MODULE-1

CHAPTER 1 INTRODUCTION

1.1 What Operating Systems Do

An Operating System (OS) is system software that manages the computer hardware.

- It provides a basis for application programs and acts as an intermediary between the computer users and the computer hardware.
- The purpose of an OS is to provide an environment in which the user can execute the program in a convenient & efficient manner.

Operating system goals:

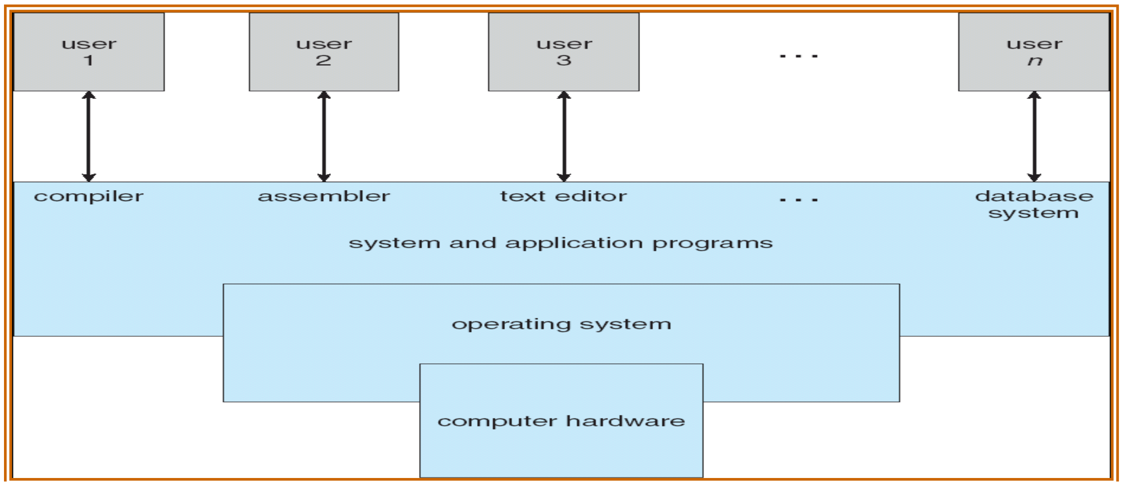
- Make the computer system convenient to use. It hides the difficulty in managing the hardware.
- Use the computer hardware in an efficient manner
- Provide and environment in which user can easily interface with computer.
- It is a resource allocator.

Computer System Structure (Components of Computer System)

A computer system can be divided into **four components**

- ✓ **Hardware**: The Hardware consists of memory, CPU, ALU, I/O devices, peripherals devices & storage devices.
- ✓ **OS:** The OS controls & co-ordinates the use of hardware among various application programs for various users.
- ✓ **Application Program:** The application programs includes word processors, spread sheets, compilers & web browsers which defines the ways in which the resources are used to solve the problems of the users.
- ✓ **User:** Who works/executes the required function.

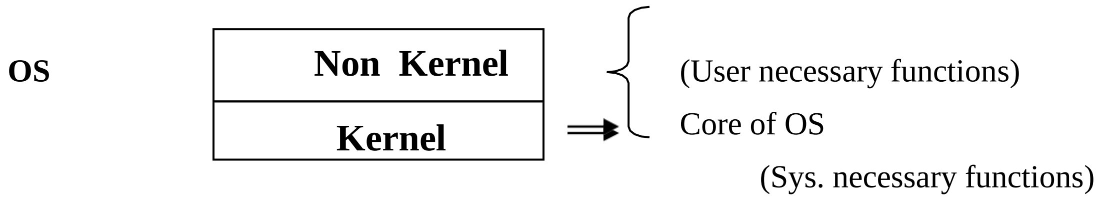
The following **figure** shows the abstract view of the components of a computer system



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.



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

To completely understand the role of operating system two views are considered as below:

1. User View:

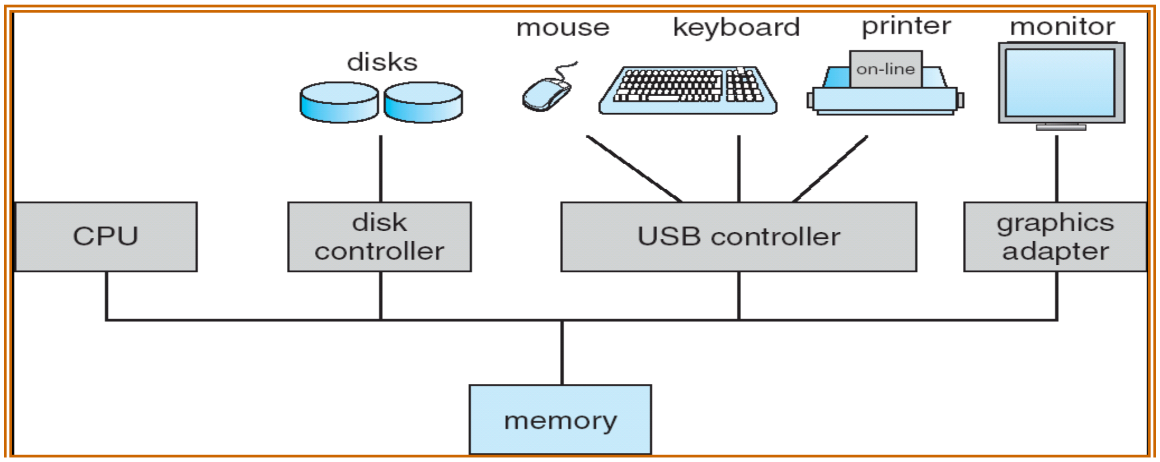
- ✓ The user view of the computer depends on the interface used. Some users may use PC's. Such system is designed for one user. Here the OS is designed for **ease of use** where some attention is mainly on performances and not on the resource utilization.
- ✓ Some users may use a terminal connected to a mainframe or mini computers. Other users may access the same computer through other terminals. These users may share resources and exchange information. In this case the OS is designed to maximize **resource utilization**- so that all available CPU time, memory & I/O are used efficiently.
- ✓ Other users may sit at workstations, connected to the networks of other workstation and servers, then the user have a system unit of their own and shares resources and files with other systems. In this case OS is designed to compromise between individual usability & resource utilization.
- ✓ Users of **handheld** systems expects the OS to be designed for ease of use and performance per amount of battery life.
- ✓ Other systems like embedded systems used in home devices (like washing m/c) & automobiles do not have any user interaction. There are some LEDs to show the status of its work.

2. System View:

- ✓ An operating system can be viewed as **resource allocator and control program**.
- ✓ A computer system has many resources such as CPU Time, memory space, file storage space, I/O devices and so on that may be used to solve a problem.
- ✓ The OS acts as a manager of these resources and decides how to allocate these resources to programs and the users so that it can operate the computer system efficiently and fairly.
- ✓ A different view of an OS is that it controls various I/O devices & user programs i.e. an OS is a **control program** which manages the execution of user programs to prevent errors and improper use of the computer.

1.2 Computer System Organization

- **Computer system operation**
  - ✓ A general purpose computer system consists of one or more CPUs and device controllers connected through common bus providing access to shared memory as shown in below **figure.**



  - ✓ 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.
  - ✓ CPUs and device controllers can execute concurrently competing for memory utilization and **memory controller** synchronizes the memory access.
  - ✓ 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.

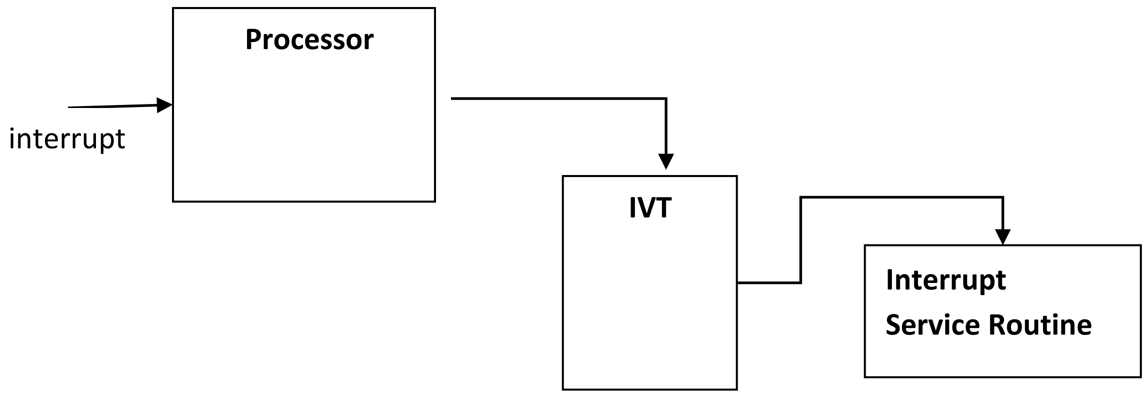
Switch on $\longrightarrow$ 'Bootstrap' program

- 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



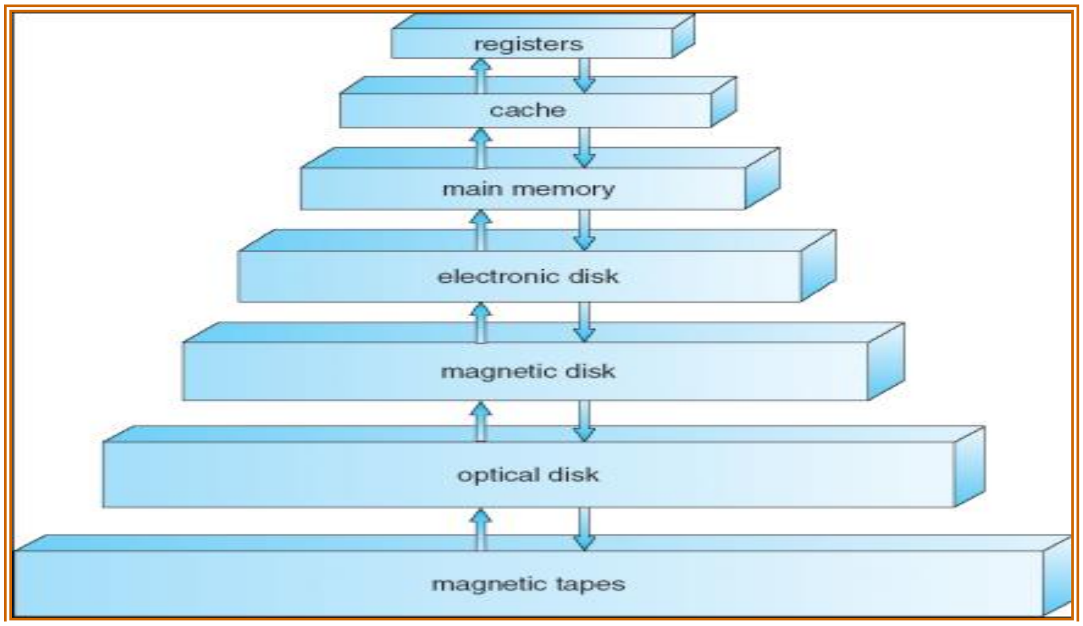
Stored at a fixed location

- **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:
  - ✓ Main memory is usually too small to store all needed programs and data permanently. 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. V**olatile 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.

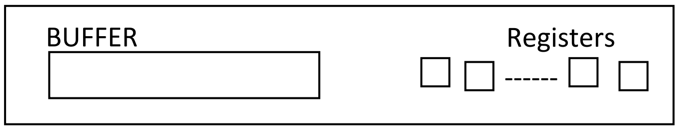


Storage devices hierarchy

- **I/O Structure**

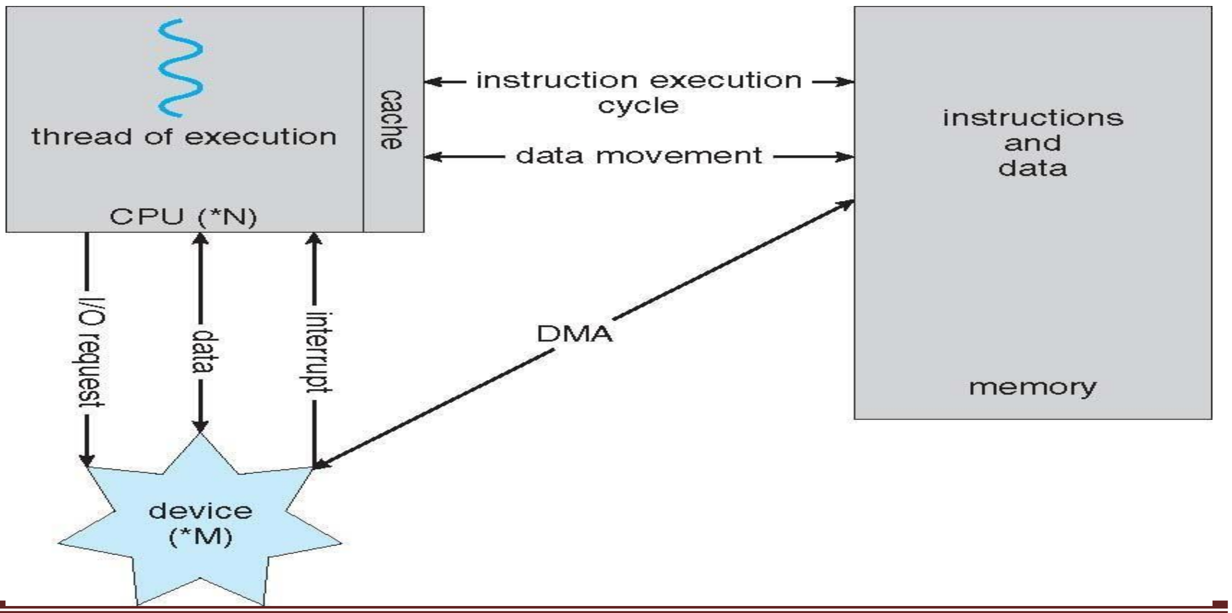
A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

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 

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

1.3 Computer System Architecture

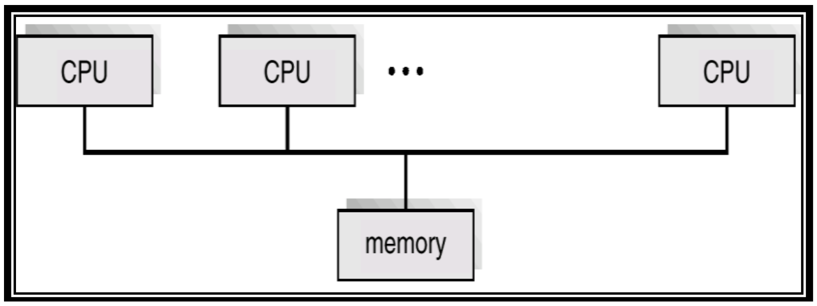
A computer system can be categorized based on number of processors used.

- **Single Processor Systems**
  - ✓ A system that has one main CPU and is capable of executing a general-purpose instruction set, including instructions from the user processes.
  - ✓ Some systems also have special purpose processor to perform specific task, or on mainframes, they come in the form of general purpose processors such as I/O processor. These special purpose processors have limited instruction set and do not run user processes. They are managed by the OS by sending information about the next task and monitor their status.
  - ✓ **Ex1:** A **disk controller microprocessor** receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm. This relieves the main CPU from disk scheduling.
  - ✓ **Ex2:** PCs contain a **microprocessor in the keyboard** to convert the keystrokes into codes to be sent to the CPU.
  - ✓ These special purpose processors do not convert single processor system into multiprocessor system.

- **Multiprocessor Systems**
  - ✓ Multiprocessor systems have more than one processor in close communication. Also known as **Tightly Coupled System** or **Parallel Systems**.
  - ✓ They share computer bus, the clock, memory & peripheral devices.
  - ✓ Two processes can run in parallel.
  - ✓ Multi Processor Systems have **3 advantages**,
    - o **Increased Throughput**: By increasing the number of processors we can get more work done in less time. Speed up ratio with N processors is not N, but it is less than N.
    - o **Economy of Scale:** As Multiprocessor systems share peripherals, mass storage & power supplies, they can save more money than **multiple single processor** systems. If many programs operate on same data, they will be stored on one disk and all processors can share them instead of maintaining data on several systems.
    - o **Increased Reliability**: If a program is distributed properly on several processors, then the failure of one processor will not halt the system but it only slows down.
  - ✓ The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Such systems that provide graceful degradation are **fault tolerant**. Fault tolerant requires a mechanism to allow failure to be detected, and diagnosed and corrected.

  - ✓ Multi processor systems are of **two types**
    - o **Asymmetric Multiprocessing:** Each processor is assigned a specific task. It uses a master slave relationship. A master processor controls the system. The master processors schedules and allocates work to slave processors.

- **Symmetric Multiprocessing (SMP):** Each processor performs all tasks within the OS. SMP means all processors are peers i.e. no master slave relationship exists between processors. Each processor concurrently runs a copy of OS.
- The following **figure** shows SMP architecture.



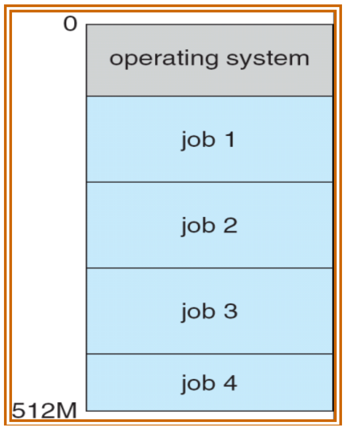
- ✓ The differences between symmetric & asymmetric multiprocessing may be result from either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master & multiple slaves.
- ✓ A recent trend in CPU design is to include multiple compute **cores** on a single chip.
- ✓ **Blade Servers** are recent development in which multiple processor boards, I/O boards, and networking boards are placed in same chassis. Here each processors can boot independently and run their own OS.

- **Clustered Systems**

  - ✓ The clustered systems have multiple CPUs but they are composed of two or more individual systems coupled together.
  - ✓ Clustered systems share storage and are closely linked via LAN Network.
  - ✓ Clustering is usually used to provide high availability.
  - ✓ A layer of software cluster runs on the **cluster nodes**. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine takes ownership of its storage and restarts the applications that were running on failed machine.
  - ✓ Clustered systems can be categorized into two groups
    1. Asymmetric Clustering.
    2. Symmetric clustering.
  - ✓ In **asymmetric clustering** one machine is in **hot standby mode** while others are running the application. The hot standby machine does nothing but it monitors the active server. If the server fails the hot standby machine becomes the active server.
  - ✓ In **symmetric mode** two or more hosts are running the application & they monitor each other. This mode is more efficient since it uses all the available hardware.
  - ✓ Other forms of clusters include **parallel clusters** and **clustering over WAN**.
  - ✓ Parallel clusters allow multiple hosts to access the same data on shared storage.
  - ✓ To provide this shared access, system must also supply access control and locking to ensure that no conflicting operations occur. This function known as **distributed lock manager (DLM)** is included.
  - ✓ Clustering provides better reliability than the multiprocessor systems.

1.4 Operating System Structure

- **Multiprogramming system**
  - ✓ Single user cannot keep CPU and I/O devices busy at all times.
  - ✓ Multiprogramming increases CPU utilization by organizing jobs so that CPU always has one to execute.
  - ✓ The OS has to keep several jobs in memory simultaneously as shown in below **figure** shown below



Memory layout for multiprogramming system

  - ✓ The OS picks up and starts executing one of the jobs.
  - ✓ Eventually if this job may not need the CPU due to some reason like, an I/O operation to complete, then in non multiprogrammed system CPU would sit idle.
  - ✓ But in a multiprogrammed system instead of having the CPU idle the OS switches to the next job in the memory.

- **Timesharing (multitasking)**

  - ✓ It is a logical extension of multiprogramming in which CPU switches among jobs so frequently that users can interact with each job while it is running, creating interactive computing.
  - ✓ Allows many users to share the computer simultaneously.
  - ✓ It requires an **interactive system or a hands-on** computer system that provides direct communication between the user and the system.
  - ✓ Response time should be less than 1 second.
  - ✓ Each user has at least one program executing in memory.
  - ✓ A program loaded into memory and executing is called a **process.**
  - ✓ Timesharing and multiprogramming requires several jobs to be kept simultaneously in memory.
- **Job scheduling**: A job pool consists of all processes residing on disk and awaiting allocation of main memory. If several jobs are ready to be brought into memory and if there is not enough room for them, then the system must choose among them. Making this decision is Job scheduling.

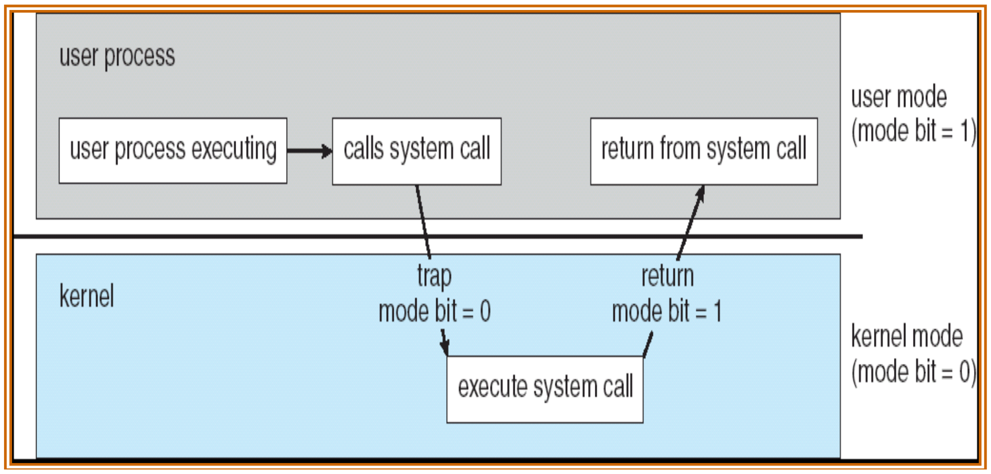
- **CPU scheduling**: If several jobs are ready to run at the same time then the system must choose among them. Making this decision is CPU scheduling.
- **Swapping**: In time shared system processes are swapped in and out of main memory into the disk to ensure reasonable response time. A common method for achieving this is **Virtual memory.** The main advantage of the virtual-memory scheme is that it enables users to run programs that are larger than actual physical memory.

1.5 Operating System Operations

- ✓ Modern OS is **Interrupt driven**.
- ✓ Events are signaled by **interrupt or trap**.
- ✓ An exception or trap is a software generated interrupt either by an error. (For ex: Division by zero) or specific request from a user program.

- **Dual-mode operation**

- ✓ To ensure proper execution of the OS, we must be able to distinguish between **OS code** and **user defined code.**
- ✓ Most computer systems provide hardware support to differentiate among various modes of execution.
- ✓ **Two modes** of operation are
  1. **User mode**
  2. **kernel mode** (also called supervisor, system or privileged mode)
- ✓ **Mode bit** is added to the hardware to indicate current mode User mode(1) and kernel mode(0).
- ✓ When system is executing on behalf of user application, the s/m is in user mode.
- ✓ When a user application requests a service from OS, it must transit from user to kernel mode to fulfill the request as shown in **figure.**



Transition from user to kernel mode

- ✓ At **system boot time**, the hardware starts in kernel mode. The OS 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. Thus whenever the OS gains control of the computer, it is in **kernel mode**. The system always switches to user mode before passing control to user program.

- ✓ The hardware allows **privileged instructions** to be executed only in kernel mode. If an attempt is made to execute privileged instructions in user mode, the hardware does not execute it but rather treats it as an illegal and traps it to the OS. Examples: Instruction to switch to user mode, I/O control instructions, timer management instructions and interrupt management instructions.

- **Timer**

- ✓ Timer is used to prevent a program from getting stuck in an **infinite loop** or not calling system services and never returning control to the OS.
- ✓ Timer can be set to **interrupt** the computer after a specific period.
- ✓ The period may be **fixed or variable**. The **variable timer** is implemented by fixed rate clock and a counter. Whenever the clock ticks, operating system decrements the counter. When counter reaches zero it generates an interrupt.
- ✓ Timer has to be set before scheduling process to regain control or terminate program that exceeds allotted time.

1.6 Process Management

- ✓ A process is a **program in execution**. **Ex1**. A time-shared user program like a compiler is a process. **Ex2**. A word processing program run by an individual user on a PC is a process.
- ✓ A process requires certain **resources** like CPU time, Memory, I/O devices to complete its task.
- ✓ When the process terminates, the OS reclaims all the reusable resources.
- ✓ A program in a file is stored on disk and is a **passive entity**, where as a process is an **active entity** located on main memory.
- ✓ A single threaded process has one **PC (program counter)** specifying the address of the next instruction to be executed. Such processes are sequential i.e. the CPU executes one instruction after the other.
- ✓ A multi-threaded process has multiple **program counters** each pointing to the next instruction to execute for a given thread.
- ✓ A system consists of a collection of processes, some of which are **OS processes** and the rest are **user processes**. All these processes can execute concurrently by multiplexing the CPU among them on a single CPU.
- ✓ The OS is responsible for the following activities of the process management,
    - Creating & deleting of the user & system processes.
    - Suspending and resuming processes.
    - Providing mechanisms for process synchronization.
    - Providing mechanisms for process communication.
    - Providing mechanisms for deadlock handling.

1.7 Memory Management

- ✓ Main memory is the **central** to the operation of the computer system.

- ✓ Main memory is the large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte will have their **own address**.
- ✓ The CPU reads the instruction from main memory during **instruction fetch cycle** & during the **data-fetch cycle** it reads & writes the data.
- ✓ The main memory is the only storage device in which a CPU is able to address & access directly.
- ✓ For a program to be executed, it must be loaded into memory & mapped to absolute addresses. When the program terminates, all available memory will be returned back.
- ✓ To improve the utilization of CPU & the response time several programs will be kept in memory.
- ✓ Several memory management schemes are available & selection depends on the **Hardware design** of the system.
- ✓ The OS is responsible for the following activities
    - o Keeping track of which parts of the memory are used & by whom.
    - o Deciding which process and data to move into and out of memory.
    - o Allocating & deallocating memory space as needed.

1.8 Storage Management

- **File System Management**

- ✓ File management is one of the most visible components of an OS.
- ✓ Computer can store information on different types of **physical media** like Magnetic Disks, Magnetic tapes, optical disks etc.
- ✓ These devices have their own **unique characteristics** like access speed, capacity, data transfer rate, and access method (sequential or random).
- ✓ OS implements the abstract concept of a file by managing mass storage media like tapes, disks etc.,
- ✓ A **file** is a collection of related information defined by its **creator**. They commonly represent programs (source and object) and data. Data files may be numeric, alphabetic or alphanumeric.
- ✓ Files can be organized into **directories** to make them easier to use.
- ✓ The OS is responsible for the following activities,
    - o Creating & deleting files.
    - o Creating & deleting directories.
    - o Supporting primitives for manipulating files & directories.
    - o Mapping files onto secondary storage.
    - o Backing up files on stable (non volatile) storage media.

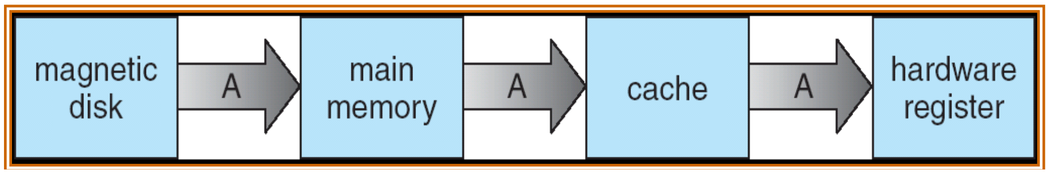
- **Mass Storage management**

- ✓ Computer system must provide **secondary storage** to back up main memory because,
    - o It is too small to accommodate all data and programs.
    - o Data held in this memory is lost when power goes off.

- ✓ Most programs including compilers, assemblers, word processors, editors etc are stored on the disk until loaded into the memory and then use disk as both source and destination of processing. Hence proper management of disk storage is very important.
- ✓ The OS is responsible for the following activities,
    - o Free space management.
    - o Storage allocation.
    - o Disk scheduling.
- ✓ Slower and low cost but high capacity backup storage devices are called **tertiary devices** which are used for back-up of the regular disk data, seldom used data, long term archival storage etc. **Eg**. Magnetic tapes and their drives, CD and DVD drives and platters like tape and optical platters.

- **Caching**

- ✓ It is an important principle of a computer system and is a **fast memory** which is used for storing information on a temporary basis.
- ✓ First, the cache is searched when a particular piece of information is required during processing. If the information already available, then it is directly used from the cache, otherwise we use information from the source, putting a copy in the cache, under the assumption that we will need the information again very soon.
- ✓ Internal **programmable registers** like index registers can be used as high-speed cache for the main memory.
- ✓ Caches have limited size and thus **Cache management** is an important design problem.
- ✓ In a hierarchical storage structure, the same data may appear in different levels of storage system. **For ex**, Suppose that an integer A is to be incremented by 1 is located in file B which resides on disk, the migration of integer A from Disk to Register is shown in below **figure.**



- ✓ Once the increment to A takes place in the internal registers, the value of A differs in various storage systems. The value of A becomes same only after the new value of A is written from the internal register back to the **disk**.
- ✓ In **multitasking environments**, extreme care must be taken to use most recent value, not matter where it is stored in the storage hierarchy.
- ✓ The situation becomes more complicated in **multiprocessor environment**, where each CPU is associated with local cache. A care must be taken to make sure that an update to the value of A in one cache is immediately reflected in all other caches. This situation is called as **cache coherency.**
- ✓ The situation becomes even more complex in a **distributed environment**. Several copies of the same file can be kept on different computers. Since the various replicas

may be accessed and updated concurrently, some distributed systems ensure that, when a replica is updated in one place all other replicas are also updated as soon as possible.

- **I/O Systems**

- ✓ OS hides peculiarities of hardware devices from the user.
- ✓ I/O subsystem consists of several components like,
  - o The memory management component that includes buffering, caching and spooling.
  - o A general device driver interface.
  - o Drivers for specific hardware devices.
- ✓ Only the device driver knows the peculiarities of each of the devices.

1.9 Protection and Security

- ✓ If a computer system has multiple users and allows the concurrent execution of multiple processes, then a protection mechanism is required to regulate access to data.
- ✓ System resources like files, memory segments, CPU etc. are made available to only those processes which have gained **authorization** from OS.
- ✓ **Protection** is a mechanism for controlling the access of processes or users to the resources defined by a computer system.
- ✓ The mechanism must specify the controls to be imposed and what for.
- ✓ The **advantages** of providing protection are: it can improve reliability by detecting latent errors at the interfaces between component subsystems and early detection of interface errors can prevent corruption of good subsystems by another malfunctioning subsystem. Protection can prevent misuse by an unauthorized or incompetent user.
- ✓ The **security system** must defend the system from external and internal attacks. **Attacks** can be of various types like viruses, worms, denial-of-service attack, identity theft, theft of service etc. **Ex**. If a user's authentication information is stolen then the owner's data can be stolen, corrupted or deleted.
- ✓ The mechanism of protection and security must be able to distinguish among all its users. This is possible because the system maintains a list of all **user ids**. These ids are unique per user.
- ✓ When it is required to distinguish among a set of users rather than individual users then group functionality is implemented.
- ✓ A system-wide list of **group names and group ids** are stored.
- ✓ If a user needs to **escalate privileges** for gaining extra permissions then different methods are provided by the OS.

1.10 Distributed Systems

- ✓ A distributed system is a collection of physically separate **heterogeneous computer systems** that are networked to provide the users with access to various resources that the system maintains.

- ✓ A distributed system is one in which Hardware or Software components located at the networked computers communicate& coordinate their actions only by passing messages.
- ✓ Distributed systems depend on **networking** for their functionality. Network may vary by the protocols used, distance between nodes (LAN, WAN, MAN, etc) & transport media.
- ✓ A **network operating system** is an OS that provides features such as file sharing across the network and allows different processes on different computers to exchange messages.
- ✓ **The advantages** of Distributed Systems are,
  - o Resource sharing
  - o Higher reliability
  - o Better price performance ratio
  - o Shorter response time
  - o Higher throughput
  - o Incremental growth

1.11 Special Purpose Systems

The special purpose computers are those whose functions are more limited and whose objectives are to deal with limited computation domains. **Eg**: Realtime Embedded Systems, Multimedia Systems and Handheld Systems.

- **Real- Time Embedded Systems**

- ✓ Embedded computers are found almost everywhere from car engines, robots, alarm systems, medical imaging systems, industrial control systems, microwave ovens, weapon systems etc.
- ✓ This class of computers have very **specific task** and run an OS with very **limited features**. Usually they have limited or **no user interface**.
- ✓ Embedded systems runs on **real time OS**.
- ✓ A real time system should have well defined, **fixed time** constraints. Processing must be done within the defined constraints or the system will fail. Hence they are often used as **controlled device** in a dedicated application. Real time OS uses priority scheduling algorithm to meet the response requirement of a real time application.
- ✓ Real time systems are of **two types**
  - o Hard Real Time Systems
  - o Soft Real Time Systems
- ✓ **A hard real time system** guarantees that the critical tasks to be completed on time. This goal requires that all delays in the system be bounded from the retrieval of stored data to time that it takes the OS to finish the request.
- ✓ In **soft real time system** is a less restrictive one where a critical real time task gets priority over other tasks & retains the property until it completes. Soft real time system is achievable goal that can be mixed with other type of systems. They have limited utility than hard real time systems. Soft real time systems are used in area of multimedia, virtual reality & advanced scientific projects. It cannot be used in

robotics or industrial controls due to lack of deadline support. Soft real time requires two conditions to implement, CPU scheduling must be priority based & dispatch latency should be small.

- **Multimedia Systems**

  ✓ A recent trend in technology is the incorporation of **multimedia data**.
  ✓ Multimedia data consists of audio and video files along with conventional files(text files, word document, etc).
  ✓ The difference from conventional data is that the multimedia data must be delivered or streamed according to some **time** restrictions.
  ✓ Multimedia applications include video conferencing, news stories download over the internet, live webcasts of speeches and so on.

- **Handheld Systems**

  ✓ Handheld systems include Personal Digital Assistants (PDAs), Cellular telephones, palm and pocket PCs and so on, which uses special purpose embedded OS.
  ✓ **Drawbacks** are, because of smaller size they have small amount of memory, slow processors and small display screen.
  ✓ **Memory**-Because of **small size**, OS and applications must manage the memory efficiently. (Making sure all memory allocated is returned back to the memory manager if no longer used).Many handheld devices do not use virtual memory techniques.
  ✓ **Speed-**Processors run at a fraction of the speed of the PC processor. Faster processors require more power. Therefore, OS and applications must not tax the processor.
  ✓ The **small display screen** also limits the output options. To allow the display of the contents of the web pages **web clipping** is done where only a small subset of the web page is delivered and displayed on the screen.
  ✓ **Advantages are**
    - Ability to synchronize with desktops.
    - Small size hence can be carried around easily.

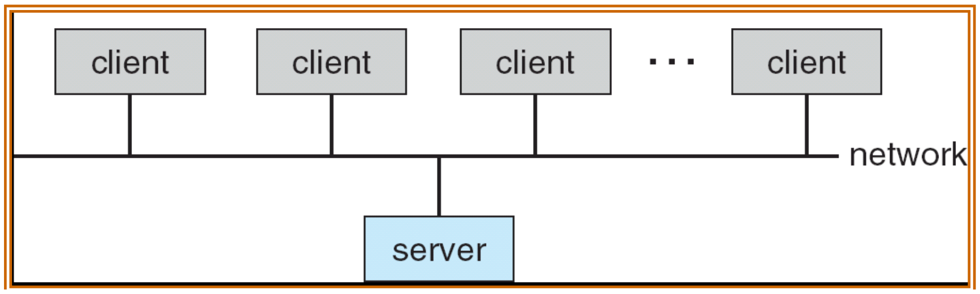
1.12 Computing Environments

- **Traditional Computing**

  ✓ Consider the **"typical office environment"**: Few year's back it consisted of PCs connected to the network with servers providing file and print service, Remote access looked tough and portability was achieved through laptop.
  ✓ Terminals attached to mainframes were common at many companies with even few remote access and portability option.
  ✓ The web technologies are stretching the boundaries of traditional computing. Companies have **portals,** which provide web access to their internal servers. Network computers are terminals that understand the web based computing. Hand held PDAs can also connect to wireless networks to use company's web portal.

- **Client-Server Computing**

  ✓ Many of today's systems act as **server systems** to satisfy requests of clients.
  ✓ This form of specialized distributed system, called **client-server system** has general structure as shown in below **figure**



  ✓ Server system can be **classified** as follows
    a. **Compute-Server Systems**: Provides an interface to which client can send requests to perform some actions, in response the server execute the action and send back result to the client. **Ex**. A server running a database that responds to client requests for data.
    b. **File-Server Systems**: Provides a file system interface where clients can create, update, read & delete files. **Ex**. A web server that delivers files to clients running web browsers.

- **Peer-to-Peer(P2P) Computing**

  ✓ It is another form of a distributed system. Here, clients and servers are not distinguished from one another.
  ✓ All nodes within the system are considered as **peers.** Each can act as a server or a client depending on who is requesting or providing a service.
  ✓ The **advantage** is the removal of bottleneck as the services can be provided by several nodes that are distributed throughout the network.
  ✓ To participate in a P2P system a node must first join the network of peers. On joining, the new node can provide and request for services.
  ✓ **Determining** what services are available in the network can be accomplished in one of **two** methods,

    1. When a node joins a network, it registers its services with a centralized lookup service on the network. Any node wants service, first contacts the centralized lookup service to determine which nodes provides the service. Then the communication takes place between the client and the service provider.
    2. A peer which is a client broadcasts a request for service to all nodes in the network. The nodes that provide the service responds to the requesting peer. A discovery protocol is used by the peers to discover the services provided by other peers.

- **Web Based Computing**

  - ✓ It leads to more access by wider variety of devices other than PCs workstations, PDAs, and cell phones.
  - ✓ Web computing has increased the emphasis on networking.
  - ✓ Devices that were not previously networked have been wired or wireless nowadays.
  - ✓ The network connectivity is faster through improved network technology and optimized network implementation code.
  - ✓ Web based computing has given rise to a new category of devices called **load balancers** which distribute network connections among a pool of similar servers.

CHAPTER 2 OPERATING SYSTEM STRUCTURES

2.1 Operating System Services

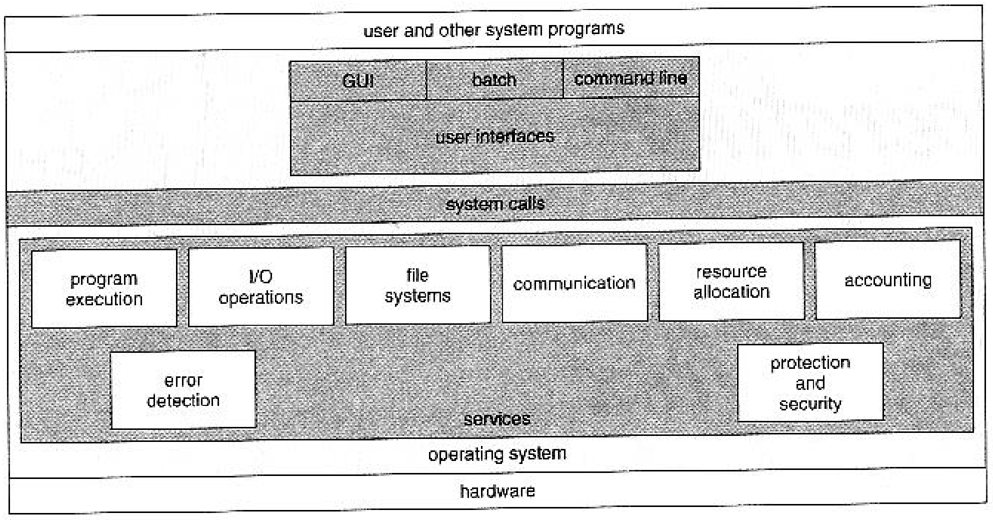


Figure 2.1 A view of operating system services.

An OS provides an environment for the execution of the programs. The common services provided by the OS are

1. **User interface:** Almost all operating systems have a user interface (UI).This interface can take several forms.
   a. **Command-line interface (CLI):** uses text commands and a specific method for entering them.
   b. **Batch Interface:** commands and directives to control are entered into files and those files are executed.
   c. **Graphical User Interface (GUI):** most common. Interface is a window system with a pointing device directing the I/O, choose from menus, make selections along with keyboard to enter text.
2. **Program Execution:** The OS must able to load the program into memory & run that program. The program must be able to end its execution either normally or abnormally.
3. **I/O Operation:** A running program may require I/O(file or an I/O device). Users cannot control the I/O devices directly. So the OS must provide a means for controlling I/O devices.
4. **File System manipulation**: Program needs to read and write files and directories. They also need to create and delete files, search for a given file and list file information. Some programs include **permission management** to deny access to files or directories based on file ownership.
5. **Communication**: In certain situation one process may need to exchange information with another process. This communication may takes place in two ways.
   i. Between the processes executing on the same computer.

ii. Between the processes executing on different computer that are connected by a network.

Communications can be implemented via **shared memory** or by **message passing**, in which packets of information are moved between processes by the OS.

6. **Error Detection**: Errors may occur in CPU, I/O devices or in Memory Hardware. The OS constantly needs to be aware of possible errors. For each type of errors the OS should take appropriate actions to ensure correct & consistent computing.
7. **Resource Allocation**: When multiple users logs onto the system or when multiple jobs are running, resources must be allocated to each of them. The OS manages different types of OS resources. Some resources may need some special allocation codes & others may have some general request & release code.
8. **Accounting:** We need to keep track of which users use how many & what kind of resources. This record keeping may be used for accounting. This accounting data may be used for statistics or billing. It can also be used to improve system efficiency.
9. **Protection and security**: Protection ensures that all the access to the system are controlled. Security starts with each user having authenticated to the system, usually by means of a password. External I/O devices must also be protected from invalid access. In multi process environment it is possible that one process may interface with the other or with the OS, so protection is required.

2.2 User Operating-System Interface

There are two fundamental approaches for users to interface with OS,

1. Command-line Interface
2. Graphical User Interface

- **Command Interpreter(CI)**

✓ Some OS include CI in the kernel, and in others like windows-XP and UNIX, it is treated as a special program that is running when a job is initiated or when a user first logs on.

✓ On systems with multiple command interpreters to choose from, the interpreters are known as **shells. For ex:** On UNIX and Linux systems, there are different shells a user may choose from including Bourne shell, C shell and K or n shell etc

✓ The main function of the command interpreter is to get and execute the next user-specified command.

✓ Commands are implemented in two ways:

1. In one approach, the command interpreter itself has the code to execute the command. Ex. a command to delete a file. This will result in the command interpreter to go to a section of its code that sets up the parameters and makes the appropriate system call. In this method, the size of the command interpreter depends on the number of commands that can be given.

2. Alternative approach used by UNIX is most commands are implemented through system programs. The command interpreter uses the command to identify a file to be loaded into memory and executed. Ex. **rmfile.txt** would make the command interpreter search for a **file rm**, load that file into memory and execute it with parameter **file.txt**.

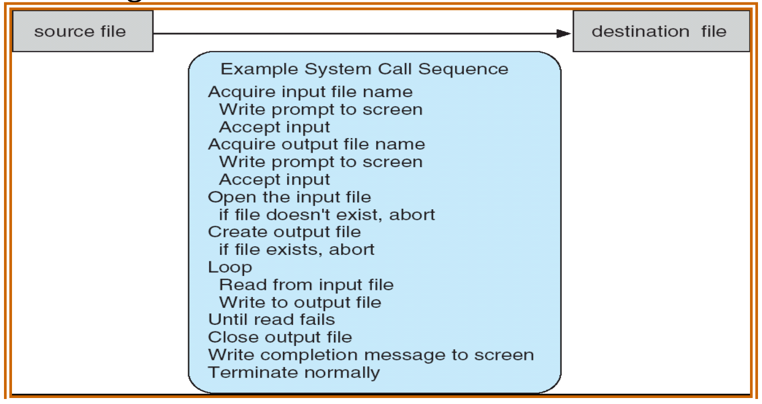
- ✓ Advantages of CIs are,
  - o Command interpreter program is small.
  - o Command interpreter does not have to be changed when new commands are added.
  - o New commands can be easily added to the system.

- **Graphical User Interface**

  - ✓ GUI provides user-friendly **desktop** metaphor interface where the mouse is moved to position where images or icons on the desktop that represent programs, files directories and other system functions.
  - ✓ Depending on mouse pointer's location, clicking the mouse button can invoke the corresponding program, select a file or directory known as folders or pull down a menu that contains commands.
  - ✓ First appeared in 1970s as a part of research at **Xerox Parc research facility**.
  - ✓ It became widespread with the coming of **Apple Macintosh** in 1980s.
  - ✓ Microsoft's first version of Windows was based on GUI interface for MS-DOS. The various windows systems that have been appeared and had enhancements in the GUI.
  - ✓ UNIX later implemented GUI in CDE (Common Desktop Environment) and X-Windows Systems. Also seen in Solaris and IBM's AIX system.

2.3 SYSTEM CALLS

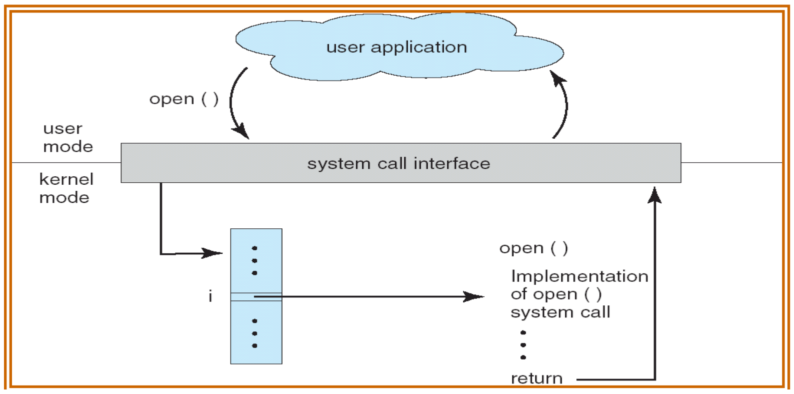
- ✓ System provides interface to the services made available by an OS.
- ✓ These calls are generally available as routines written in C and C++, although certain low-level tasks may need to be written using assembly language instruction.
- ✓ System call sequence to read the contents of one file and copy to another file is illustrated in below **figure** shown below



```
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
```

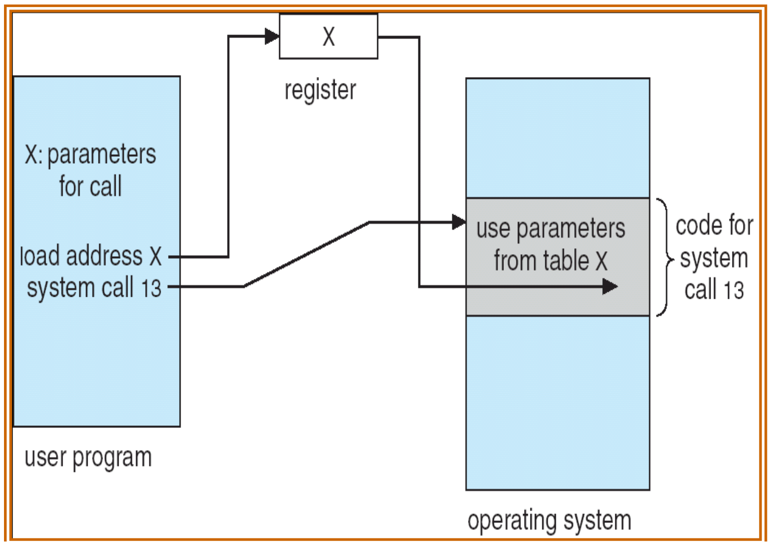
- The first input that the program will need is the names of two files which can be specified in many ways. This sequence requires many I/O system calls.
- Next, the program must open the input file which requires another system call. If opening of file fails, it should display error message on console (another system call) and should terminate abnormally (another system call).
- Next, the program must create the output file (another system call), If fails, it should display error message on console (another system call) and should also abort (another system call).
- Next, we enter a loop that reads from input file (system call) and writes to the output file (system call).Write/read operation may fail, which needs another system call to continue.
- Finally, after the entire file is copied, the program may close both files (system call), write message to console (system call)), and terminate normally (system call).

- ✓ Application developers design programs according to an **Application Program Interface (API)**. API specifies the set of functions that are available to an application programmer including the parameter that are passed to each function and returns values the programmer can expect.
- ✓ Three most common APIs are **Win32** API for Windows, **POSIX** API for POSIX-based systems (UNIX, Linux, and Mac OS X), and **Java** API for the Java virtual machine (JVM).
- ✓ The runtime support system(a set of functions built into libraries included with a compiler) for most programming languages provides a **system call interface** that serves as the link to system calls made available by the OS.
- ✓ The system call interface intercepts function call in the API and invokes the necessary system call within the OS.
- ✓ A **number** is associated with each system call and the **system-call interface** maintains a **table** indexed according to these numbers.
- ✓ The **system call interface** invokes intended system call in OS kernel and returns status of the system call and any return values.
- ✓ The caller needs to know nothing about how the system call is implemented or what it does during execution.
- ✓ The below **figure** illustrates how the OS handles a user application which is invoking **open ()** system call.



Three general methods are used to pass the parameters to the OS.

1. The simplest approach is to pass the parameters in registers.
2. In some cases there can be more parameters than registers. In these cases the parameters are stored in a **block or table** in memory and the address of the block is passed as a parameter in register. It is shown in below **figure** This approach is used by Linux and Solaris.



3. Parameters can also be placed or **pushed** onto **stack** by the program &**popped** off the stack by the OS.

Some OS prefer the **block or stack** methods, because those approaches do not limit the number or length of parameters being passed.

2.4 Types of System calls

System calls may be grouped roughly into **5 categories**

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

1. Process control

- **end, abort**

  ✓ A running program needs to be able to halt its execution either **normally** or **abnormally**. In an abnormal termination a dump of memory is taken and an error message is generated. Dump is written to disk and examined by the debugger to determine the cause of problem.

- ✓ In normal or abnormal situations the OS must transfer the control to the command interpreter system, which then reads the next command given by the user.
- ✓ In batch system the command interpreter terminates the execution of job & continues with the next job. Batch-systems uses **control cards** to indicate the special recovery action to be taken in case of errors. It is a command to manage the execution of a process.
- ✓ More severe errors can be indicated by a higher level error parameter. Normal & abnormal termination can be combined by defining normal termination as an error at **level 0**.The command interpreter uses this error level to determine next action automatically.

- **load, execute**

  - ✓ A process executing one program may want to **load and execute** another program. This feature allows the command interpreter to execute programs as directed by the user.
  - ✓ The question of where to return the control when the loaded program terminates is related to the problem of whether the existing program is lost, saved or allowed to continue execution concurrently with the new program. There is a system call for this purpose (**create or submit process**).

- **create process, terminate process**

  - ✓ If we create a new job or process, it should be able to control its execution.
  - ✓ This control requires the ability to determine and reset the attributes of a process, including the process priority, its maximum allowable execution time, and so on (get process attributes and set process attributes).
  - ✓ We may also want to terminate a process that we created (terminate process)

- **wait event, signal event**

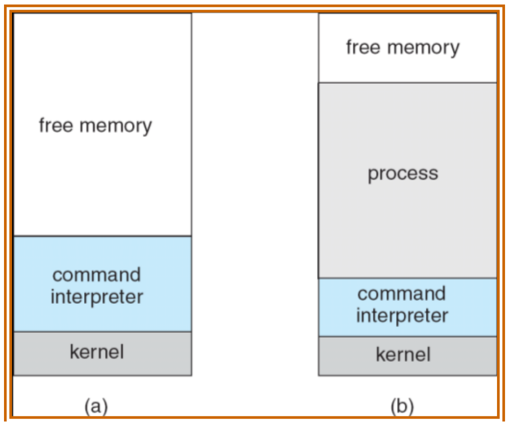
  - ✓ When new jobs have been created, we may want to wait for certain amount of time using **wait time** system call.
  - ✓ When a job has to wait for a certain event to occur **wait event** system call is used.
  - ✓ When the event has occurred the job should signal the occurrence through the **signal event** system call.

- **get process attributes, set process attributes**
- **wait for time**
- **allocate and free memory**

  - o **In MS-DOS**

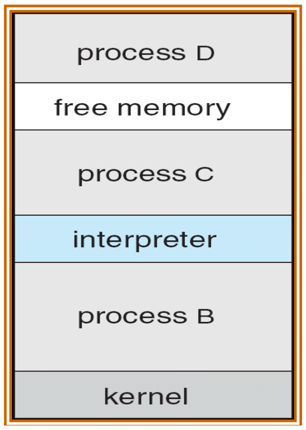
  - ✓ MS-DOS is an example of single tasking system, which has command interpreter system that is invoked when the computer is started as shown in **figure (a)**.

- ✓ To run a program MS-DOS uses simple method. It does not create a newprocess when one process is running.
- ✓ It loads the program into memory and gives the program as much memory as possible as shown in **figure (b)**.



- **In FreeBSD**

- ✓ Free BSD is an example of multitasking system.
- ✓ In free BSD the command interpreter may continue running while other program is executed as shown in below **figure**.
- ✓ fork () is a system call used to create new process.
- ✓ Then, the selected program is loaded into memory via an exec () system call and then program is executed.



2. **File management**

- create file, delete file
  - ✓ System calls can be used to create & delete files. System calls may require the name of the files and attributes for creating & deleting of files.
- open, close file
  - ✓ Opens the file for usage and finally we need to close the file.
- read, write, reposition
  - ✓ Other operation may involve the reading of the file, write & reposition the file after it is opened.

- get and set file attributes
  - ✓ For directories, some set of operation are to be performed. Sometimes it is required to reset some of the attributes on files & directories. The system call **get file attribute** & **set file attribute** are used for this type of operation.

3. Device management

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

  - ✓ The system calls are also used for accessing devices.
  - ✓ Many of the system calls used for files are also used for devices.
  - ✓ A system with multiple users may require us to first request the device, to ensure exclusive use of it.
  - ✓ After using the device, it must be released using **release** system call. These functions are similar to open & close system calls of files.
  - ✓ Read, write & reposition system calls may be used with devices.
  - ✓ MS-DOS & UNIX merge the I/O devices & the files to form **file-device structure**.

4. Information maintenance

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

  - ✓ Many system calls exist for the purpose of transferring information between the user program and the operating system.
  - ✓ For example, most systems have a system call to return the current **time** and **date**.
  - ✓ Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.
  - ✓ The operating system also keeps information about all its processes, and system calls are used to access this information.
  - ✓ System calls are also used to reset the process information (**get process attributes and set process attributes**).

5. Communications

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

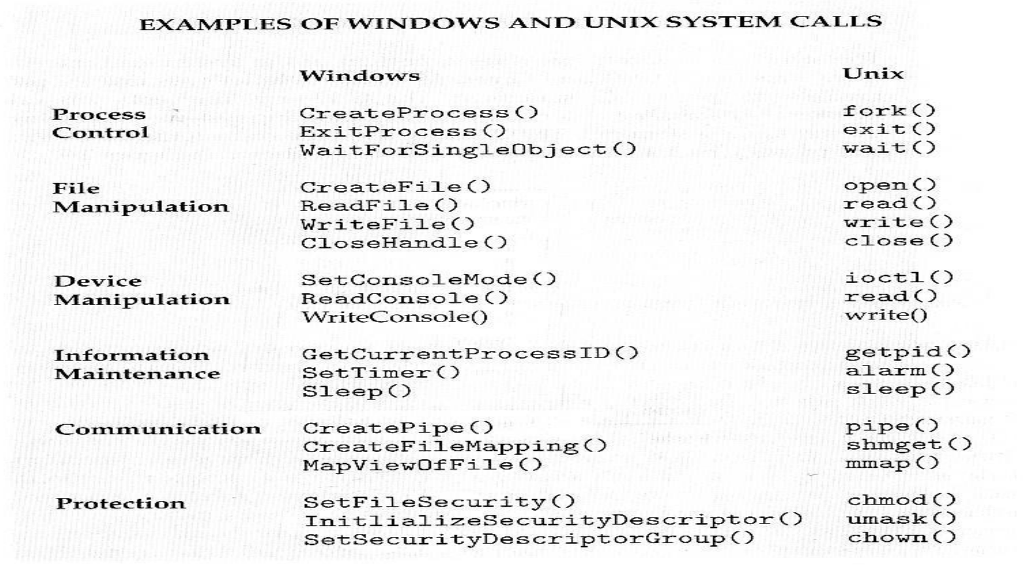
There are two models of inter-process communication:

❖ **Message Passing model**

- ✓ In message passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common **mailbox**.
- ✓ Each computer in a network will have a **host name.** Similarly, each process has a **process name**, and this name is translated into an identifier by which the operating system can refer to the process. The **get hostid** and **get processid** system calls do this translation.
- ✓ The identifiers are then passed to the general purpose **open** and **close** calls provided by the file system or to specific **open connection** and **close connection** system calls, depending on the system's model of communication.
- ✓ The recipient process must give its permission for communication to take place with an **accept connection** system call.
- ✓ The receiving daemons execute a **wait for connection** call and are awakened when a connection is made.
- ✓ The source of the communication, known as the **client**, and the receiving daemon, known as a **server**, exchange messages by using **read message** and **write message** system calls.
- ✓ The **close connection** call terminates the communication.

❖ **Shared Memory model**

- ✓ In shared memory model, processes use **shared memory create** and **shared memory attach** system calls to create and gain access to regions of memory owned by other processes.
- ✓ The OS tries to prevent one process from accessing another process’s memory, so several processes have to agree to remove this restriction. Then they exchange information by reading and writing in the shared areas.
- ✓ The processes are also responsible for ensuring that they are not writing to the same location simultaneously.



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()

2.5 System Programs

- ✓ **System programs**, also known as system utilities, provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls and others are considerably more complex. They can be divided into these **categories**:

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

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

2.6 OS Design & Implementation

- **Design goals**
  - ✓ The first aspect in designing a system is defining goals and specifications. Next we need to define the mechanisms and policies to be implemented. Finally the implementation takes place.
  - ✓ At the highest level, the system design will be affected by the choice of hardware and type of system like timesharing, batch, distributed, real time, single user, multiuser OS or general purpose etc.
  - ✓ At the next level the requirements can be divided into two basic groups: **user** goals and **system** goals.

- ✓ The **user goals** basically comprises of convenient to use, easy to learn and to use, reliable, safe and fast etc.
- ✓ The **system goals** are from the designer's perspective that the system must be easy to design, create and maintain. It should also be flexible, reliable, error free and efficient.
- ✓ The requirements vary from system to system. Different requirements result in different solutions and hence different Operating Systems.

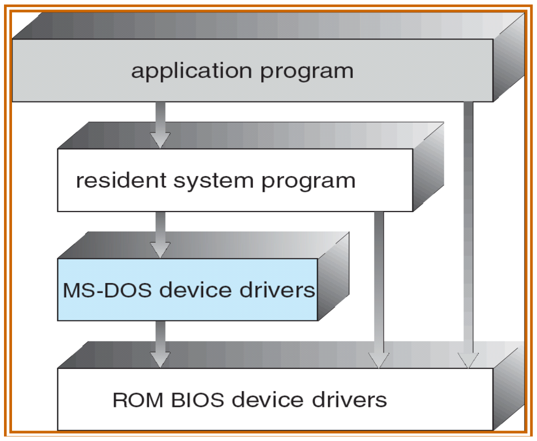
- **Mechanisms and policies**
  - ✓ Mechanisms determine **how** to do something. Mechanisms that are insensitive to changes in policy are more desirable.
  - ✓ Policy determines **what** will be done. Policies are likely to change across places and over time.
  - ✓ Change in policy may require redefinition of certain parameters of the system.
  - ✓ Policy decisions are important for all resource allocation.
    - o **Ex**. **timer construct** is a **mechanism** to ensure CPU protection, whereas for **how long** the timer is to be set for a particular user is a **policy** decision.
- **Implementation**
  - ✓ Once an operating system is designed, it must be implemented.
  - ✓ Traditionally, operating systems have been written in **assembly language**.
  - ✓ MS-DOS was initially implemented in Intel 8088 and was available on Intel CPUs only. Master Control Program (MCP) written in ALGOL, MULTICS in PL/I, and Linux is with C and available on Intel 8086, Motorola 680, SPARC and MIPS RX000.
  - ✓ Now, they are most commonly written in **higher-level languages** such as **C or C++.**
  - ✓ The **advantages** of using higher-level languages are, the code can be written faster, it is more compact, and is easier to understand and debug.
  - ✓ The improvements in compiler technology will improve the generated code for the entire operating system by simple recompilation.
  - ✓ Finally, an operating system is easier to **por**t (to move to some other hardware) if it is written in a higher-level language.
  - ✓ The only possible **disadvantages** of implementing an operating system in a higher-level language are **reduced speed and increased storage requirements.**

2.7 Operating System Structures

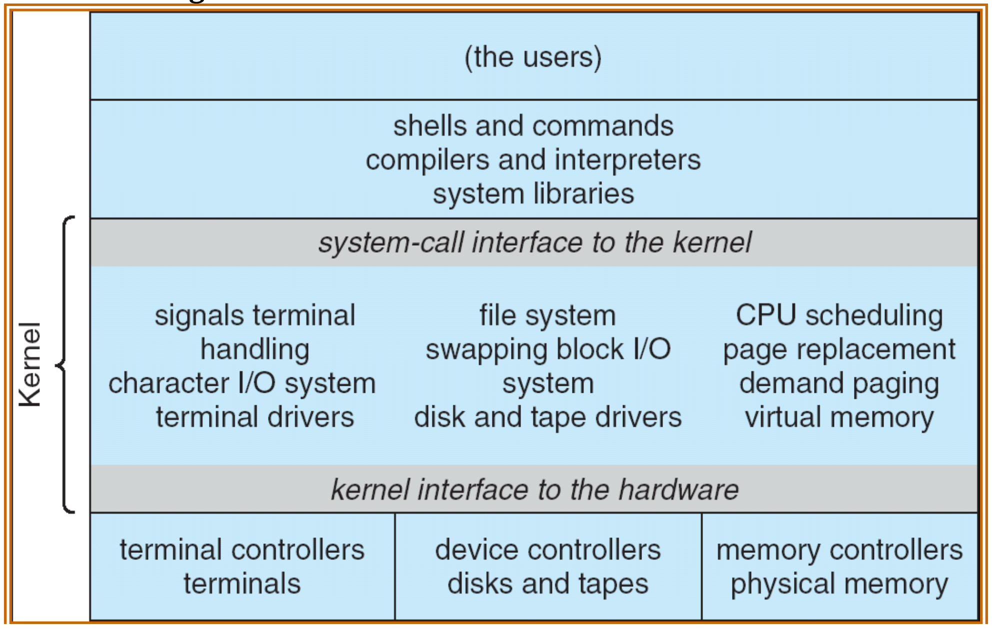
- ✓ Modern OS is large & complex. It consists of different types of components. These components are interconnected & melded into kernel.
- ✓ For designing the system, different types of structures are used. They are,
  - Simple structures.
  - Layered Approach.
  - Micro kernels.

- **Simple Structures**

  - ✓ Simple structure OS are small, simple & limited systems. The structure is not well defined.
  - ✓ MS-DOS is an example of simple structure OS.MS-DOS layer structure is shown in below **figure**.



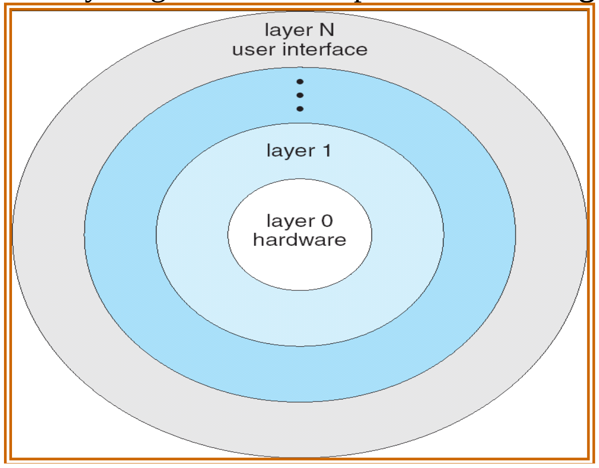
  - ✓ In MS-DOS, the interfaces and levels of functionality are not well separated.
  - ✓ For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives.
  - ✓ **UNIX** is another example for simple structure. Initially it was limited by hardware functions.
    - o It consists of **two** separable parts: the kernel and the system programs.
    - o The kernel is further separated into series of interfaces & device drivers.
    - o We can view the traditional UNIX operating system as being layered, as shown in below **figure**.



- ✓ Everything below the system-call interface and above the physical hardware is the kernel.
- ✓ Kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.
- ✓ This **monolithic** structure was difficult to implement and maintain.

- **Layered Approach**

  - ✓ A system can be made modular in many ways. One method is the **layered approach** in which the OS is divided into number of layers, where one layer is built on the top of another layer.
  - ✓ The bottom layer (layer 0) is **hardware** and higher layer (layer N) is the **user interface**. This layering structure is depicted in below **figure** .



  - ✓ An OS is an implementation of **abstract object** made up of data & operations that manipulate these data.
  - ✓ A typical operating-system layer say layer *M* consists of data structures and a set of routines that can be invoked by higher-level layers. Layer *M*, in turn can invoke operations on lower level layers.
  - ✓ The main **advantage** of layered approach is the **simplicity** i.e. each layer uses the services & functions provided by the lower layer. This approach simplifies the debugging & verification. Once first layer is debugged the correct functionality is guaranteed while debugging the second layer. If an error is identified then it is a problem in that layer because the layer below is already debugged.
  - ✓ Each layer tries to hide some data structures, operations & hardware from the higher level layers.
  - ✓ A problem with layered implementation is that they are less efficient.

- **Micro Kernels**

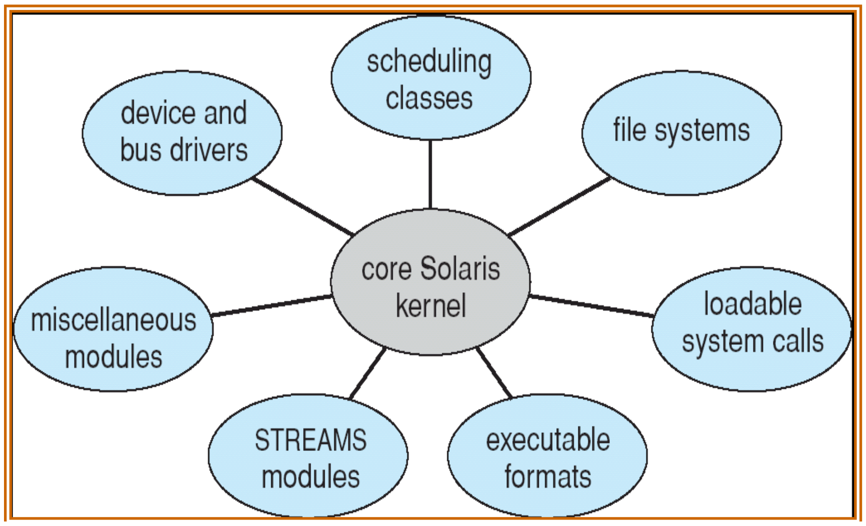
  - ✓ In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the operating system by removing all nonessential components from the kernel and implementing them as system and user level programs. The result is a smaller kernel.

- ✓ The main function of the micro kernels is to provide **communication facilities** between the client program and various services that are running in user space.
- ✓ This approach provided a high degree of flexibility and modularity.
- ✓ It includes the ease of extending OS. All the new services are added to the user space & do not need the modification of kernel.
- ✓ This approach also provides more **security** & **reliability**.
- ✓ Most of the services will be running as user process rather than the kernel process.
- ✓ A micro kernel in Windows NT provides **portability** and **modularity**.

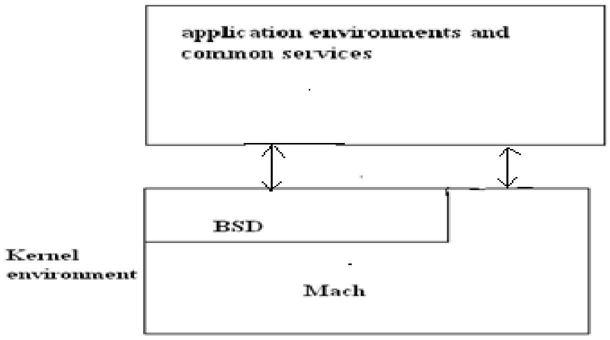
- **Modules**

- ✓ The best current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel.
- ✓ Here, the kernel has a set of core components and links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X.
- ✓ For example, the Solaris operating system structure, shown in the below **figure 2**, is organized around a core kernel with **seven types** of loadable kernel modules:

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



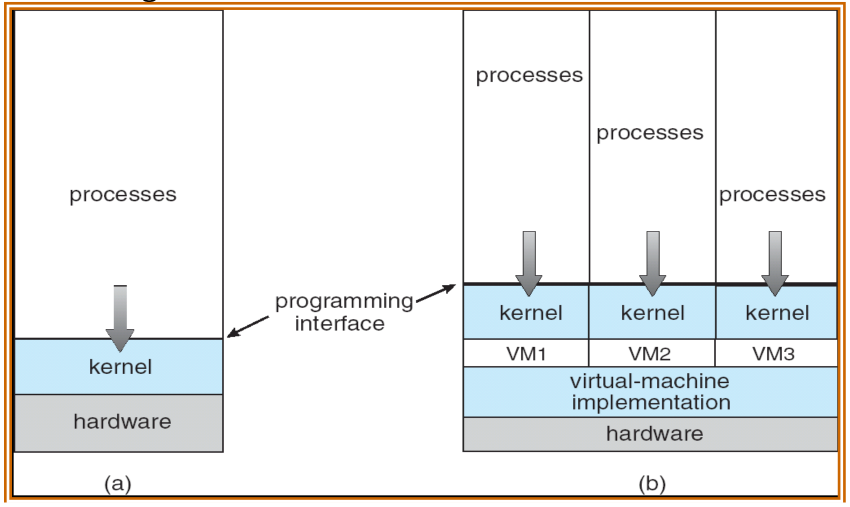
- ✓ The Apple Macintosh Mac OS X operating system uses a hybrid structure. It is a layered system in which one layer consists of the Mach microkernel. The structure of Mac OS X appears as shown in below **figure.**



- ✓ The top layers include application environments and a set of services providing a graphical interface to applications.
- ✓ Below these layers is the kernel environment, which consists primarily of **the Mach microkernel and the BSD kernel.**
- ✓ Mach provides memory management, support for remote procedure calls (RPCs) and interprocess communication facilities, including message passing and thread scheduling.
- ✓ The BSD component provides a BSD command line interface, support for networking and file systems, and an implementation of POSIX APIs, including Pthreads.

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 on) into several different **execution environments**, thereby creating the illusion that each separate execution environment is running its own **private computer**.
- ✓ By using CPU scheduling and virtual-memory techniques, an operating system can create the illusion that a process has its own processor with its own (virtual) memory.
- ✓ Each process is provided with a (virtual) copy of the underlying computer as shown in the below **figure.**



(a) Non virtual machine (b) virtual machine

- ✓ A major difficulty with the virtual machine approach involves disk systems. Suppose that the physical machine had three disk drives but wanted to support seven virtual machines. Clearly, it could not allocate a disk drive to each virtual machine, because the virtual machine software itself will need substantial disk space to provide virtual memory and spooling. The solution is to provide virtual disks-termed **mini disks** in IBM's VM operating system which are identical in all respects except size.

- **Implementation**

  - ✓ It is difficult to implement VM concept. Much work is required to provide an **exact** duplicate of the underlying machine.
  - ✓ The machine typically has two modes: **user mode and kernel mode**.
  - ✓ The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute in only user mode.
  - ✓ The major difference between virtual and non virtual m/c is **time**. The real I/O might have taken 100 milliseconds, the virtual I/O might take less time (because it is spooled) or more time (because it is interpreted). In addition, the CPU is being multi programmed among many virtual machines, further slowing down the virtual machines in unpredictable ways.

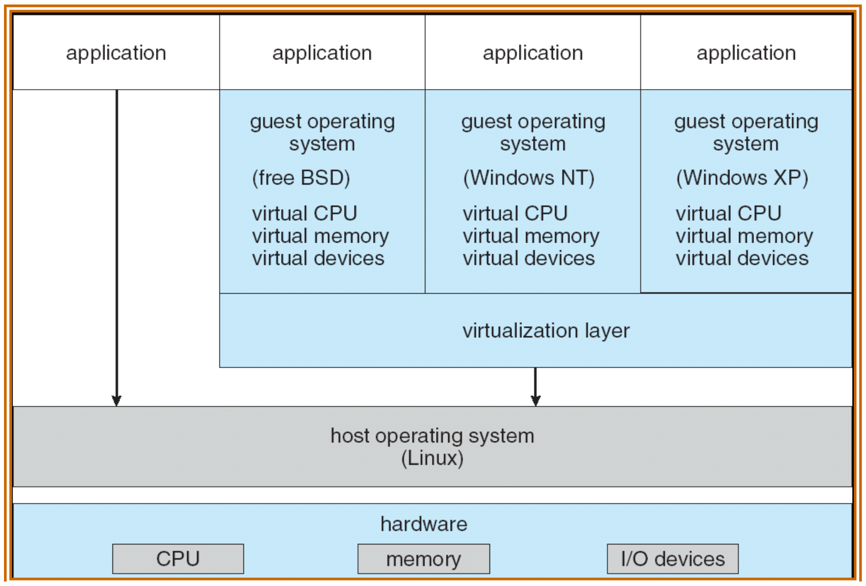
- **Benefits**

  - ✓ The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation permits no direct sharing of resources.
  - ✓ A virtual-machine system is a perfect vehicle for operating-systems research and development.
  - ✓ System programmers are given their own VM, and system development is done on the virtual machine instead on a physical machine. Thus changing OS will not cause any problem.

Examples

1. **VMware**

   - ✓ It is a popular commercial application that abstracts Intel X86 and compatible hardware into isolated virtual machines.
   - ✓ It 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.
   - ✓ The architecture is shown below **figure**.

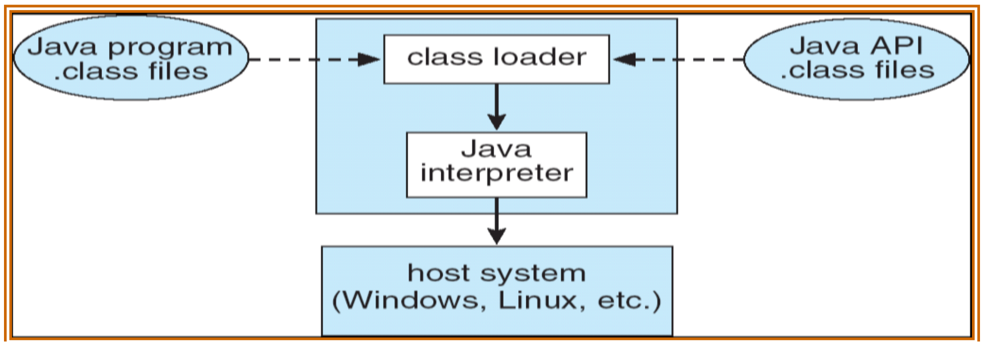


VMware architecture

- ✓ Here, Linux is running as the host operating system and FreeBSD, Windows NT, and Windows XPare 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 on.

2. Java virtual machine

- ✓ Java is a popular object-oriented programming language introduced by Sun Microsystems in 1995.
- ✓ In addition to a language specification and a large API library, Java also provides a specification for a Java virtual machine-or JVM.
- ✓ Java objects are specified with the **class** construct. A Java program consists of one or more classes. For each Java class, the compiler produces an **architecture-neutral byte code output (.class) file** that will run on any implementation of the JVM.
- ✓ The JVM is a specification for an abstract computer. It consists of a class loader and a Java interpreter that executes the architecture-neutral byte codes, as given in below **figure**.



The Java virtual machine

- ✓ The class loader loads the compiled **.class** files from both the Java program and the Java API for execution by the Java interpreter.
- ✓ After a class is loaded, the verifier checks that the **.class** file is valid Java byte code and does not overflow or underflow the stack. It also ensures that the byte code does not perform pointer arithmetic, which could provide illegal memory access.
- ✓ If the class passes verification, it is run by the Java interpreter.
- ✓ The JVM also automatically manages memory by performing **garbage collection** -the practice of reclaiming memory from objects no longer in use and returning it to the system.
- ✓ The JVM may be implemented in software on top of a host operating system, such as Windows, Linux, or Mac OS X, or as part of a Web browser.

2.11 System Boot

- ✓ The procedure of starting a computer by loading the kernel is known as **booting** the system.
- ✓ **Bootstrap program or Bootstrap loader** locates the kernel, loads it into main memory and start its execution.
- ✓ Bootstrap program is in the form of **read only memory (ROM)** because the RAM is in unknown state at a system startup. All forms of ROM are knows as **firmware**.
- ✓ For large OS like Windows, Mac OS, the Bootstrap loaders is stored in firmware and the OS is on disk.
- ✓ Bootstrap has a bit code to read a single block at a fixed location from disk into the memory and execute the code from that **boot block**.
- ✓ A disk that has a boot partition is called a **boot disk or system disk**.

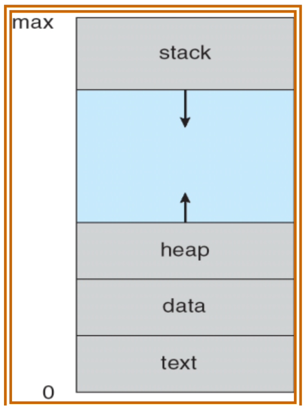
CHAPTER 3 PROCESS CONCEPTS

3.1 Process Concepts

- ✓ Process is an **active** entity. A process is a sequence of instruction execution. Process exists in a limited span of time. Two or more process may execute the same program by using its own data & resources.
- ✓ A program is a **passive** entity which is made up of program statement. Program contains instructions.

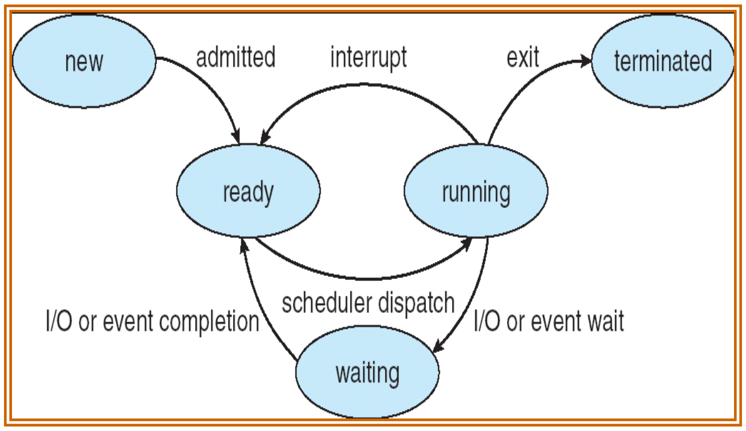
- **The Process**

  - ✓ A process is more than the program code which is also called **text section**.
  - ✓ It contains **program counter** which represents the current activity and also the contents of the processor's registers.
  - ✓ A process also consists of a process **stack section** which contains temporary data &**data section** which contains global variables.
  - ✓ A process may also include a **heap**, which is memory that is dynamically allocated during process run time.
  - ✓ The structure of a process in memory is shown in below **figure.**



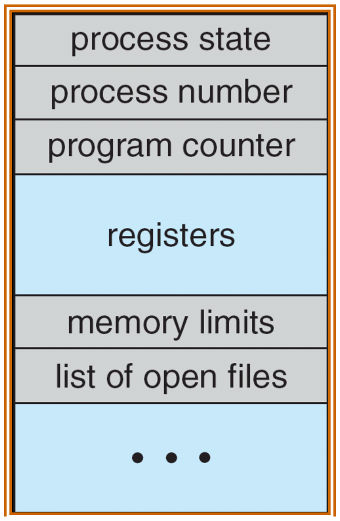
- **Process State**

  - ✓ As process executes it changes its state and each process may be in one of the following states:
    - o **New:** The process is being created
    - o **Running:** Instructions are being executed
    - o **Waiting:** The process is waiting for some event to occur
    - o **Ready:** The process is waiting to be assigned to a process
    - o **Terminated:** The process has finished execution
  - ✓ Only one process can be running on any processor at any instant. Many processes may be ready and waiting.
  - ✓ The **state diagram** corresponding to these states is shown below **figure.**



- **Process Control Block**

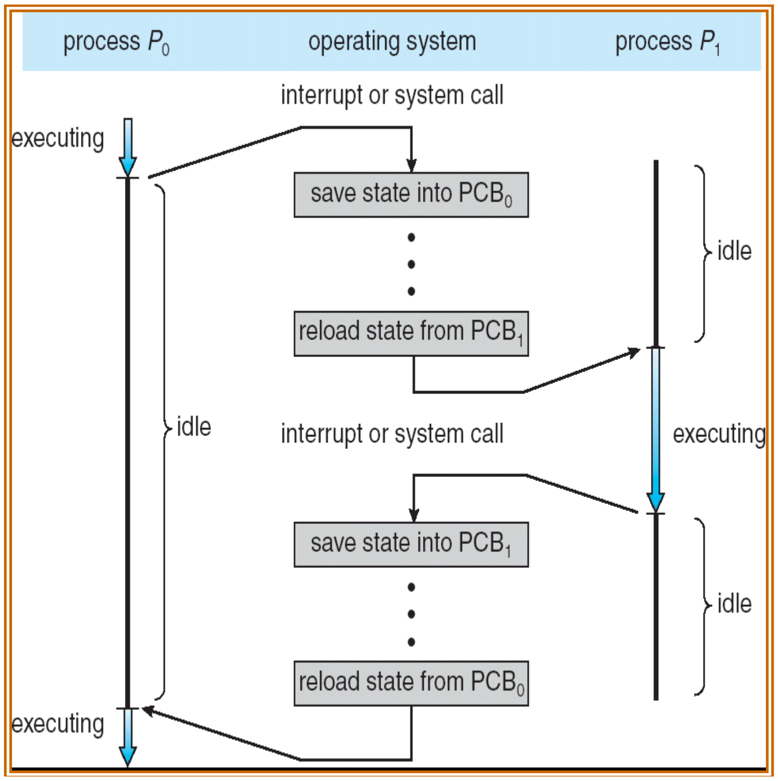
  ✓ A process in an operating system is represented by a **data structure** known as a **Process Control Block (PCB)** and it is also called as **task control block**. The following **figure** shows the process control block.



  ✓ The PCB contains important **information** about the specific process including,

  - **Process state:** The current state of the process i.e., whether it is ready, running, waiting, halted and so on.
  - **Program counter:** Indicates the address of the next instruction to be executed for a process.
  - **CPU registers:** The registers vary in **number** and **type**. Along with program counter this state information should be saved to allow process to be continued correctly after an interrupt occurs as shown in below **figure.**
  - **CPU scheduling information:** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

- **Memory-management information:** This information may include the value of **base** and **limit** registers, the page tables, or the segment tables, depending on the memory system used by the OS.
- **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.



CPU switch from process to process

- **Threads**

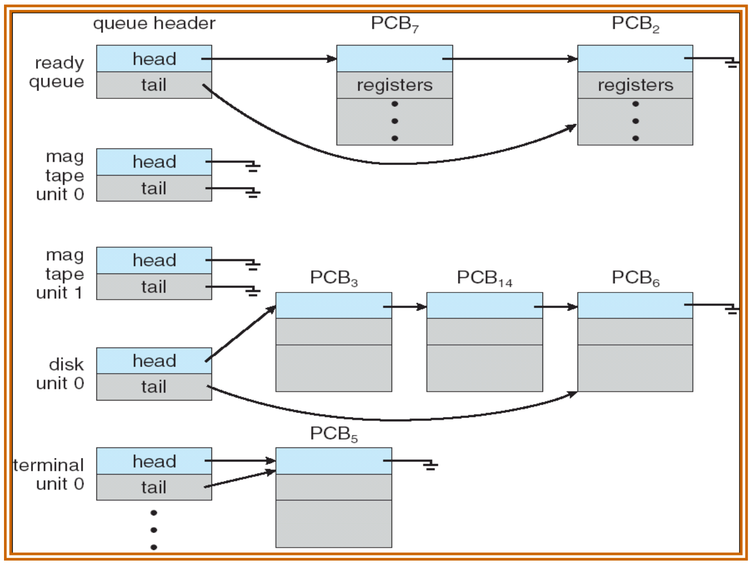
✓ A process is a program that performs a single **thread** of execution. **For example**, when a process is running a word-processor program, a single thread of instruction is being executed. This single thread of control allows the process to perform only one task at one time. The user cannot simultaneously type in characters and run the spell checker within the same process.

✓ Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time. On a system that supports threads, the PCB is expanded to include information for each thread. Eg: Windows OS and UNIX

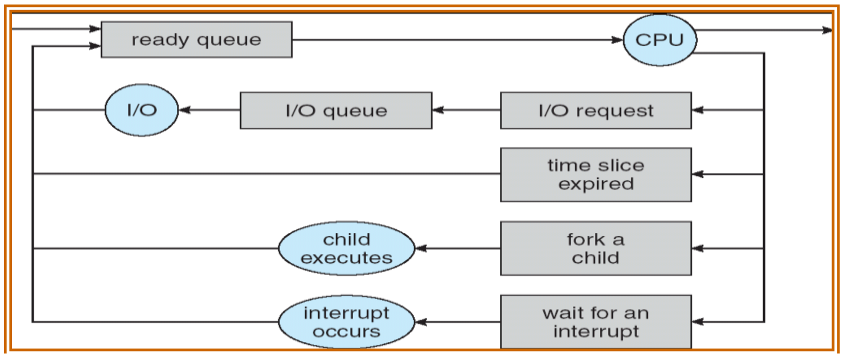
3.2 Process Scheduling

The **process scheduler** selects an available process for execution on the CPU.

- **Scheduling queues**
  - ✓ The following are the different types of process scheduling queues.
    - **Job queue:** set of all processes in the system.
    - **Ready queue**: The processes that are placed in main memory and are **ready and waiting** to execute are placed in a list called the ready queue. This is in the form of linked list, the header contains pointer to the first and final PCB in the list. Each PCB contains a pointer field that points to next PCB in ready queue.
    - **Device queue**: The list of processes waiting for a particular **I/O device** is called device queue. When the CPU is allocated to a process it may execute for some time and may quit or interrupted or wait for the occurrence of a particular event like completion of an I/O request, but the I/O may be busy with some other processes. In this case the process must wait for I/O and it will be placed in device queue. Each device will have its own queue.
  - ✓ The below **figure.** Shows Ready queue and various I/O Device queues



  - ✓ The process scheduling is represented using a **queuing diagram** as shown in below **figure. Queues** are represented by the **rectangular box** and **resources** they need are represented by **circles**, and the **arrows** indicate the **flow of processes** in the system.

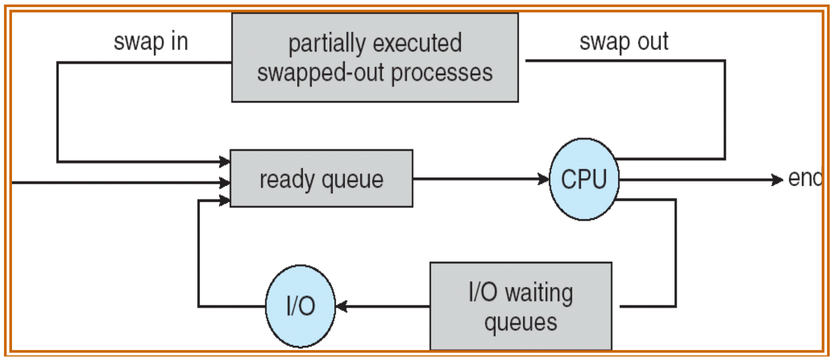


- ✓ A new process is initially put in the ready queue and it waits there until it is selected for execution or **dispatched**. Once the process is assigned CPU and is executing, the following **events** can occur,
  - It can execute an I/O request and is placed in I/O queue.
  - The process can create a sub process & wait for its termination.
  - The process may be removed from the CPU as a result of interrupt and can be put back into ready queue.

- **Schedulers**

- ✓ The following are the **different types** of schedulers,

  - **Long-term scheduler (or job scheduler):** selects which processes should be brought into the ready queue. Long-term scheduler is invoked very infrequently (in terms of seconds or minutes). The long-term scheduler controls the **degree of multiprogramming** (number of processes in main memory)**.**
  - **Short-term scheduler (or CPU scheduler):** selects which process should be executed next and allocates CPU. Short-term scheduler is invoked very frequently**.**
  - **Medium-term schedulers:** Some OS introduces intermediate level of scheduling called Medium-term schedulers as shown in below **figure**. It can be advantageous to remove processes from memory and thus reduce the degree of multiprogramming. The process can be later reintroduced into memory, and its execution can be continued where it left off. The process is **swapped out**, and is later **swapped in**, by the **medium-term scheduler**.



- ✓ Processes can be described as either:
  - o **I/O-bound process**: processes spend more time doing I/O than computations.
  - o **CPU-bound process**: processes spend more time doing computations; and generates I/O request less frequently.

- **Context Switch**

  - ✓ When an **interrupt** occurs, the system needs to save the current **context** of the process running on the CPU. The context is represented in the **PCB** of the process.
  - ✓ When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process. A **state save** of the current state of the CPU, and then **state restore** to resume operations is performed.
  - ✓ Context-switch time is overhead and the system does no useful work while switching. Context-switch times are highly dependent on hardware support. Context-switch speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions.

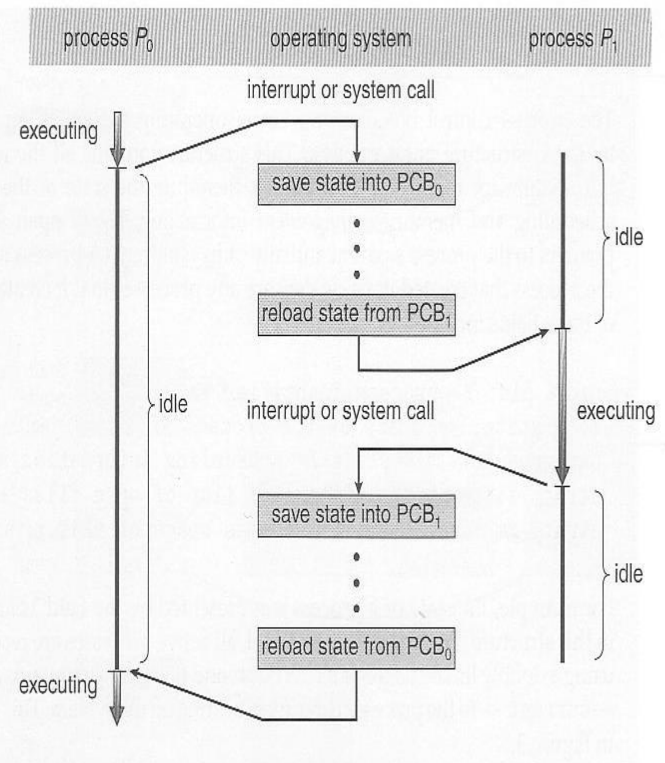


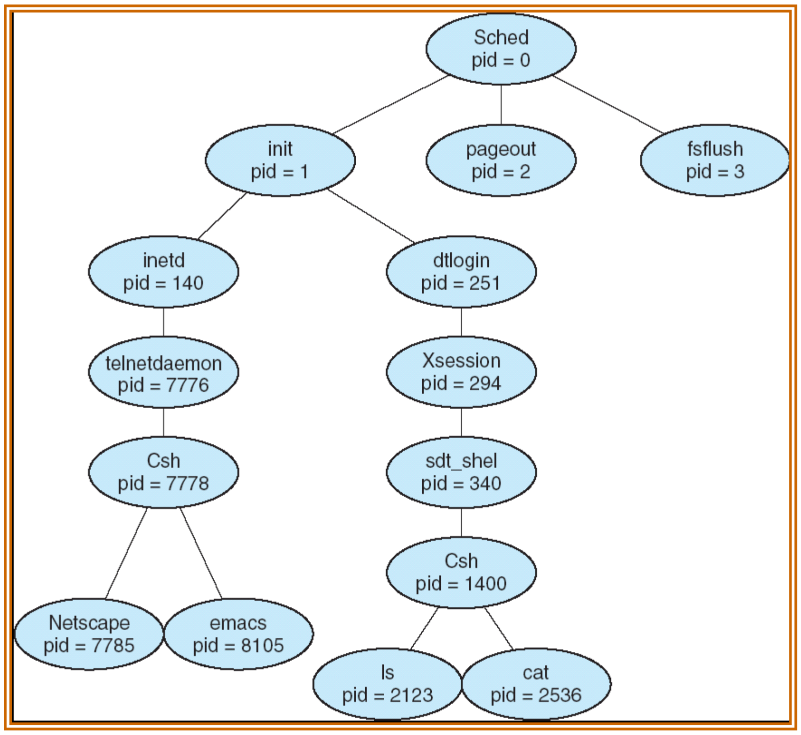
Figure 3.4 Diagram showing CPU switch from process to process.

3.3 Process Operations

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination.

- **Process Creation**

  - ✓ A process may create several new processes by some **create-process** system call, during the course of execution.
  - ✓ The creating process is called **parent** process and the created one is called the **child** process. Each of the new process may in turn create other processes, forming a **tree** of processes. Processes are identified by unique **process identifier (** or **pid).**
  - ✓ Below **Figure**. shows the process tree for the solaris OS. The process at the top of the tree is **sched** process, with pid of 0,and this creates several children processes. The sched process creates several children processes including **pageout and fsflush**. These processes are responsible for managing memory and file systems. The sched process also creates the **init** process, which serves as the root parent process for all user processes. These processes are responsible for managing memory and file systems.
  - ✓ inetd and dtlogin are two children of init where inetd is responsible for networking services such as telnet and ftp; dtlogin is the process representing a user login screen.
  - ✓ When a user logs in, dtlogin creates an X-windows session (Xsession), which in turns creates the sdt_shel process. Below sdt_shel, a user's command-line shell, the C-shell or csh is created. In this command line interface, the user can then invoke various child processes, such as the ls and cat commands.
  - ✓ There is also csh process with pid of 7778 representing a user who has logged onto the system using telnet. This user has started the Netscape browser (pid of 7785) and the emacs editor (pid of 8105).
  - ✓ A process needs certain resources to accomplish its task. Along with the various logical and physical resources that a process obtains when it is created, **initialization data** may be passed along by the parent process to the child process.
  - ✓ When a process creates a new process, **two possibilities** exist in terms of **execution.**
    1. The parent continues to execute concurrently with its children.
    2. The parent waits until some or all of the children have terminated.
  - ✓ There are also **two possibilities** in terms of the **address space** of the new process.
    1. The child process is a duplicate of the parent process.
    2. The child process has a new program loaded into it.
  - ✓ In UNIX OS, **fork ()** system call creates new process. In windows Create Process() does the job.
  - ✓ **Exec ()** system call is called after a **fork ()** to replace the process memory space with a new program.

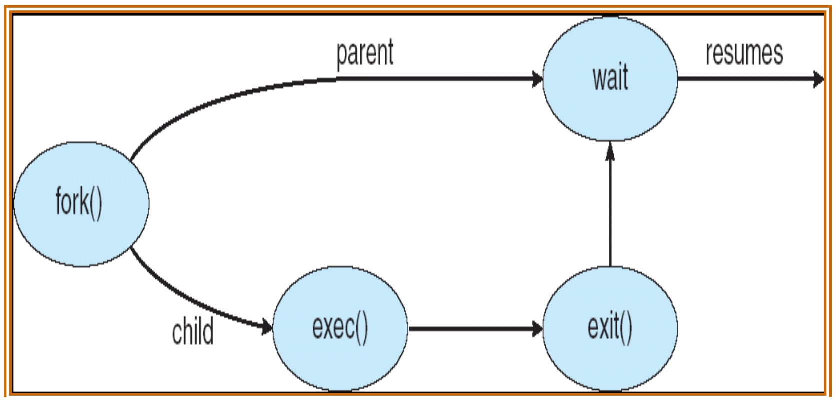


✓ The **C program** shown below illustrates these system calls.

```
            int main()
            {
            Pid_t  pid;
            pid = fork();/* fork another process */

            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 */
                {
                wait (NULL);/* parent will wait for the child to complete */
             printf ("Child Complete");
                exit(0);
            }
      }
```

- ✓ If there are two different processes running a copy of the same program, the pid for child is **zero** and for the parent it is **greater than zero**. The parent process waits for the child process to complete with the **wait ()** system call.
- ✓ When the child process completes, the parent process resumes from the call to wait(), where it completes using **exit()** system call. This is shown in below **figure.**



- **Process Termination**

- ✓ A process terminates when it finishes executing its last statement and asks the operating system to delete it by using **exit()** system call.
- ✓ Process resources are deallocated by the operating system. A process can terminate another process via **Terminate Process()** system call. A Parent may terminate execution of children processes (**abort**) for the following **reasons**.
  - Child has exceeded usage of allocated resources.
  - Task assigned to child is no longer required.
  - If parent is exiting some operating system do not allow child to continue if its parent terminates.
- ✓ Some systems does not allow child to exist if its parent has terminated. If process terminates then all its children must also be terminated, this phenomenon is referred as **cascading termination**.

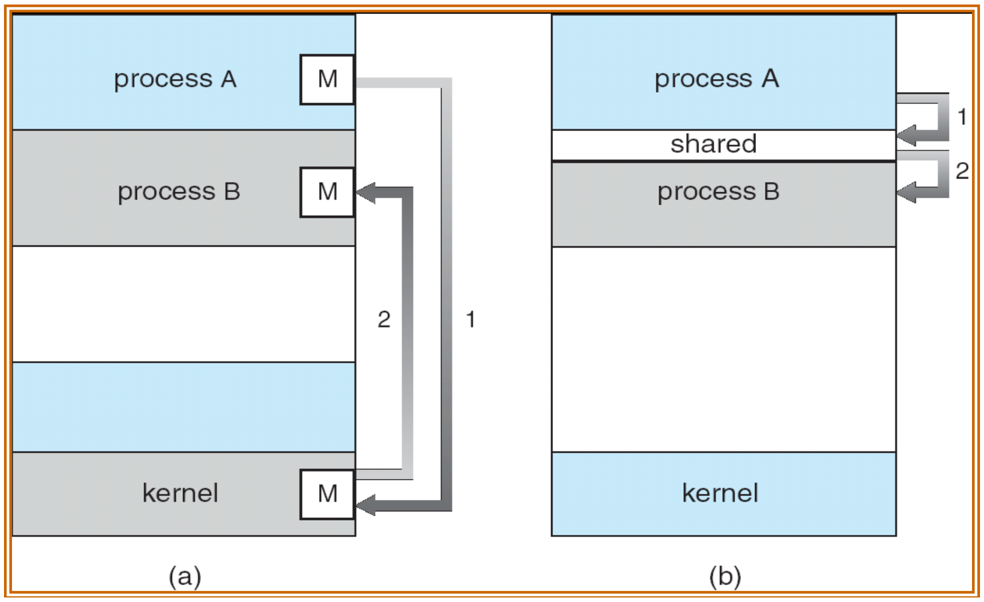
3.4 Interprocess Communication (IPC)

- ✓ Processes executing concurrently in the operating system may be either **independent** processes or **cooperating** processes.
- ✓ A process is **independent** if it cannot affect or be affected by the other processes executing in the system. Any process that **does not share data** with any other process is independent.
- ✓ A process is **cooperating** if it can affect or be affected by the other processes executing in the system. Any process that **shares data** with other processes is a cooperating process.
- ✓ **Advantages** of process cooperation are,
  - **Information sharing**: several users may be interested in same piece of information, so an environment must be provided to allow concurrent access to such information.

- **Computation speed-up**: If we want particular task to run faster, it can be broken into subtasks, each of which will be executing in parallel with each other.
- **Modularity**: If the system is to be constructed in modular fashion, then system functions can be divided into separate processes or threads.
- **Convenience**: An individual user may work on many tasks at the same time.

✓ Cooperating processes require an **Interprocess Communication (IPC)** mechanism that will allow them to **exchange** data and information. There are **two fundamental models** of IPC as shown in **figure.**

1. **Shared memory**: A region of memory that is shared by cooperating processes is established. Processes then exchange information by reading and writing data to the shared region.
2. **Message passing**: Communication takes place by means of message exchange between the cooperating processes.



Communication models (a) Message passing. (b) Shared memory.

✓ **The differences between these two models are,**

Message passing	Shared memory
a. Useful for exchanging small amount of data.	a. large data
b. easy to implement	b. complex
c. slower	c. faster
d. implemented using system calls	d. system calls are required only to establish Shared memory region

- **Shared Memory System**
  - ✓ A region of memory that is shared by cooperating processes is established. Processes then exchange information by reading and writing data to the shared region.
  - ✓ To illustrate cooperating processes, consider **producer-consumer problem**.
  - ✓ Producer process produces information that is consumed by a consumer process.
- ✓ One **solution** to producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be a **buffer of items** that can be filled by a producer and emptied by consumer. The buffer will reside in a shared memory region.
  - ✓ The 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 by the producer.
  - ✓ **Two types** of buffers can be used.
    - o **Unbounded-buffer:** places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.
    - o **Bounded-buffer:** assumes that there is a fixed buffer size, so the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.
  - ✓ The following **variables** reside in a region of memory shared by the producer and consumer processes

```
#define BUFFER_SIZE 10
typedef struct {
……..
……..

}item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

  - ✓ The shared buffer is implemented as a circular array with **two** logical pointers **in and out**. The variable **in** points to the next free position in the buffer; **out** points to the first full position in the buffer. The buffer is empty when in= =out, the buffer is full when ((in+ 1)% BUFFER_SIZE) = = out.
  - ✓ The **code** for the **producer** process is shown below.

```
item nextProduced;
while (true)
                                         {
/* produce an item in nextProduced */
                                        while ( ((in + 1) % BUFFER_SIZE) = = out);  //do nothing
                                        buffer[in] = nextProduced;
                                        in = (in + 1) % BUFFER_SIZE;
                                        }
```

- ✓ The **code** for the **consumer** process is shown below.

```
item nextConsumed;
          while (true)
                    {
                           while (in = = out);  //do nothing
                    nextConsumed = buffer[out];
                    out = (out + 1) % BUFFER_SIZE;
                          /* consume the item in nextConsumed */
                    }
```

- **Message Passing System**
  - ✓ Communication takes place by means of **message exchange** between the cooperating processes.
  - ✓ Message passing facility provides two operations.
    - Send (message)
    - Receive (message)
  - ✓ Message size can be fixed or variable.
  - ✓ If P and Q wish to communicate, they need to establish a **communication link** between them and communication link can be,
    - physical (eg: shared memory, hardware bus)
    - logical (eg: logical properties)
  - ✓ Several methods for logically implementing a link are,
    - Direct or Indirect communication
    - Synchronous or asynchronous communication
    - Automatic or explicit buffering

➔ **Naming**

- ✓ Processes that want to communicate must have a **way to refer** to each other. They can use either **direct** or **indirect** communication.
- ✓ Under **Direct Communication** processes must name each other explicitly
- ✓ The send() and receive() primitives are defined as,
  - **Send (P, message)** – send a message to process P.
  - **Receive (Q, message)** – receive a message from process Q.
- ✓ **Properties** of communication link in this scheme are,
  - Links are established automatically between every pair of processes that want to communicate.

   - A link is associated with exactly two communicating processes.
   - Between each pair there exists exactly one link.
- ✓ This scheme exhibits **two** types of **addressing**,
  - **Symmetry**: Both sender and receiver must name the other to communicate.
  - **Asymmetry**: Only the sender names the recipient, the recipient is not required to name the sender. The send() and receive() primitives are defined as,

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

    Receive (id, message) – receive a message from any process; the variable **id** is set to the name of the process with which communication has taken place.
- ✓ In **Indirect Communication** messages are sent and received from **mailboxes** (also referred to as **ports**)
- ✓ A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- ✓ Each mailbox has a unique **id** and processes can communicate only if they share a mailbox.

- ✓ The send() and receive() primitives are defined as,
  - **Send (A, message) –** send a message to mailbox A
  - **Receive (A, message)** – receive a message from mailbox A
- ✓ Properties of communication link are,
  - Links are established only if processes share a common mailbox.
  - A link may be associated with many processes.
  - Each pair of processes may share several communication links.
- ✓ OS allows the process to do the following operations
  - **create** a new mailbox
  - **send** and receive messages through mailbox
  - **destroy** a mailbox

➔ **Synchronization**

- ✓ Message passing may be either **blocking or non-blocking. Blocking** is considered **synchronous. Non-blocking** is considered **asynchronous.**

  - **Blocking send:** The sending process is blocked until the message is received by the receiving process or by the mailbox.
  - **Non-blocking send:** The sending process sends the message and resumes operation.
  - **Blocking receive:** The receiver blocks until a message is available.
  - **Non-blocking receive:** The receiver retrieves either a valid message or a null.

➔ **Buffering**

✓ Messages exchanged by communicating processes reside in a temporary queue, and such queues can be implemented in one of **three** ways,

1. **Zero capacity** – Maximum length is **zero** and sender must wait for the receiver.
2. **Bounded capacity** – **finite** length ofn messages, sender must wait if link is full.
3. **Unbounded capacity** – **infinite** length and sender never waits.

Question Bank

1. What is an Operating system? Explain its functions and goals.
2. Define the essential properties of the following types of operating systems:
   a) Batch
   b) Multiprogramming
   c) Multitasking
   d) Distributed
   e) Realtime
3. Give the examples of real time system application.
4. Explain the function of memory management.
5. Explain the various operating system services.
6. What are the different types of system calls?
7. Explain different types of system structures.
8. Explain file management and its activity.
9. What is microkernel? Discuss the layers of kernel.
10. Explain the different categories of system calls.
10. What are the three main purposes of an operating system?
11. What is the main advantage of multiprogramming?
12. What are the differences between a trap and an interrupt? What is the use of each function?
13. What are the three major activities of an operating system in regard to secondary-storage management?
14. What are the five major activities of an operating systemin regard to process management?
15. List five services provided by an operating system.
16. What is the main advantage of the layered approach to system design?
17. What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?
18. Explain the functions of the following:
    a. System calls b. System programs c. Command interpreter
19. What is a process? What is PCB? What are the different states of a Process? Explain using diagrams?
20. What are two main operations on processes?
21. Write a note on IPC. Explain two methods.

22. Differentiate between shared memory & message passing.
23. Explain in detail direct & indirect communication.

VTU Question Paper Questions

1. Explain fundamental difference between i) N/w OS and distributed OS ii) web based and embedded computing. **(8)Dec 07/Jan 08**
2. What do you mean by cooperating process? Describe its four advantages. **(6)Dec 07/Jan 08**
3. What are different categories of system programs? Explain. **(6)Dec 07/Jan 08**
4. Define OS. Discuss its role from different perspectives**. (7)Dec 08/Jan 09.**
5. List different services of OS. Explain. **(6)Dec 08/Jan 09.**
6. Explain the concept of virtual machines. Bring out its advantages. **(5)Dec 08/Jan 09.**
7. Difference between a trap and an interrupt **(2)Dec 08/Jan 09.**
8. Define an operating system. Discuss its role with user and system view points. **(06) Dec.09/Jan.10**
9. Give features of symmetric and asymmetric multiprocessing systems **(4)Dec.09/ Jan.10**
10. Briefly explain common classes of services provided by various OS for helping use for ensuring efficient operation of system**. (10)Dec.09/ Jan.10**
11. Define OS. Explain its two view points **(5)Dec 2010**
12. What are OS operations? Explain **(6)Dec.09/ Jan.10**
13. Define Virtual machine. With diagram, explain its working. What are its benefits? **(9) Dec.09/Jan.10**
14. Distinguish among following terminologies: Multiprogramming systems, multitasking Systems, multiprocessor systems. **(12)Dec.09/ Jan.10**
15. What are system calls? With examples explain different categories of system call? **(10) Dec 2012**
16. What do you mean by PCB? Where is it used? What are its contents? Explain. **(8) Dec 07/Jan 08**
17. Explain direct and indirect communications of message passing systems**. (6) Dec 07/Jan 08**
18. Explain the difference between long term and short term and medium term schedulers **(6) Dec 07/Jan 08**
19. What is process? Draw and explain process state diagram. **(5) Dec 08/ Jan 09**

20. Discuss operations of process creation and termination in UNIX**. (7) Dec 08/ Jan 09**

21. Describe implementation of IPC using shared memory and message passing. (8)

    Dec08/ Jan 09

22. List and explain the functions and services of an operating system and OS operations.(8) **June/July 16**

23. What are virtual machines? Explain VM-WARE architecture with a neat diagram.(8) **June/July 16**

24. Differentiate between multiprogramming, multiprocessing and multitasking systems. (4) **June/July 16**

25. Explain process states with state transition diagram. Also explain PCB with a neat diagram.(8) **June/July 16**

26. What is IPC? Explain Direct and Indirect communication with respect to message passing systems. (5) **June/July 16**

27. Distinguish between the following pairs of terms:

    i) Symmetric and asymmetric multiprocessor systems

    ii) CPU burst and I/O burst jobs.

    iii) User's view and systems view of OS.

    iv) Batch systems and time sharing systems.

    v) User mode and kernel mode operations. (10) **Dec16/Jan17**

28. List the three main advantages of multiprocessor systems. Also bring out the differences between graceful degradation and fault tolerance in this context.(5) **Dec16/Jan17**

29. What are virtual machines? How are they implemented? (5) **Dec16/Jan17**

30. What is a process? What are the states a process can be in? Give the process state diagram clearly indicating the conditions for a process to shift from one state to another.(8) **Dec16/Jan17**

31. What are the merits of interprocess communication? Name the two major models of inter-process communication.(6) **Dec16/Jan17**

32. What is operating system? Explain multiprogramming and time sharing systems.(6) **June/July 17**

33. Explain dual mode operation in OS with a neat block diagram.(4) **June/July 17**

34. What are system calls? Briefly point out its types (4) **June/July 17**

35. What are virtual machines? Explain with block diagram. Point out its benefits. (6) **June/July 17**

36. What is interprocess communication? Explain its types. (6) **June/July 17**

37. a. Distinguish between the following terms:
    i) Multiprogramming and multitasking
    ii) Multiprocessor and clustered systems (4)

    b. Analyze modular kernel approach with layered approach with a neat sketch. (6)
    c. List and explain the services provided by OS for the user and efficient operation of system. (6) **Dec 18/ Jan19**

38. a. Illustrate with a neat sketch the process states and process control block. (8)
    b. Discuss the methods to implement message passing IPC in detail. (8) **Dec 18/ Jan19**
39. Discuss the implementation of IPC using message passing and shared memory in detail. (6) **Jun/July 19**
40. Explain the different states of a process, with a neat diagram. (4) **Jun/July 19**
41. Explain the role of OS from different viewpoints. Explain the dual mode of operation of an OS. (7) **Jun/July 19**
42. Demonstrate the concept of Virtual Machine with an example. (5) **Jun/July 19**
43. Explain the types of multiprocessing system and the types of clustering. (4) **Jun/July 19**
44. Demonstrate the operations of process creation and process termination in UNIX. (6) **Jun/July 19**