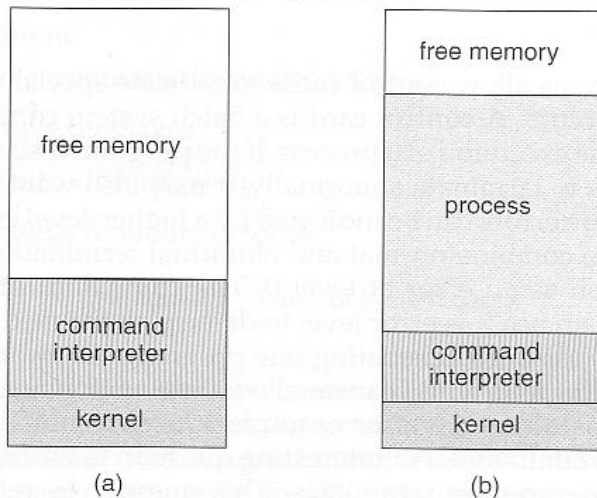
MS-DOS execution. (a) At system startup. (b) Running a program.

Figure 2.10

o Because UNIX is a multi-tasking system, the command interpreter remains completely resident when executing a process, as shown in Figure 2.11 below.
  ▪ The user can switch back to the command interpreter at any time, and can place the running process in the background even if it was not originally launched as a background process.
  ▪ In order to do this, the command interpreter first executes a "fork" system call, which creates a second process which is an exact duplicate (clone) of the original command interpreter. The original process is known as the parent, and the cloned process is known as the child, with its own unique process ID and parent ID.
  ▪ The child process then executes an "exec" system call, which replaces its code with that of the desired process.
  ▪ The parent (command interpreter) normally waits for the child to complete before issuing a new command prompt, but in some cases it can also issue a new prompt right away, without waiting for the child process to complete. (The child is then said to be running "in the background", or "as a background process".)

b) File Management

The file management functions of OS are –

- File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
- After **creating** a file, the file is **opened**. Data is **read** or **written** to a file.
- The file pointer may need to be **repositioned** to a point.
- The file **attributes** like filename, file type, permissions, etc. are set and retrieved using system calls.
- These operations may also be supported for directories as well as ordinary files.

c) Device Management

- Device management system calls include **request device**, **release** device, **read**, **write**, **reposition**, **get/set** device attributes, and logically **attach** or **detach** devices.
- When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
- In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
- Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).

d) Information Maintenance

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- These system calls care used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.

e) Communication

- Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
- The **message passing** model must support calls to:
  - ○ Identify a remote process and/or host with which to communicate.
  - ○ Establish a connection between the two processes.
  - ○ Open and close the connection as needed.

- o Transmit messages along the connection.
- o Wait for incoming messages, in either a blocking or non-blocking state.
- o Delete the connection when no longer needed.
- The **shared memory** model must support calls to:
  - o Create and access memory that is shared amongst processes (and threads.)
  - o Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared. This model is difficult to implement, and it consists of only few system calls.

f) Protection

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non-priveleged users to be granted elevated access permissions under carefully controlled temporary circumstances.

2.4 System Programs

A collection os programs that provide a convenient environment for program development and execution (other than OS) are called system programs or system utilities.

- It is not a part of the kernel or command interpreters.
- System programs may be divided into five categories:
  - o **File management** - programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
  - o **Status information** - Utilities to check on the date, time, number of users, processes running, data logging, etc. System **registries** are used to store and recall configuration information for particular applications.
  - o **File modification** - e.g. text editors and other tools which can change file contents.
  - o **Programming-language support** - E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.
  - o **Program loading and execution** - loaders, dynamic loaders, overlay loaders, etc., as well as interactive debuggers.

o **Communications** - Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.

2.5 Operating-System Design and Implementation

2.5.1 Design Goals

Any system to be designed must have its own goals and specifications. Similarly the OS to be built will have its own goals depending on the type of system in which it will be used, the type of hardware used in the system etc.

- **Requirements** define properties which the finished system must have, and are a necessary steps in designing any large complex system. The requirements may be of two basic groups:

    1. User goals (User requirements)
    2. System goals (system requirements)

    o **User requirements** are features that users care about and understand like system should be convenient to use, easy to learn,reliable, safe and fast.
    o **System requirements** are written for the developers, ie. People who design the OS. Their requirements are like easy to design, implement and maintain, flexible, reliable, error free and efficient.

2.5.2 Mechanisms and Policies

- Policies determine *what* is to be done. Mechanisms determine *how* it is to be implemented.
- Example: in timer, counter and decrementing counter is the mechanism and deciding how long the time has to be set is the policies.
- Policies change overtime. In the worst case, each change in policy would require a change in the underlying mechanism.
- If properly separated and implemented, policy changes can be easily adjusted without re-writing the code, just by adjusting parameters or possibly loading new data / configuration files.

2.5.3 Implementation

- Traditionally OS were written in assembly language.
- In recent years, Os are written in C, or C++. Critical sections of code are still written in assembly language.
- The first OS that was not written in assembly language was the Master Control Program (MCP).
- The advantages of using a higher-level language for implementing operating systems are: The code can be written faster, more compact, easy to port to other systems and is easier to understand and debug.
- The only disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.

2.7 Operating-System Structure

OS structure must be carefully designed. The task of OS is divided into small components and then interfaced to work together.

2.7.1 Simple Structure

Many operating systems do not have well-defined structures. They started as small, simple, and limited systems and then grew beyond their original scope. Eg: MS-DOS.

In MS-DOS, the interfaces and levels of functionality are not well separated. Application programs can access basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS in bad state and the entire system can crash down when user programs fail.

UNIX OS consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system
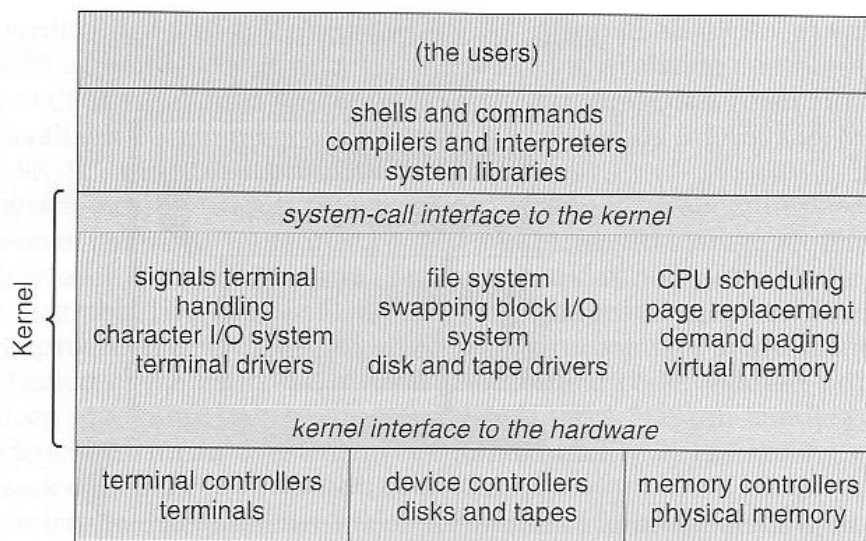
calls.
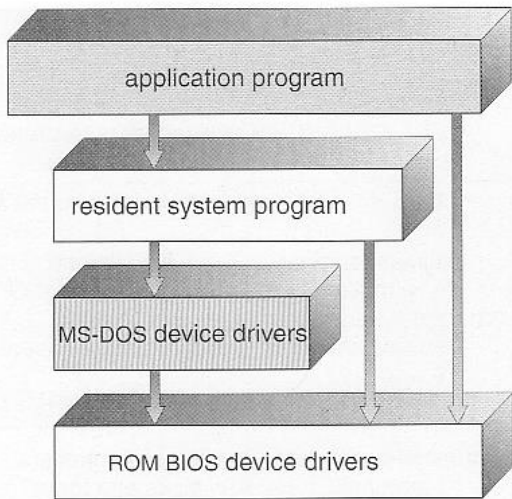
MS-DOS Layer Structure

Figure 2.13 UNIX System Structure

2.7.2 Layered Approach

- The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.

- Bottom layer(layer 0) is the hardware and the topmost layer is the user interface.
- A typical layer, consists of data structure and routines that can be invoked by higher-level layer.

Advantage of layered approach is simplicity of construction and debugging.

The layers are selected so that each uses functions and services of only lower-level layers. So simplifies debugging and system verification. The layers are debugged one by one from the lowest and if any layer doesn't work, then error is due to that layer only, as the lower layers are already debugged. Thus the design and implementation is simplified.

A layer need not know how its lower level layers are implemented. Thus hides the operations from higher layers.
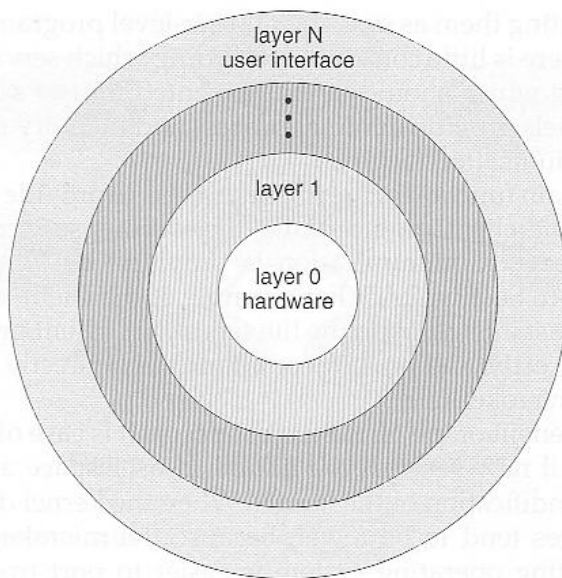

Figure 2.14   A layered Operating System

Disadvantages of layered approach:

- The various layers must be appropriately defined, as a layer can use only lower level layers.
- Less efficient than other types, because any interaction with layer 0 required from top layer. The system call should pass through all the layers and finally to layer 0. This is an overhead.

2.7.3 Microkernels

- The basic idea behind micro kernels is to **remove all non-essential** services from the kernel, thus making the kernel as small and efficient as possible.
- The removed services are implemented as system applications.
- Most microkernels provide basic process and memory management, and message passing between other services.
- **Benefit** of microkernel - System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.
- Disadvantage of Microkernel is, it suffers from reduction in performance due to increases system function overhead.
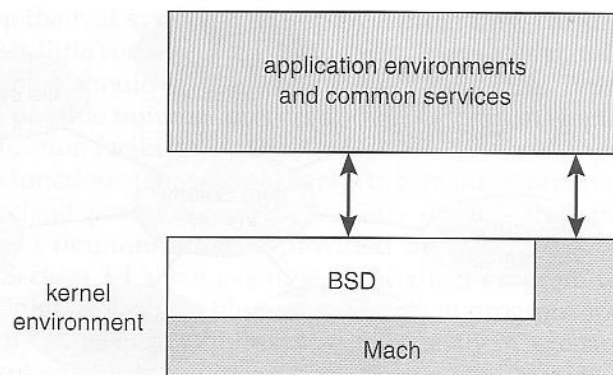

Figure 2.14 The Mac OS X structure.

2.7.4 Modules

- Modern OS development is object-oriented, with a relatively small core kernel and a set of *modules* which can be linked in dynamically.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly. Eg: Solaris, Linux and MacOSX.
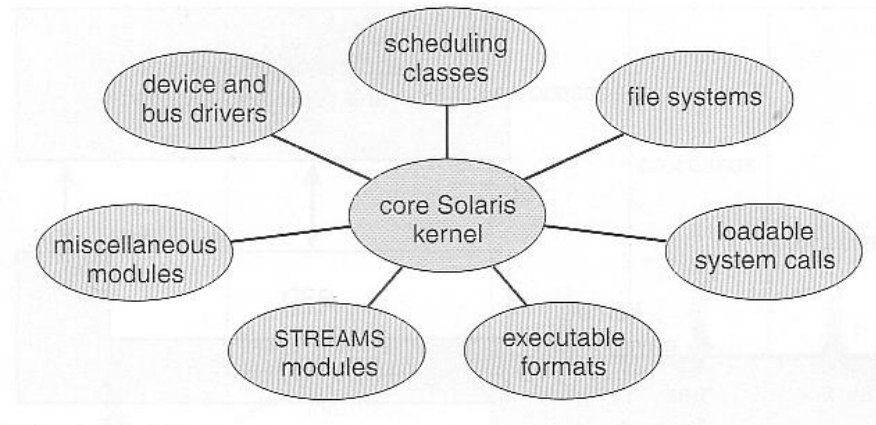
Figure 2.15 Solaris loadable modules

- The Max OSX architecture relies on the Mach microkernel for basic system management services, and the BSD kernel for additional services. Application services and dynamically loadable modules (kernel extensions) provide the rest of the OS functionality.
- Resembles layered system, but a module can call any other module.
- Resembles microkernel, the primary module has only core functions and the knowledge of how to load and communicate with other modules.

2.8 Virtual Machines

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

Creates an illusion that a process has its own processor with its own memory. Host OS is the main OS installed in system and the other OS installed in the system are called guest OS.
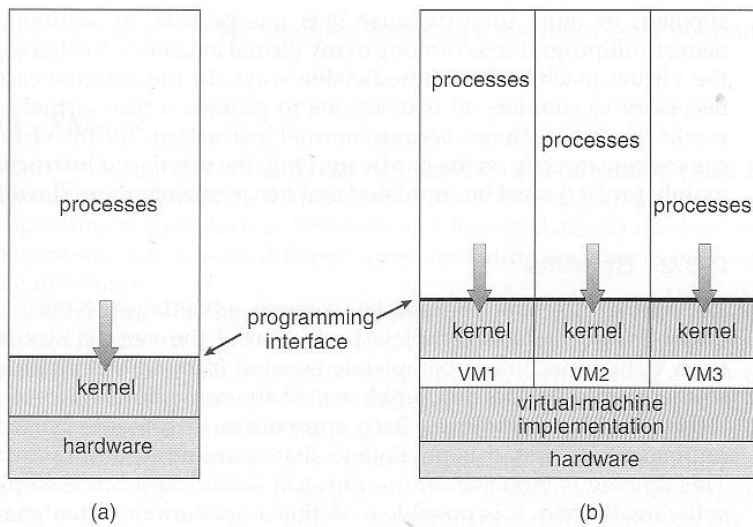
Figure 2.17 System modes. (A) Nonvirtual machine (b) Virtual machine

- Virtual machines first appeared as the VM Operating System for IBM mainframes in 1972.

Benefits

- Able to share the same hardware and run several different execution environments(OS).
- Host system is protected from the virtual machines and the virtual machines are protected from one another. A virus in guest OS, will corrupt that OS but will not affect the other guest systems and host systems.
- Even though the virtual machines are separated from one another, software resources can be shared among them. Two ways of sharing s/w resource for communication are: a)To share a file system volume(part of memory).   b)To develop a virtual communication network to communicate between the virtual machines.
- The operating system runs on and controls the entire machine. Therefore, the current system must be stopped and taken out of use while changes are made and tested. This period is commonly called *system development time.* In virtual machines such problem is eliminated. User programs are executed in one virtual machine and system development is done in another environment.
- Multiple OS can be running on the developer's system **concurrently**. This helps in rapid porting and testing of programmers code in different environments.
- **System consolidation** – two or more systems are made to run in a single system.

Simulation –

Here the host system has one system architecture and the guest system is compiled in different architecture. The compiled guest system programs can be run in an emulator that translates each instructions of guest program into native instructions set of host system.

Para-Virtualization –

This presents the guest with a system that is similar but not identical to the guest's preferred system. The guest must be modified to run on the para-virtualized hardware.

2.8.6 Examples

2.8.6.1 VMware

VMware is a popular commercial application that abstracts Intel 80X86 hardware into isolated virtual machines. The virtualization tool runs in the user-layer on top of the host OS. The virtual machines running in this tool believe they are running on bare hardware, but the fact is that it is running inside a user-level application.

VMware runs as an application on a host operating system such as Windows or Linux and allows this host system to concurrently run several different **guest operating systems** as independent virtual machines.

In below scenario, Linux is running as the host operating system; FreeBSD, Windows NT, and Windows XP are running as guest operating systems. The virtualization layer is the heart of VMware, as it abstracts the physical hardware into isolated virtual machines running as guest operating systems. Each virtual machine has its own virtual CPU, memory, disk drives, network interfaces, and so forth.
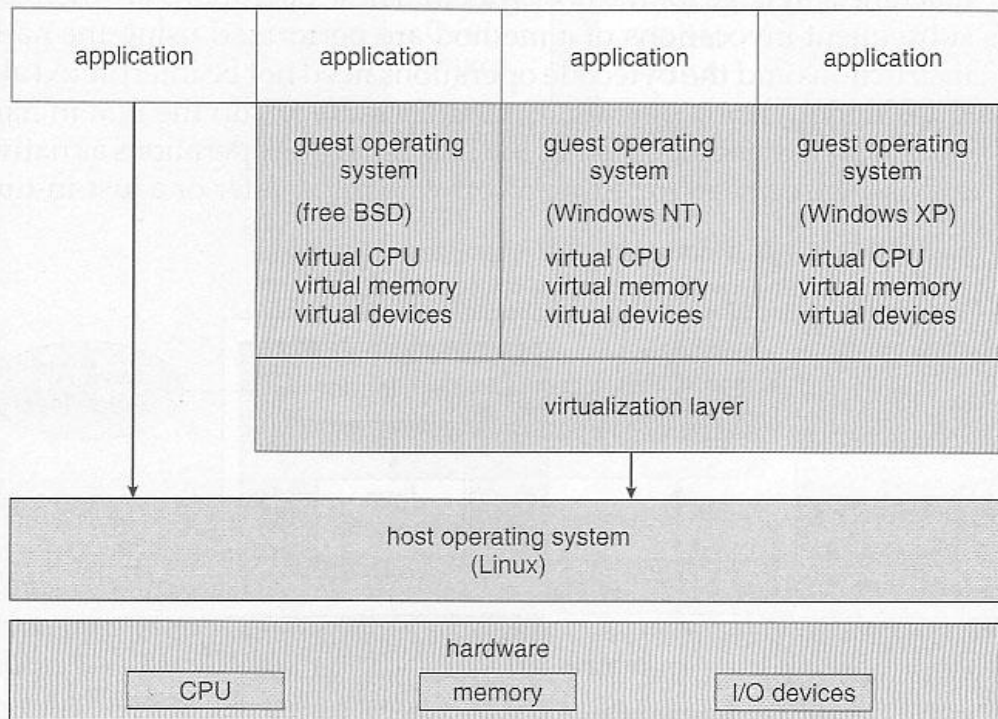
Figure 2.19 VMware architecture

2.8.6.2 The Java Virtual Machine

- Java was designed from the beginning to be platform independent, by running Java only on a Java Virtual Machine, JVM, of which different implementations have been developed for numerous different underlying HW platforms.
- Java source code is compiled into Java byte code in .class files. Java byte code is binary instructions that will run on the JVM.
- The JVM implements memory management and garbage collection.
- JVM consists of class loader and Java Interpreter. Class loader loads compiled .class files from both java program and java API for the execution of java interpreter. Then it checks the .class file for validity.
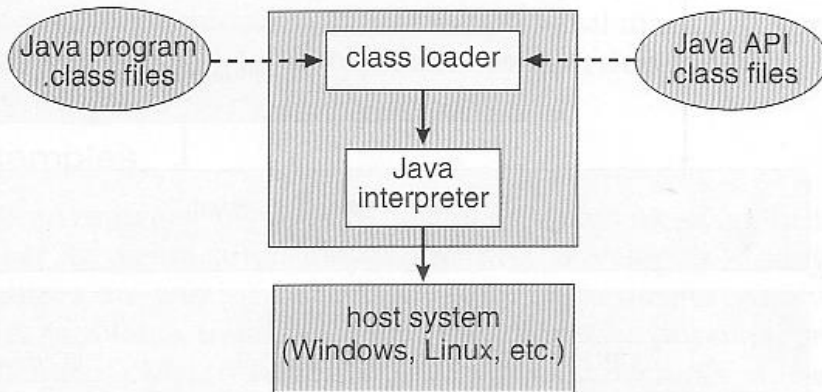
Figure 2.20 The JVM

2.10 Operating-System Generation

- OSes may be designed and built for a specific HW configuration at a specific site, but more commonly they are designed with a number of variable parameters and components, which are then configured for a particular operating environment.
- Systems sometimes need to be re-configured after the initial installation, to add additional resources, capabilities, or to tune performance, logging, or security.

- At one extreme the OS source code can be edited, re-compiled, and linked into a new kernel.
- More commonly configuration tables determine which modules to link into the new kernel, and what values to set for some key important parameters. This approach may require the configuration of complicated makefiles, which can be done either automatically or through interactive configuration programs; Then make is used to actually generate the new kernel specified by the new parameters.
- At the other extreme a system configuration may be entirely defined by table data, in which case the "rebuilding" of the system merely requires editing data tables.
- Once a system has been regenerated, it is usually required to reboot the system to activate the new kernel. Because there are possibilities for errors, most systems provide some mechanism for booting to older or alternate kernels.

2.11 System Boot

The general approach when most computers boot up goes something like this:

- When the system powers up, an interrupt is generated which loads a memory address into the program counter, and the system begins executing instructions found at that address. This address points to the "bootstrap" program located in ROM chips (or EPROM chips) on the motherboard.
- The ROM bootstrap program first runs hardware checks, determining what physical resources are present and doing power-on self tests (POST) of all HW for which this is applicable. Some devices, such as controller cards may have their own on-board diagnostics, which are called by the ROM bootstrap program.
- The user generally has the option of pressing a special key during the POST process, which will launch the ROM BIOS configuration utility if pressed. This utility allows the user to specify and configure certain hardware parameters as where to look for an OS and whether or not to restrict access to the utility with a password.
    - Some hardware may also provide access to additional configuration setup programs, such as for a RAID disk controller or some special graphics or networking cards.
- Assuming the utility has not been invoked, the bootstrap program then looks for a non-volatile storage device containing an OS. Depending on configuration, it may look for a floppy drive, CD ROM drive, or primary or secondary hard drives, in the order specified by the HW configuration utility.
- Assuming it goes to a hard drive, it will find the first sector on the hard drive and load up the fdisk table, which contains information about how the physical hard drive is divided up into logical partitions, where each partition starts and ends, and which partition is the "active" partition used for booting the system.
- There is also a very small amount of system code in the portion of the first disk block not occupied by the fdisk table. This bootstrap code is the first step that is not built into the hardware, i.e. the first part which might be in any way OS-specific. Generally this code knows just enough to access the hard drive, and to load and execute a (slightly) larger boot program.
- For a single-boot system, the boot program loaded off of the hard disk will then proceed to locate the kernel on the hard drive, load the kernel into memory, and then transfer control over to the kernel. There may be some opportunity to specify a particular kernel to be loaded at this stage, which may be useful if a new kernel has just been generated and doesn't work, or if the system has multiple kernels available with different configurations for different purposes. (Some systems may boot different configurations automatically, depending on what hardware has been found in earlier steps.)
- For dual-boot or multiple-boot systems, the boot program will give the user an opportunity to specify a particular OS to load, with a default choice if the user does not pick a particular OS within a given time frame. The boot program then finds the boot loader for the chosen single-boot OS, and runs that program as described in the previous bullet point.
- Once the kernel is running, it may give the user the opportunity to enter into single-user mode, also known as maintenance mode. This mode launches very few if any system services, and does not enable any logins other than the primary log in on the console. This mode is used primarily for system maintenance and diagnostics.

Processes Concept

- A process is a program under execution.
- Its current activity is indicated by PC(Program Counter) and CPU registers.

The Process

Process memory is divided into four sections as shown in the figure below:

- The stack is used to store local variables, function parameters, function return values, return address etc.
- The heap is used for dynamic memory allocation.
- The data section stores global and static variables.
- The text section comprises the compiled program code.
- Note that, there is a free space between the stack and the heap. When the stack is full, it grows downwards and when the heap is full, it grows upwards.
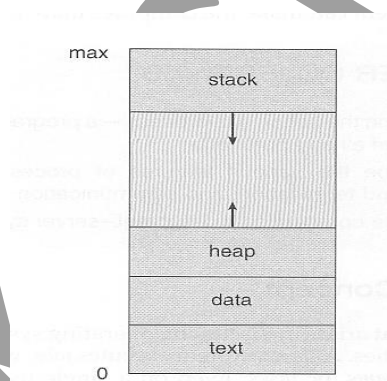

ure 3.1 Process in memory.

Process State

A Process has 5 states. Each process may be in one of the following states –

- **New** - The process is in the stage of being created.
- **Ready** - The process has all the resources it needs to run. It is waiting to be assigned to the processor.
- **Running** – Instructions are being executed..
- **Waiting** - The process is waiting for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
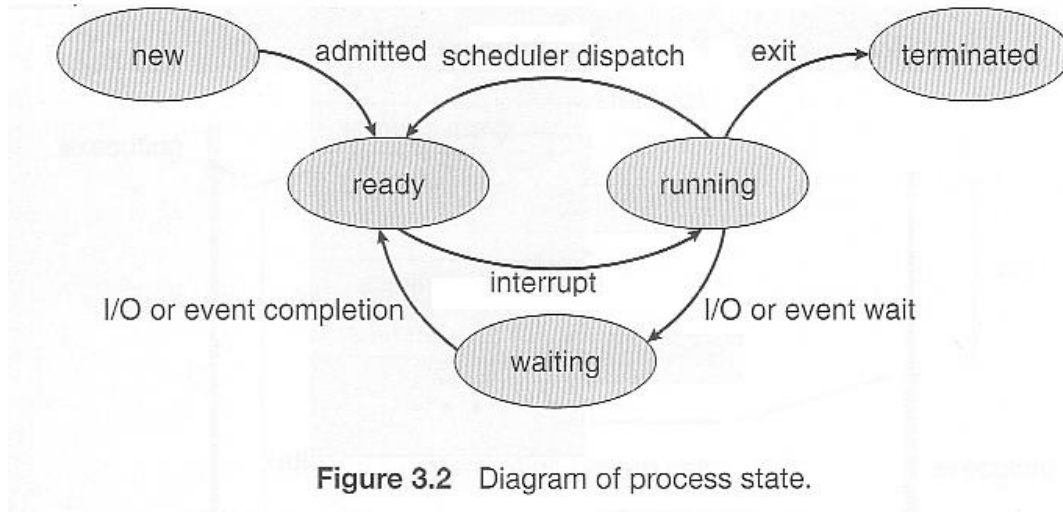- **Terminated -** The process has completed its execution.

Figure 3.2 Diagram of process state.

Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

Process State – The state of the process may be new, ready, running, waiting, and so on.

Program counter – The counter indicates the address of the next instruction to be executed for this process.

CPU registers - The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

CPU scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information – This include information such as the value of the base and limit registers, the page tables, or the segment tables.

Accounting information – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

I/O status information – This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

The PCB simply serves as the repository for any information that may vary from

process to process.

Process Scheduling

Process Scheduler selects an available process for program execution on the CPU. In a multiprocessor system - one process will be under execution and the rest of the processes have to wait until the CPU is free and can be rescheduled.

The main objective of process scheduling is to keep the CPU busy at all times.

Scheduling Queues

- All processes admitted to the system are stored in the **job queue.**
- Processes in main memory and ready to execute are placed in the **ready queue.**
- Processes waiting for a device to become available are placed in **device queues**. There is generally a separate device queue for each device.

These queues are generally stored as a linked list of PCBs. A queue header will contain two pointers - the **head pointer** pointing to the first PCB and the **tail pointer** pointing to the last PCB in the list. Each PCB has a pointer field that points to the next process in the queue.

When a process is allocated to the **CPU,** it executes for a while and eventually quits, interrupted, or waits for the completion of an I/O request. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk in the device queue.
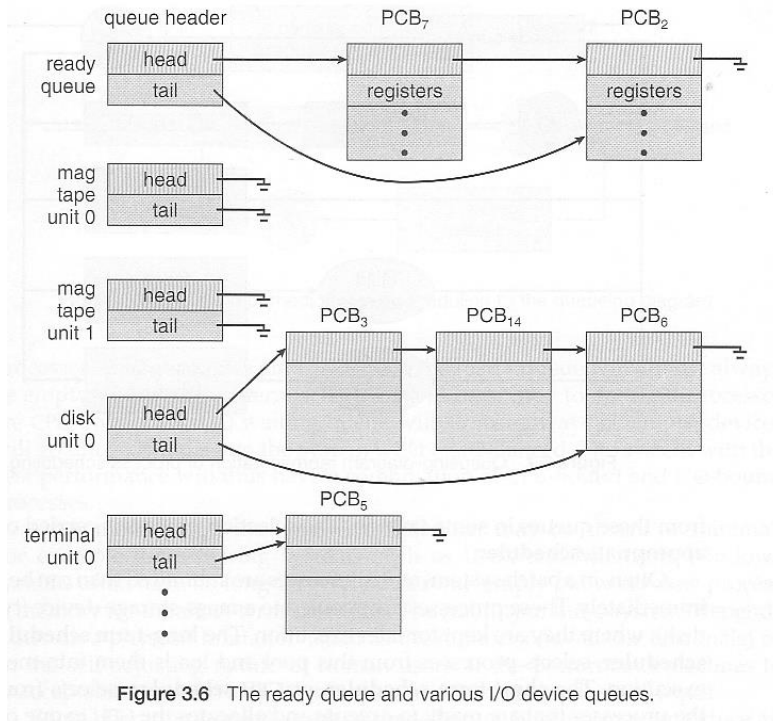
Figure 3.6 The ready queue and various I/O device queues.

A common representation of process scheduling is a *queueing diagram*. Each rectangular box in the diagram represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution and is given the CPU. Once the process is allocated the CPU and is executing, one of several events could occur:

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues.
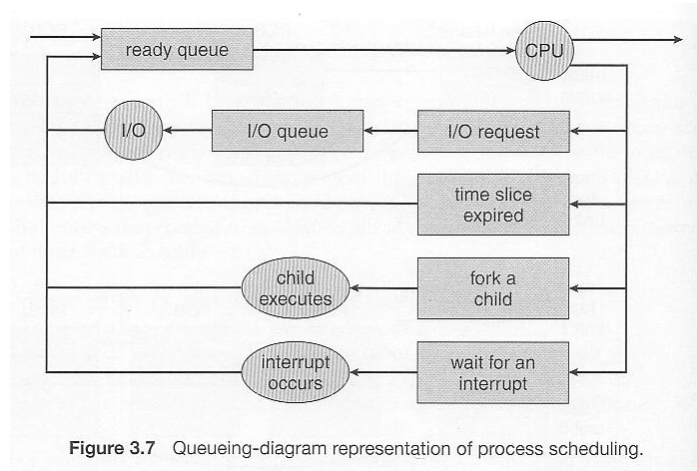
Figure 3.7 Queueing-diagram representation of process scheduling.

Schedulers

Schedulers are software which selects an available program to be assigned to CPU.

- A **long-term scheduler** or **Job scheduler** – selects jobs from the job pool (of secondary memory, disk) and loads them into the memory.

  If more processes are submitted, than that can be executed immediately, such processes will be in secondary memory. It runs infrequently, and can take time to select the next process.

- The **short-term scheduler**, or **CPU Scheduler** – selects job from memory and assigns the CPU to it. It must select the new process for CPU frequently.

- The **medium-term scheduler** - selects the process in ready queue and reintroduced into the memory.

Processes can be described as either:
I/O-bound process – spends more time doing I/O than computations,
CPU-bound process – spends more time doing computations and few I/O operations.

An efficient scheduling system will select a good mix of **CPU-bound** processes and **I/O bound** processes.

- If the scheduler selects **more I/O bound process**, then I/O queue will be full and ready queue will be empty.

- If the scheduler selects **more CPU bound process**, then ready queue will be full and I/O queue will be empty.

Time sharing systems employ a **medium-term scheduler**. It swaps out the process from ready queue and swap in the process to ready queue. When system loads get high, this scheduler will swap one or more processes out of the ready queue for a few seconds, in order to allow smaller faster jobs to finish up quickly and clear the system.

Advantages of medium-term scheduler –
- To remove process from memory and thus reduce the degree of multiprogramming (number of processes in memory).
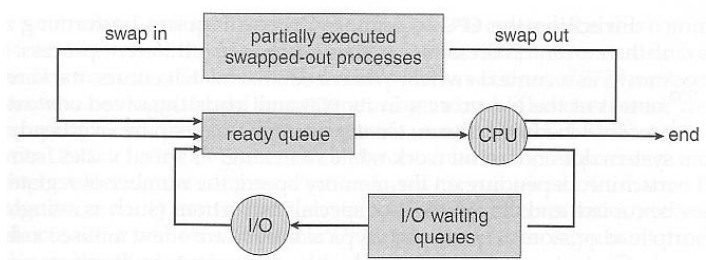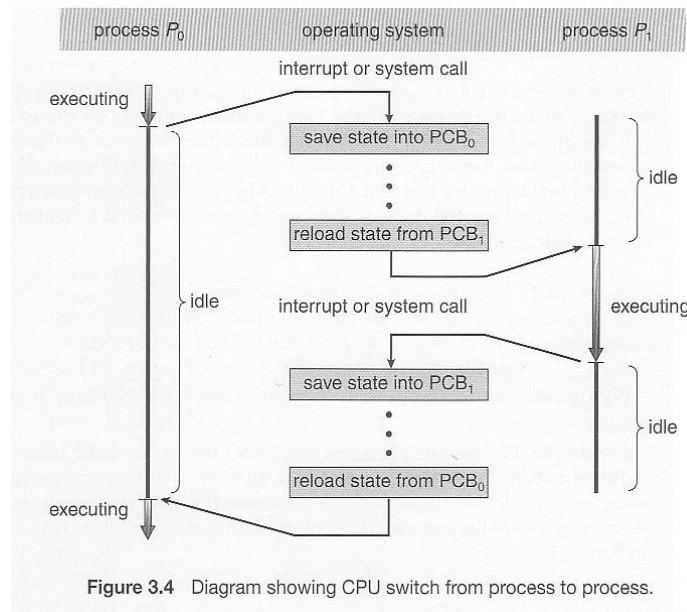- To make a proper mix of processes(CPU bound and I/O bound)

Figure 3.8 Addition of medium-term scheduling to the queueing diagram.

Context Switch

Figure 3.4 Diagram showing CPU switch from process to process.

The task of switching a CPU from one process to another process is called context switching. Context-switch times are highly dependent on hardware support (Number of CPU registers).

Whenever an interrupt occurs (hardware or software interrupt), the state of the currently running process is **saved** into the PCB and the state of another process is **restored** from the PCB to the CPU.

Context switch time is an overhead, as the system does not do useful work while switching.

Operations on Processes

Process Creation

A process may create several new processes. The creating process is called a **parent** process, and the new processes are called the **children** of that process. Each of these new processes may in turn create other processes. Every process has a unique process ID.

On typical Solaris systems, the process at the top of the tree is the '**sched**' process with PID of 0. The '**sched**' process creates several children processes – **init**, **pageout** and **fsflush**. Pageout and fsflush are responsible for managing memory and file systems. The init process with a PID of 1, serves as a parent process for all user processes.
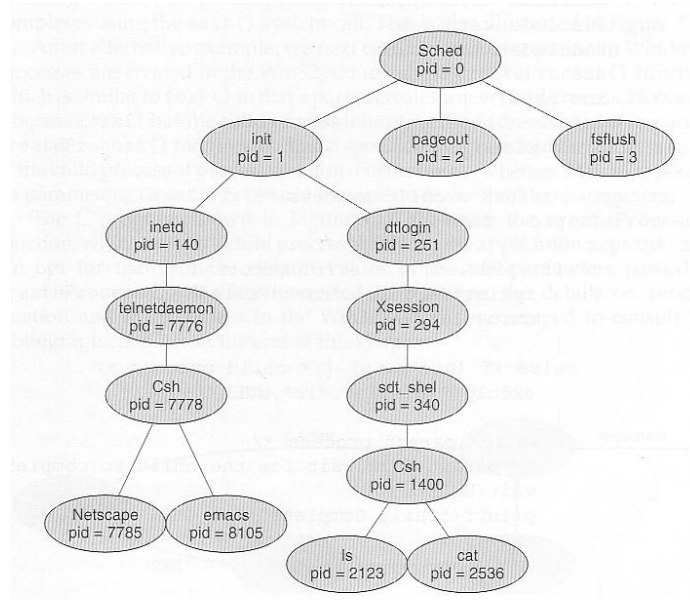
Figure 3.9 A tree of processes on a typical Solaris system.

A process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task. When a process creates a subprocess, the subprocess may be able to obtain its resources in two ways :

- directly from the operating system
- Subprocess may take the resources of the parent process.
    The resource can be taken from parent in two ways –

        o   The parent may have to partition its resources among its children
        o   Share the resources among several children.

There are two options for the parent process after creating the child:

- Wait for the child process to terminate and then continue execution. The parent makes a wait( ) system call.
- Run concurrently with the child, continuing to execute without waiting.

Two possibilities for the address space of the child relative to the parent:

- The child may be an exact duplicate of the parent, sharing the same program and data segments in memory. Each will have their own PCB, including program counter, registers, and PID. This is the behaviour of the **fork** system call in UNIX.
- The child process may have a new program loaded into its address space, with all new code and data segments. This is the behaviour of the **spawn** system calls in Windows.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) {/* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) {/* child process */
        execlp("/bin/ls","ls",NULL);
    }
    else {/* parent process */
    /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
        exit(0);
    }
}
```

Figure 3.10 C program forking a separate process.

In UNIX OS, a child process can be created by **fork()** system call. The **fork** system call, if successful, returns the PID of the child process to its parents and returns a zero to the child process. If failure, it returns -1 to the parent. Process IDs of current process or its direct parent can be accessed using the getpid( ) and getppid( ) system calls respectively.

The parent waits for the child process to complete with the wait() system call. When the child process completes, the parent process resumes and completes its execution.

Figure 3.11 Process creation.
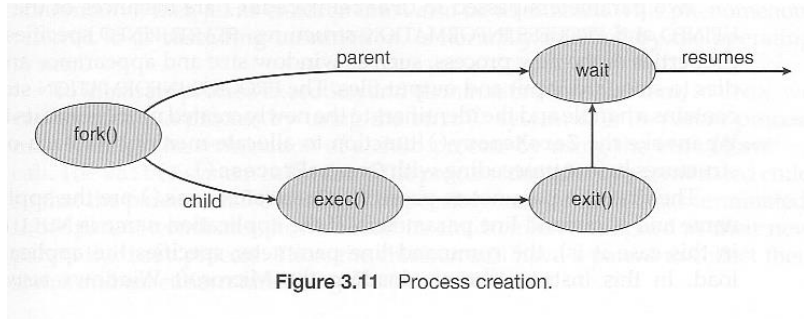
In windows the child process is created using the function **createprocess( )**. The createprocess( ) returns 1, if the child is created and returns 0, if the child is not created.

Process Termination

A process terminates when it finishes executing its last statement and asks the operating system to delete it, by using the **exit**( ) system call. All of the resources assigned to the process like memory, open files, and I/O buffers, are deallocated by the operating system.

A process can cause the termination of another process by using appropriate system call. The parent process can terminate its child processes by knowing of the PID of the child.

A parent may terminate the execution of children for a variety of reasons, such as:

- The child has exceeded its usage of the resources, it has been allocated.
- The task assigned to the child is no longer required.
- The parent is exiting, and the operating system terminates all the children. This is called cascading termination.

Note : Processes which are trying to terminate but which cannot because their parent is not waiting for them are termed **zombies**. These are eventually inherited by init as orphans and killed off. (Modern UNIX shells do not produce as many orphans and zombies as older systems used to.)

Interprocess Communication

Processes executing may be either co-operative or independent processes.

- **Independent Processes** – processes that cannot affect other processes or be affected by other processes executing in the system.
- **Cooperating Processes** – processes that can affect other processes or be affected by other processes executing in the system.

Co-operation among processes are allowed for following reasons –

- Information Sharing - There may be several processes which need to access the same file. So the information must be accessible at the same time to all users.
- Computation speedup - Often a solution to a problem can be solved faster if the problem can be broken down into sub-tasks, which are solved simultaneously (particularly when multiple processors are involved.)
- Modularity - A system can be divided into cooperating modules and executed by sending information among one another.
- Convenience - Even a single user can work on multiple task by information sharing.

Cooperating processes require some type of inter-process communication. This is allowed by two models : 1) Shared Memory systems 2)Message Passing systems.
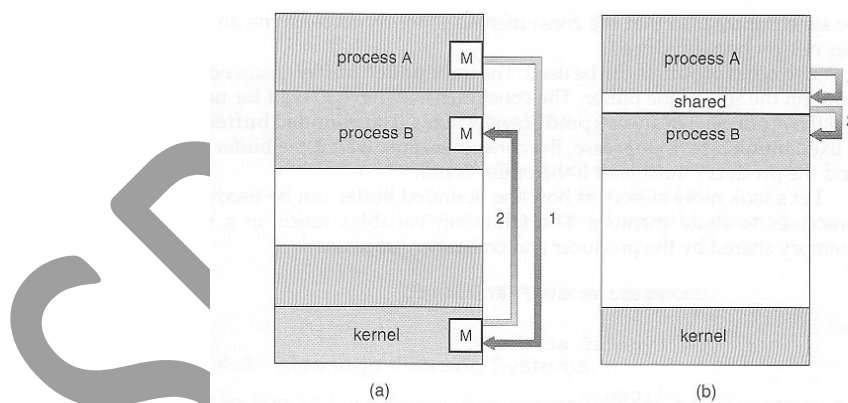

Figure 3.13 Communications models. (a) Message passing. (b) Shared memory.

Sl. No.	Shared Memory	Message passing
1.	A region of memory is shared by communicating processes, into which the information is written and read	Message exchange is done among the processes by using objects.
2.	Useful for sending large block of data	Useful for sending small data.

3.	System call is used only to create shared memory	System call is used during every read and write operation.
4.	Message is sent faster, as there are no system calls	Message is communicated slowly.

- Shared Memory is faster once it is set up, because no system calls are required and access occurs at normal memory speeds. Shared memory is generally preferable when large amounts of information must be shared quickly on the same computer.
- Message Passing requires system calls for every message transfer, and is therefore slower, but it is simpler to set up and works well across multiple computers. Message passing is generally preferable when the amount and/or frequency of data transfers is small.
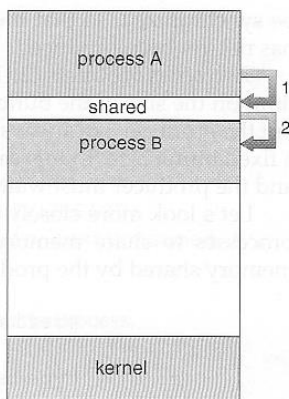
Shared-Memory Systems

A region of shared-memory is created within the address space of a process, which needs to communicate. Other processes that needs to communicate uses this shared memory.

The form of data and position of creating shared memory area is decided by the process. Generally a few messages must be passed back and forth between the cooperating processes first in order to set up and coordinate the shared memory access.

The process should take care that the two processes will not write the data to the shared memory at the same time.

Producer-Consumer Example Using Shared Memory

This is a classic example, in which one process is producing data and another process is consuming the data.

The data is passed via an intermediary buffer (shared memory). The producer puts the data to the buffer and the consumer takes out the data from the buffer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.

There are two types of buffers into which information can be put –

- Unbounded buffer
- Bounded buffer

With Unbounded buffer, there is no limit on the size of the buffer, and so on the data produced by producer. But the consumer may have to wait for new items.

With bounded-buffer – As the buffer size is fixed. The producer has to wait if the buffer is full and the consumer has to wait if the buffer is empty.

This example uses shared memory as a circular queue. The **in** and **out** are two pointers to the array. Note in the code below that only the producer changes "in", and only the consumer changes "out".

- First the following data is set up in the shared memory area:

```
#define BUFFER_SIZE 10          //buffer size
Typedef struct {
   . . .
} item;
item buffer[ BUFFER_SIZE ];
int in = 0;
int out = 0;
```

- The producer process –
  Note that the buffer is full when [(in+1)%BUFFER_SIZE == out]

```
item nextProduced;
while( true )
{
      /* Produce an item and store it in nextProduced */
      nextProduced = makeNewItem( . . . );

      /* Wait for space to become available */
      while( ( ( in + 1 ) % BUFFER_SIZE ) == out )  //full
         ; /* Do nothing */

      /* And then, if not full store the item */
      buffer[ in ] = nextProduced;
      in = ( in + 1 ) % BUFFER_SIZE;
}
```

- The consumer process –
  Note that the buffer is empty when [in == out]

```
item nextConsumed;
while( true )
{
        /* Wait for an item to become available */
        while( in == out )     // buffer empty
            ; /* Do nothing */

        /* Get the next available item */
        nextConsumed = buffer[ out ];
        out = ( out + 1 ) % BUFFER_SIZE;
}
```

Message-Passing Systems

A mechanism to allow process communication without sharing address space. It is used in distributed systems.

- Message passing systems uses system calls for "send message" and "receive message".
- A communication link must be established between the cooperating processes before messages can be sent.
- There are three methods of creating the link between the sender and the receiver-
  - o Direct or indirect communication (naming)
  - o Synchronous or asynchronous communication (Synchronization)
  - o Automatic or explicit buffering.

a)Naming

The processes that wants to communicate should have a way to refer eachother. (using some identity)

Direct communication the sender and receiver must explicitly know eachothers name. The syntax for send() and receive() functions are as follows-

send (*P, message*) – send a message to process P

receive(*Q, message*) – receive a message from process Q

Properties of communication link :
- A link is established automatically between every pair of processes that wants to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly one pair of communicating processes
- Between each pair, there exists exactly one link.

Types of addressing in direct communication –

- Symmetric addressing – the above described communication is symmetric communication. Here both the sender and the receiver processes have to name each other to communicate.
- Asymmetric addressing – Here only the sender name is mentioned, but the receiving data can be from any system.

send(P, message) --- Send a message to process *P*

receive(id, message). Receive a message from any process

Disadvantages of direct communication – any changes in the identifier of a process, may have to change the identifier in the whole system(sender and receiver), where the messages are sent and received.

Indirect communication uses shared mailboxes, or ports.

A mailbox or port is used to send and receive messages. Mailbox is an object into which messages can be sent and received. It has a unique ID. Using this identifier messages are sent and received.

Two processes can communicate only if they have a shared mailbox.

The send and receive functions are –

send(*A, message*) – send a message to mailbox A
receive(*A, message*) – receive a message from mailbox A

Properties of communication link:
- A link is established between a pair of processes only if they have a shared mailbox
- A link may be associated with more than two processes

- Between each pair of communicating processes, there may be any number of links, each link is associated with one mailbox.

A mail box can be owned by the operating system. It must take steps to –
— create a new mailbox
— send and receive messages from mailbox
— delete mailboxes.

b) Synchronization

The send and receive messages can be implemented as either **blocking** or **non-blocking**.

- **Blocking (synchronous) send -** sending process is blocked (waits) until the message is received by receiving process or the mailbox.
- **Non-blocking (asynchronous) send** - sends the message and continues (doesnot wait)

- **Blocking (synchronous) receive -** The receiving process is blocked until a message is available
- **Non-blocking (asynchronous) receive** - receives the message without block. The received message may be a valid message or null.

c) Buffering

when messages are passed, a temporary queue is created. Such queue can be of three capacities:

- **Zero capacity** – The buffer size is zero (buffer does not exist). Messages are not stored in the queue. The senders must block until receivers accept the messages.
- **Bounded capacity**- The queue is of fixed size(n). Senders must block if the queue is full. After sending 'n' bytes the sender is blocked.
- **Unbounded capacity** - The queue is of infinite capacity. The sender never blocks.