Operating System - 15CS64

Module - 1

Introduction to Operating System & System Structures

Suhas. K.C.
Assistant Professor
Dept of CSE
CIT, Gubbi

What operating Systems do :-

An Operating System is a program that manages the Computer Hardware.

→ It also provides a basis for application Programs & acts as an Intermediary b/w the Computer User & the Computer Hardware.

* A Computer System can be roughly divided into four components.

(1) The Hardware (2) The OS (3) The Application Program (4) The User

→ The Hardware consists of Memory, CPU, ALU, I/O devices & Storage devices

→ The Application Program mainly consists of Word Processors, Spread sheets, Compilers & Web browsers - defines the ways in which these resources are used to Solve the User's problem.

→ The OS controls & Co-ordinates the use of Hardware among the Various Application Programs for the Various Users.

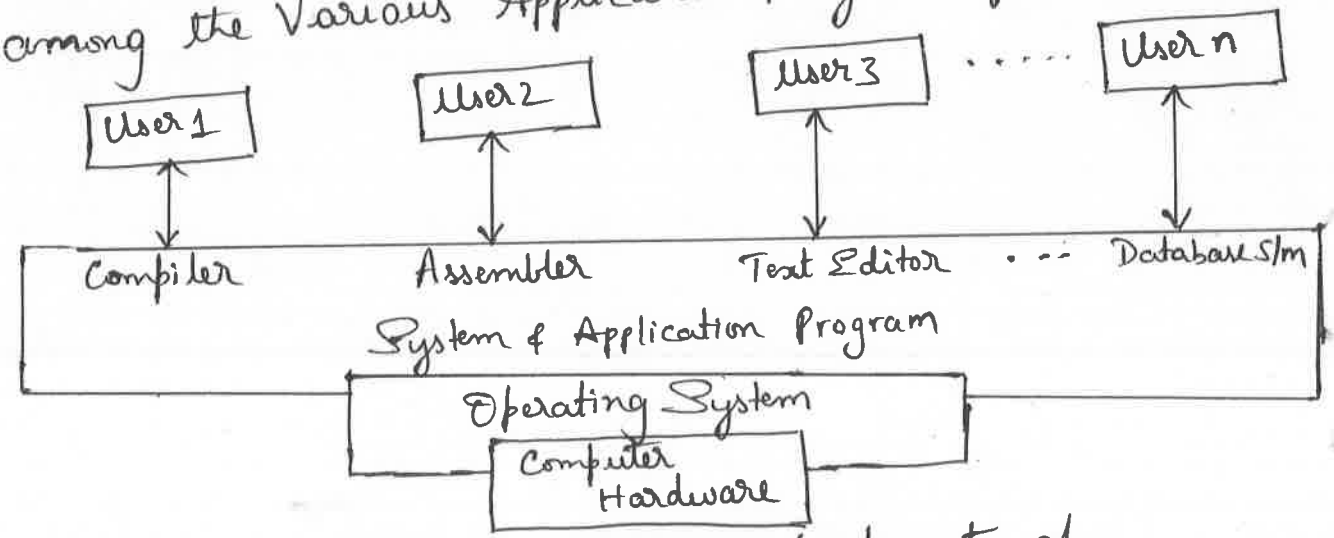


Fig 1:- Abstract View of the Components of a Computer System

Views of OPerating System

(1) User View (2) System Views

(1) User View :-

→ Single User :- The goal is to maximize the work that the User is Performing.

* In this case, the OS is designed mostly for ease of use, with some attention paid to performance & none paid to resource Utilization (How H/w & S/w resources are shared.)

→ Mainframe @ Mini Computer :-

Users may access the same computers through other Terminals These users may share resources & 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.

→ WorkStations :- Users connected to the Networks of other WorkStations & Servers.

In this case the OS is designed to compromise b/w Individual Usability & Resource Utilization.

(2) System View :-

We can View System as Resource Allocator,

A computer S/m has many resources that may be required to solve a problem : CPU time, memory Space, file storage Space, I/o devices & so on.

→ The operating System acts as the manager of these resources.

→ The OS must decide how to allocate them to Specific Programs & Users So that it can operate the computer system efficiently & fairly

→ An OS is a Control Program, It need to Control

Various I/O devices & User Programs, It is used to manage the execution of User Program to prevent errors & improper use of computer.

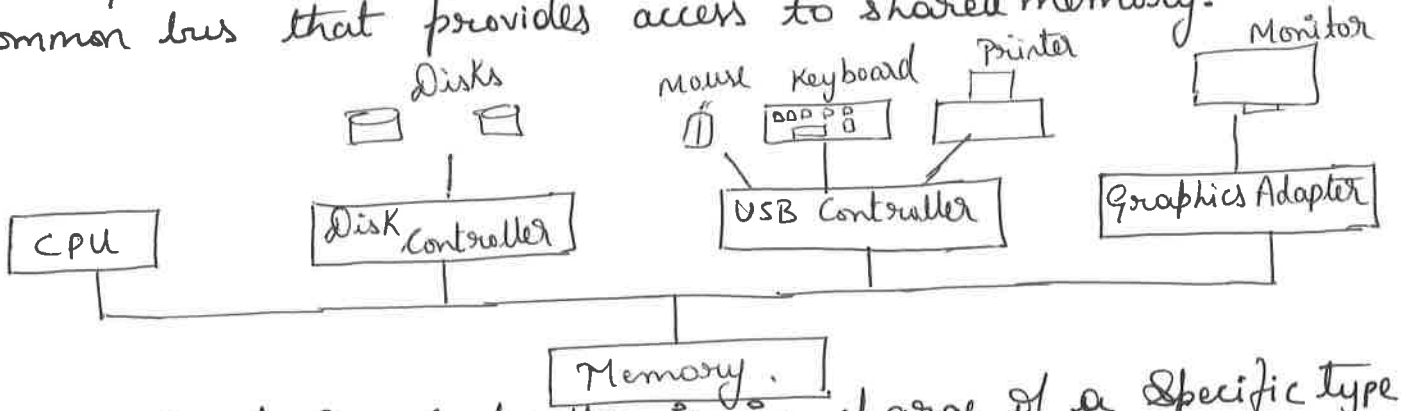
*Defining Operating System

The common function of controlling & allocating resources are then brought together into one piece of software :- "The OS"

Computer System Organization

* Computer System Operations

→ A modern general purpose System consists of one @ more CPU & a number of device controllers connected through a common bus that provides access to shared memory.



→ Each device controller is in charge of a specific type of device (disk drivers, audio devices, video displays)

→ The CPU & the device controllers can execute concurrently competing for memory cycles.

* Computer startup :-

→ When it is powered @ rebooted. It needs to have an Initial Program to run, ie. Bootstrap Program.

→ Typically it is stored in ROM @ EEPROM generally known as Firmware within the Computer Hardware

→ It initializes all aspects of the System from CPU register to device controllers to memory content.

→ The bootstrap program must Know how to load the OS & to start executing that System.

→ The OS then starts executing the first process, such as "init", & waits for some event to occur.

*Interrupts :- The occurence of an event is usually signaled by an interrupt from either the Hardware @ the Software.

→ Hardware may trigger an interrupt at any time by sending a signal to the CPU, by System bus.

→ Software may trigger an interrupt by executing a Special operation called System calls.

→ When a CPU is interrupted, It stops what it is doing & immediately transfer execution to a fixed location, The fixed location usually contains the starting address where the service routine for the Interrupt is located.

→ The Interrupt Service routine executes on completion, the CPU resumes the Interrupted Computation.

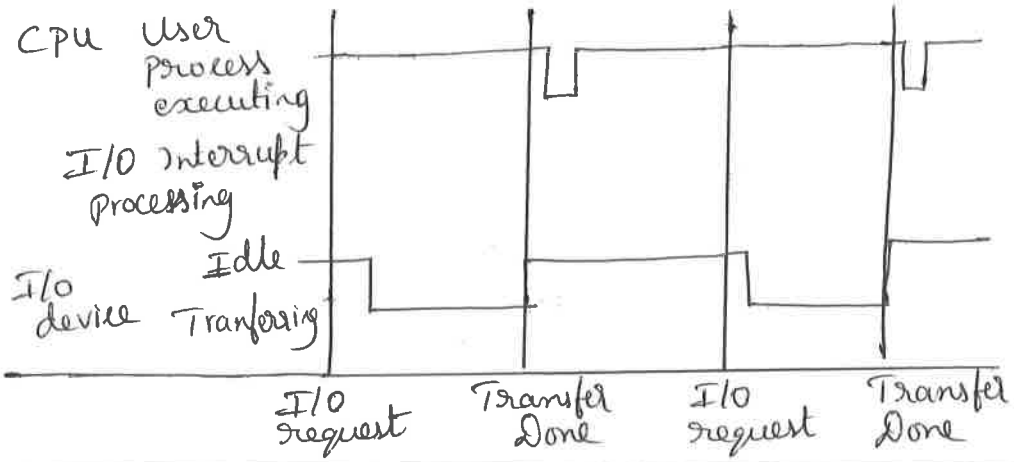


Fig:- Interrupt time line for a Single Process doing output.

→ The Interrupt must transfer control to the Appropriate interrupt service routine.

→ The Straight forward method for handling this transfer would be to invoke a generic routine to examine the interrupt Information, The routine inturn would call the Interrupt Specific Handler.

→ Interrupts must be handled quickly, Since only a predefined numbers of Interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the Necessary Speed.

* Storage Structure :-

→ Computer Programs must be in main memory (RAM) to be executed.

→ Main memory is the only large storage area that the processor can access directly.

→ It is commonly implemented in a Semiconductor technology called Dynamic random access memory (DRAM), which forms an array of Memory Words.

→ Interaction is achieved through a Sequence of load & store instruction to Specific memory address.

→ The load instruction moves a word from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.

→ Von Neumann architecture first fetches an Instruction from memory & stores that instruction in the Instruction register.

→ The instruction is then decoded & may cause operands to be fetched from memory & stored in some Internal register.

→ After the Instruction on operands has been executed the result may be stored back in a memory.

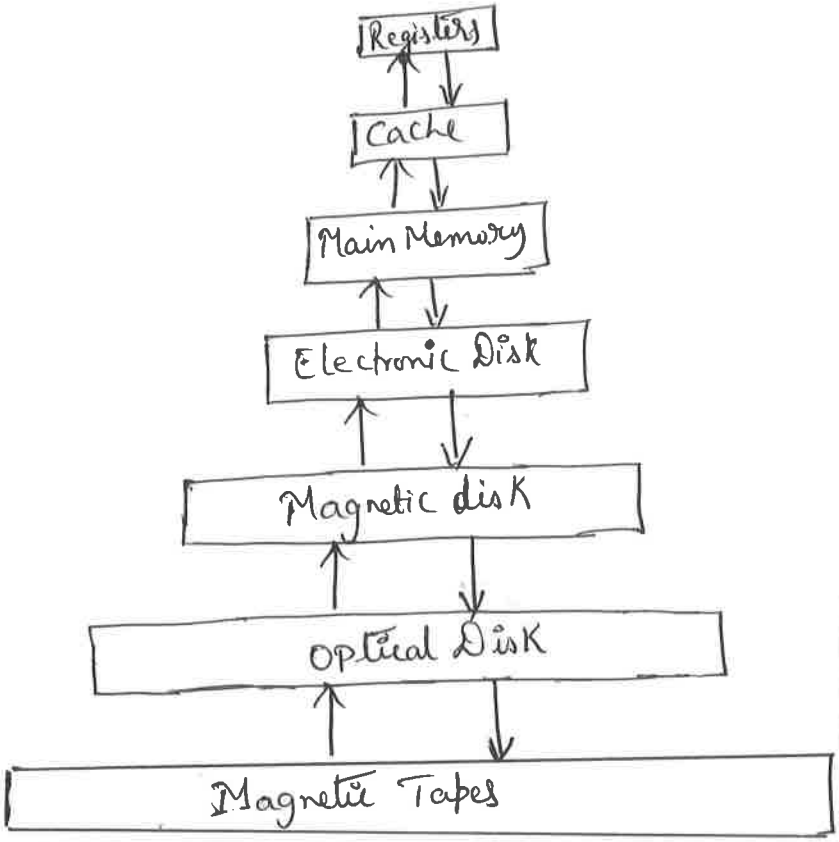
Ideally, we want the Programs & data to reside in main memory Permanently. This arrangement usually is not Possible for the following 2 reasons:-

(1) Main memory is usually too Small to store all needed Programs & data Permanently.

(2) Main memory is a Volatile storage device that loses its content when Power is turned off @ otherwise lost.

* Secondary storage is an extension of Main memory, It is able to hold large quantities of Data Permanently

Fig: Secondary - Storage Device Hierarchy



→ Many Programs are stored on disk until they are loaded into memory

→ Many Programs then use the disk as both a source & a destination of the Information for their Processing.

→ Storage s/m is Organized in Hierarchy

① Cost ② Speed ③ Volatility

Volatile Storage:- "Loses its contents when the Power to device is removed".

In the absence of Expensive battery & generator backup System

→ Data must be written to Non volatile storage for Safe Keeping.

→ In the hierarchy shown in fig, The storage system above the electronic disk are Volatile.

→ Another form of electronic disk is flash memory which is popular in Cameras & Personal digital Assistants. (Pda's)

→ Caching:- Copying Information into faster Storage System. Main memory can be Viewed as a last cache for Secondary Storage.

I/O Structure:-

Storage is only one of many types of I/o devices within a Computer.

→ A large portion of OS code is dedicated to managing I/O. both bcz of its Importance to the reliability & Performance of a System. & bcz of the Varying nature of the devices

~~and because of the varying nature of the devices.~~

→ A general-purpose Computer System consists of CPU's & multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device.

→ Depending on the controller, there may be more than one attached device. For instance, seven or more devices can be attached to a Small Computer System Interface (SCSI) controller.

→ A device controller maintains some local buffer storage & a set of special purpose registers.

→ The device controller is responsible for moving the data b/w the peripheral devices that it controls & its local buffer storage.

→ ~~I/O device~~

→ OS have a device driver for each device controller. These device driver understands the device controller & present a uniform interface to the device to the rest of the OS.

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

→ The device controller, in turn examines the contents of these register to determine what action to take (such as read)

→ The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.

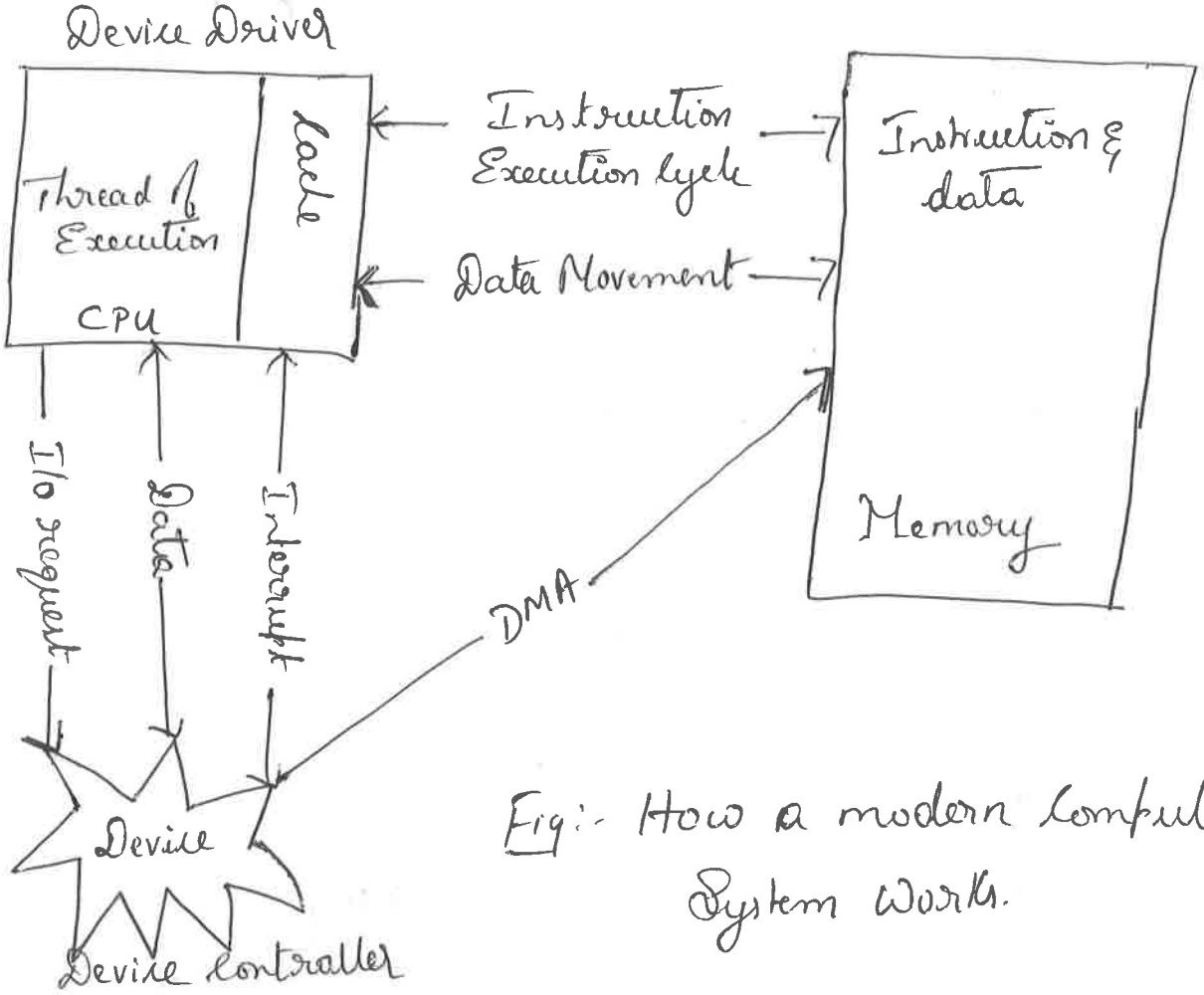


Fig:- How a modern computer system works.

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

→ To solve this problem, Direct memory access (DMA) is used.

→ Only one interrupt is generated /block, to tell the device driver that the operation has completed.

(1.3) Computer - System Architecture :-

(1) Single Processor System :-

→ System Uses Single Processor.

→ On a Single Processor System, there is one main CPU capable of executing a general purpose Instr set, including Instructions from User processes.

(2) Multiprocessor System :- Systems have Two or more processors in close communication sharing the computer bus & sometimes the clock, memory & peripheral devices.

→ multiprocessor Systems have 3 main advantages.

(1) Increased Throughput :- By Increasing the no of processor, we except to get more work done in less time.

(2) Economy of Scale :- Multiprocessor systems can cost less than Equivalent multiple Single processor System because they can share pheripherals, mass storage & power Supplies. If Several programs operate on same ~~data~~ set of data, it is cheaper to store those data on one disk & to have all the processors share them to have many computers with local disks & many copies of the data.

(3) Increased Reliability :- If functions can be distributed properly among Several processors, then the failure of one processor will not halt the System, only slow it down.

→ Multiprocessors System in use today is of 2 types

(1) Assymmetric multiprocessing :- In which each processor is assigned a specific task. A master processor controls the System. It schedules & allocates work to slave processors.

(2) Symmetric multiprocessing :- In which each processor performs all tasks within the operating System

→ NO Master Slave Relationship

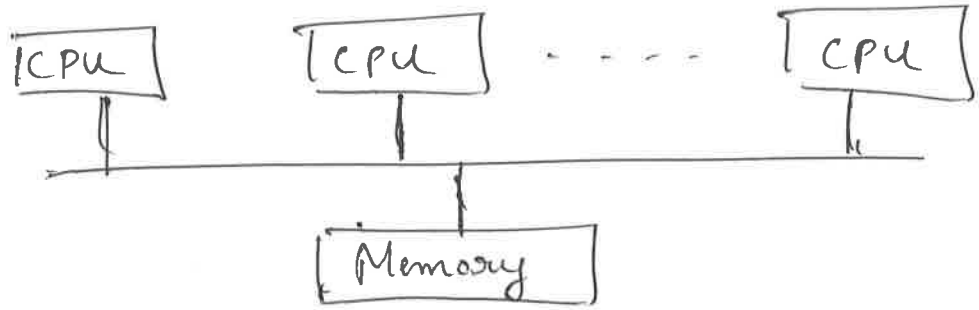


Fig:- Symmetric multiprocessing architecture

Clustered Systems :-

Like parallel Systems the clustered systems will have multiple CPU but they are composed of two @ more individual System coupled together.

→ clustered systems share storage & closely linked via LAN network.

→ clustering is usually done to provide high availability.

→ clustered systems are integrated with Hardware & Software

H/w clusters means sharing of high Performance disk.

S/w clusters are in the form of unified control of a computer system in a cluster.

→ A layer of s/w cluster runs on the cluster nodes. Each node can monitor one @ more of the others. If the monitored machine fails the monitoring Machine Take ownership of its storage & restart the application that were running on failed Machine.

→ clustered systems can be categorized into 2 groups.

(a) Assymmetric clustering
(b) 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 @ more hosts are running the application & they monitor each other. This mode is more Efficient Since it uses all the available Hardware.

→ Parallel clustering & clustering over a LAN is also available in clustering. Parallel clustering allows multiple hosts to access the same data on shared storage

→ clustering provides better reliability than the multiprocessor System.

→ It provides all the key advantages of distributed systems.

→ clustering technology is changing & include global clusters in which machine could be anywhere in the world.

(1.4) Operating System Structures :

Multiprogrammed System :-

→ If there are two @ more programs in the memory at the same time sharing the processor this is reffered as multiprogrammed OS.

→ It increases the CPU utilization by organizing the Jobs so that the CPU will always have one Job to execute.

→ The OS keeps several Jobs in memory simultaneously. This set of Jobs can be a subset of the Jobs kept in a Job Pool → which contains all Jobs that enter the System.

→ The OS picks & begins to execute one of Jobs in memory

→ Eventually the Job may have to wait for some Task, such as an I/O operation, to complete. In a non multiprogrammed system the CPU would sit idle.

→ In a multiprogrammed S/m, the OS simply switches to, & executes, another Job. When that Job needs to wait, the CPU is switched to another Job, & so on.

→ Multiprogrammed S/m monitors the state of all active programs & S/m resources & ensures the CPU is never idle until there are no Jobs

→ This will keep the CPU & I/O utilization busy.

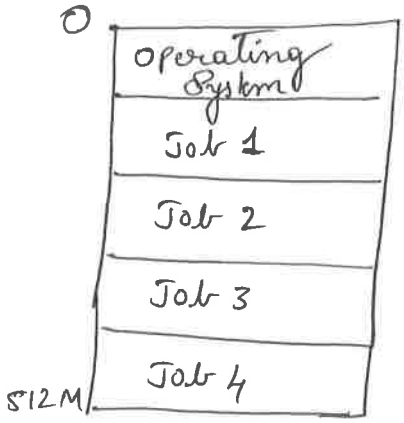


Fig:- Memory layout for a multiprogrammed System.

Time Sharing Systems :-

→ Time sharing system @ Multitasking is logical extension of multi-programming Systems. In Multitasking the CPU executes multiple Jobs by Switching b/w them but the Switching occurs so frequently that users can interact with each Program while it is running.

→ An interactive & hands on System provides direct communication b/w the user & the S/m. The user give the Instr to the OS @ Program directly through Keyboard @ mouse & waits for Immediate results.

→ A time shared S/m uses CPU Scheduling & multiprogramming to provide each User a Small Portion of time shared computers. ~~when a process executes it will be~~

→ A time shared S/m allows multiple users to use the computer simultaneously. Since each action @ Commands are short in a time shared s/m ~~tends only a small cpu time~~ tends to be short, only a little CPU time is needed for Each User.

→ Each user has at least one seperate Program in memory. A pgm loaded into memory & executing is called Process.

→ When a process executes, it typically executes for only a short time before it either finishes @ needs to perform I/O. I/O may be interactive; that is o/p goes to display for the User & the I/O comes from a ~~User~~ Keyboard, mouse @ other devices.

→ If Several Jobs are ready to be brought into memory & if there is not Enough room for all of them, then the System must choose among them Making this decision is Job Scheduling.

→ If Several Jobs are ready to run at the same time the system must choose among them, Making this decision is CPU Scheduling.

→ In the time sharing system, the OS must ensure reasonable response time, which is sometimes accomplished through swapping. where processes are swapped in & out of main memory to the disk.

→ A more common method for achieving this goal is Virtual memory, a technique that allows the execution of a process that is not completely in memory

→ The main advantage of the Virtual memory scheme is that enables users to run programs that are larger than actual physical memory.

→ Time sharing system should also provide a file system & file system resides on collection of disks so this need disk mgt. It supports concurrent execution, OS Synchronization & Communication.

(1.5) Operating System Operations

→ OS is Interrupt driven.

→ If there are no processes to execute, no I/O devices to service, no users to whom to respond, an OS will sit quietly, waiting for something to happen. Events are almost signaled by the occurrence of an interrupt @ a trap.

→ A trap is a Software generated interrupt caused either by an error (for Ex. division by zero @ Invalid memory access) @ by a specific request from a user pgm that an operating system service be performed.

→ For each type of interrupt, Separate Segments of code in the OS determine what action should be taken.
→ An Error in a user Program could cause problems only for the One Program that was running.
→ while sharing, Many Processes could be adversely affected by a bug in one Program.
→ Many Errors can occur in a multiprogramming S/m where one errors may modify another Program. Hence proper protection against these sorts of errors has to be Provided.

<1.5.1>

Dual Mode Operation :-

To Ensure proper execution of OS, we must be able to distinguish b/w the Execution of OS code & user defined code. This is Supported by allocating Various modes of Execution

→ There are 2 modes of Execution
  ① User Mode
  ② Kernel Mode (Supervisor mode, S/m mode, Priviligedmode)
→ A bit called the mode bit is added to the Hardware of the Computer to Indicate the current mode: Kernel(0) & User(1)
→ When the S/m is executing on behalf of a User applicatim the System is in User mode.
→ When a Server is requested by User from OS, It must transit from User to Kernel mode to fulfil the request as shown in figure

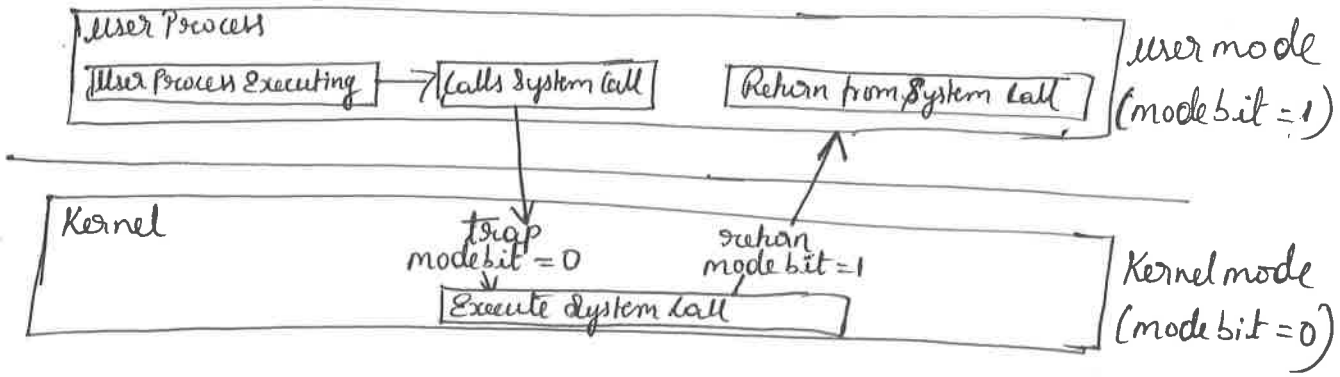


Fig :- Transition from User to Kernel mode

→ As System boot time, the hardware starts in the Kernel mode. The OS is then loaded & starts user application in User mode.

→ Whenever a trap or interrupt occurs, the hardware Switches from User mode to Kernel mode (that is, changes the state of the mode bit 0 to 1). Thus, whenever the OS gains control of the computer, it is in the Kernel mode. The System always Switches to user mode (by setting the mode bit to 1) before passing control to a User Program.

② Timer :-

→ Timer to prevent Infinite loop & to prevent a user program from getting stuck.

→ A Timer can be set to interrupt the computer after a Specified period. The period may be fixed (for Eg 1/60 Second)

→ A Variable timer is generally implemented by a fixed rate clock & a counter.

→ The OS sets the counter. Every time the clock ticks, the counter is decremented.

→ When the counter reaches 0, an interrupt occurs.

→ ~~Before~~ If the timer interrupts, control transfer automatically to the OS, which may treat the interrupt as a fatal Error or may give the program more time.

1-6) Process Management :-

→ A Process is a program in execution.

→ A Process abstraction is a fundamental OS mechanism for the management of concurrent Program Execution.

→ The OS responds by creating Process.

→ Process requires certain resources like CPU time, Memory, I/O devices. These resources are allocated to the Process when it created or while it is running.

→ When Process terminates the Process reclaims all the reusable resources.

→ Process refers to the execution of Machine Instructions.

→ ~~A Single process needs certain resources~~ —

→ A Single threaded Process has one program counter specifying the next instruction to execute.

→ A Pgm by itself is not a process but is a passive entity, such as the contents ~~stored~~ of a file stored in a disk. whereas the Process is an active entity

→ The OS is responsible for the full activities of the Process management

① Creating & destroying of the User & System Process.
② Allocating Hardware resources among the Processes.
③ Controlling the Progress of the Process.
④ Provides mechanism for Process communication.
⑤ Provides mechanism for deadlock handling

1.7 Main Memory Management

→ Main memory is the centre to the operation of the modern computer

→ Main memory is the array of bytes ranging from hundreds of thousands to billion. Each byte will have their own address.

→ The central Processor reads the instruction from main memory during Instruction fetch cycle & it both reads & write the data during the data fetch cycle.

→ The main memory is generally a large storage device in which a CPU can address & access directly.

→ When a Program is to be executed, It must be ~~loaded~~ mapped to absolute addresses & loaded into memory. As the program executes, it accesses program instructions & data from memory by generating these absolute addresses.

Eventually the Program terminates, its memory space is declared available, & the next program can be loaded & Executed.

→ To improve both the utilization of the CPU & the speed of the Computer response to its users, general purpose computers must Keep Several Programs in memory, creating a need for memory management.

→ Several memory Mgt scheme are available & Selection depends on the Hardware design of the System

The OS is responsible for the following activities

(1) Keeping track of which parts of memory are currently being used & by whom
(2) Deciding which processes. & data to move into & out of memory
(3) Allocating & deallocating memory Space as needed.

(1.8) Storage Management

→ OS Provides a Uniform, logical View of Information storage
→ Abstracts physical Properties to logical storage Unit - file
→ Each medium is controlled by device

~~→ The OS maps files onto~~

→ The OS maps files onto physical media & accesses these files via the storage devices

(8.1) File System Management :

→ File management is one of the most visible components of an OS.

→ Computer stores data on diff types of physical media like Magnetic disks, Magnetic tapes, optical disks etc.

→ For convienient use of the computer system the OS provides uniform logical view of Information storage.

→ The OS maps file on to physical media & access these files via Storage devices

→ A file is logical collection of Information.

→ File consists of both Program & data. Data files may be numeric, alphabets @ alphanumeric.

→ Files can be organised into directories.

* The OS is responsible for the foll activities

(1) Creating & deleting of files
(2) Creating & deleting directories
(3) Supporting primitives for manipulating files & directories
(4) Mapping files onto Secondary storage
(5) Backing up files onto Stable (nonVolatile) storage media.

1.8.7 Mass Storage Management

→ Main Memory is too small to accommodate all data & Programs, & b.c the data that it holds are lost when power is lost, the computer s/m must provide Secondary storage to back up main Memory.

→ Most Programs – Including compilers, assemblers, word Processors, Editors & formatters are stored on a disk until loaded into memory & then use the disk as both the source & destination of their Processing.

The OS is responsible for fall activities

(1) Free Spall Management
(2) Storage Allocation
(3) Disk Scheduling.

→ Secondary storage is used frequently, It must be used Efficiently The entire speed of operation of a computer may hinge on the speeds of the disk subsystems & the algorithms.

1.8.37 Caching

→ Information normally kept in some storage s/m (such as Main Memory). As it is used, It is copied into a faster System – The Cache – On a Temporary Basis.

→ When we need a particular piece of Information we first check, whether it is in the cache, If it is we use the Information directly from the cache, If it is not we use the Information from the source, putting a

copy in the cache under the assumption that we need it again soon.

⇒ The movement of Information b/w levels of a storage hierarchy may be either explicit or implicit, depending on the Hardware design & the controlling OS software.

⇒ For instance, Data transfer from cache to CPU & registers is usually a Hardware function, with no operating system intervention., In contrast, transfer of data from disk to memory is usually controlled by the OS.

CMOS - Complimentary metal oxide Semiconductor

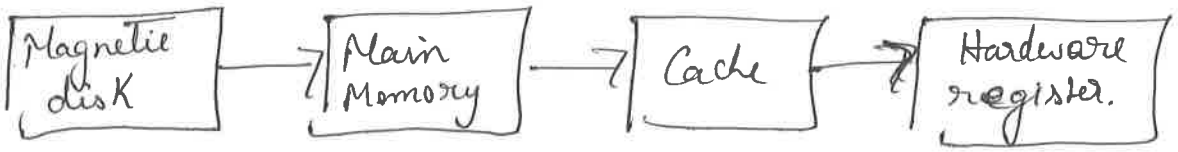


Fig:- Migration in levels of storage S/m

Level	1	2	3	4
Name	Registers	Cache	Main Memory	Disk Storage
Typical size	< 1KB	716 MB	7169 B	7 100 GB
Implementation technology	Custom memory with multiple ports CMOS	on chip or off chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0·25 – 0·5	0·5 – 25	80 – 250	5,000,000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	Compiler	Hardware	Operating System	Operating System
Backed by	Cache	Main Memory	Disk	CD or tape

Fig:- Performance of Various levels of Storage

1.8.4 I/O Subsystem:

→ One purpose of OS is to hide peculiarities of Hardware devices from User.

→ The I/O Subsystem Consists of Several Components

① Memory Mgt ∩ I/O buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for Performance), Spooling (The overlapping of o/p of one Job with i/p of other Jobs).

② General device-driver Interface

③ Drivers for Specific Hardware devices.

1.9 Protection and Security

→ Protection :- Any mechanism for controlling access of processes or Users to resources defined by the OS.

→ Security – Defense of the System against internal & external attacks.

** Huge range, including denial of Service, worms, Viruses, identity theft, theft of Service.

→ System generally first distinguish among Users, to determine who can do what.

① User identities (User IDs, Security IDs) include name & associated number, one/User.

(2) User ID then associated with all files, processes of that user to determine access control.

(3) Group Identifier (group ID) allows set of users to be defined & controls managed, then also associated with each Process, file.

(4) Privilege Escalation allows user to change to effective ID with more rights.

(1.10) Distributed Systems :-

→ A distributed System is a collection of physically seperate, possibly heterogeneous computer Systems that are networked to provide the users with access to the various resources that the System maintains.

→ Access to shared resources increases computation speed, functionality, data availability & reliability.

→ A distributed Systems is one in which hardware & Software components located at the networked computers communicate & co-ordinate their action only by passing messages.

→ A distributed Systems looks to its user like an ordinary OS but runs on multiple independent CPU's.

→ A N/W, in the simplest terms, is a communication path b/w two & more Systems. Distributed Systems depends on networking for their functionality.

Special Purpose System:- There are the systems whose functions are most limited & whose objective is to deal with limited computation domains

(1) Real time Embedded Systems:-

→ Embedded devices are found everywhere, from car Engines & manufacturing robots to VCRs & microwave Ovens.

→ They tend to have very specific tasks. The s/m they run on are usually primitive & so the OS Provide limited features, they have little @ no user Interface, preferring to spend their time monitoring & managing H/w devices.

→ The Power of these devices is more.

→ Embedded s/m almost always run real time OS.

→ A real time s/m is used when rigid time requirements have been placed on the operation of a processor @ the flow of data, it is often used as control device in a dedicated application.

→ Sensors brings data to the computer. The computer must analyse the data & possibly adjust controls to modify the sensor inputs.

→ Systems that control scientific Expt, medical imaging s/m industrial control systems & certain display s/m are real time systems.

→ A real-time s/m has well defined, fixed time constraints, processing must be done within the defined constraints @ the system will fail.

→ A real time systems functions correctly only if it returns the correct result within its time constraints

② Multimedia Systems :-

- Most of the OS are designed to handle only conventional data such as text files, programs, word processing documents & spreadsheets.

⇒ Recently multimedia data are also incorporated into computer s/m data consists of audio & video files as well as conventional files.

⇒ These data differ from conventional data in that multimedia data such as frames of video must be delivered according to certain time restrictions

⇒ Some of the multimedia application are audio files such as MP3, DVD movies, video conferencing & short video clips of movie previews @ news stories downloaded over the internet.

⇒ It also includes live web casts (broadcasting over world wide web) of speeches @ sporting events.

⇒ Multimedia Application include both Audio & Video.

③ Handheld Systems :-

⇒ It include PDA's (Personal digital assistants) such as palm pilot & Pocket PC's & cellular telephones many of which use special purpose embedded OS.

⇒ Developers face problems due to its limited size.

⇒ Limitation of Handheld devices due to its size.

① Amount of Physical memory in handheld depends on the device. but typically it is blw 1MB & 1GB. As a result the OS & applications must manage memory efficiently. This includes returning all allocated memory to manager when the memory is not being used.

② The speed of the processor used in the devices Processors for most handheld devices run at a fraction of speed of a processor in a PC. Faster processor requires more power, to include faster processor in a handheld device would require a large battery, which would take up more space & would have to be replaced more frequently.

③ Small display screen for I/O.

such as palm tops & pocket PCs & cellular telephones Some handheld devices use wireless technology, such as Bluetooth.

1.12.7 Computing Environments

→ Different Computing Environments are

1. Traditional
2. Client Server
3. Peer to Peer
4. Web based

(1) Traditional Computing:

→ Office Environment

→ PCs Connected to a network, terminals attached to mainframe or minicomputers providing batch & Timesharing.

→ Home Networks

→ Used to be single system, then modems Connections for Internet.

→ Some System have firewall to protect their networks from Security breaches

(2) Client Server Computing:

→ Since PCs are faster, powerfull, cheaper etc, designers have shifted away from the centralized System Architecture

→ User's Interface functionality that used to be handled by centralized system is handled by PC's So the centralized system today acts as server program to satisfy the request of client.

→ Server System can be classified as follows.

① Computer - Server System:. Provides an interface to which client can send request to perform some action, in response to which they execute the action & send back results to the client.

② File Server System:. Provides a file system interface where clients can create, update, read & delete files.

③ Peer - to - Peer Systems :.

→ ~~PC's are intr~~ Peer to Peer are considered as standalone computers i.e. only one user can use it at a time

→ In this model, clients & server are not distinguished from one another. instead all nodes within the S/m are considered peer's & each may act as either a client or a server depending on whether it is requesting or providing a service.

→ Peer to Peer as advantage over traditional ~~Cent~~ & client server systems. In a client server, the server is bottleneck

but in a Peer-Peer System, Services can be provided by several nodes distributed throughout the N/w.

→ To Participate in P2P S/M, a node must first Join the N/w of Peers. Once a node has Joined the N/w, it can begin Providing Services.

→ It is Classified in 2 ways in which Service is accomplished

(1) When a node Joins the N/w, It registers its services with a Centralized look up service in the N/w. Any node desiring a specific service first Contacts the Centralized lookup service to determine which node provides the Service. Then the communication takes place b/w client & the service Provider.

(2) A Peer acting as a client must first discover what node provides a desired service by broadcasting a request for the service to all other nodes in the N/w. The node providing that service responds to the Peer making the request. To support this approach, a discovery Protocol must be provided that allows peers to discover services provided by other Peers in the N/w.

(4) Web based Computing :-

→ Web has become ubiquitous (Wireless)

→ PCs are still most prevalent access devices, with workstation, handheld PDAs & even cell phones also providing access.

→ Devices that were networked now have faster N/w Connectivity, provided by either improved networking technology, optimized N/w implementation code or both.

→ The implementation of web based computing has given rise to new categories such as load balancers which manage web traffic among a pool of similar servers in a distributed N/w connections.

CHAPTER - 2 System Structures :

2.1) Operating - System Services

→ An OS provides Services for the execution of the Programs. The Services provided by one OS may be diff from other OS.

→ OS makes the Programming task Easier.

→ The Common Services provided by the OS are

① User Interface :- All OS have a user Interface. This user interface can take Several forms.

* Command line Interface (CLI) :- This uses text commands and a method of entering them. Another is batch interface, in which commands and directives to control those commands are entered into files, & those files are executed.

* Graphical User Interface (GUI) :- This interface is a window System with a pointing device to direct I/O, choose from menus, & make Selections and a Keyboard to enter text.

② Program Execution :- The OS must able to load the Program into memory & run that Program. The Program must end its execution either normally & abnormally.

3) I/O Operation :- A Program running may require any I/O . This I/O may be a file & Specific device users can't control the I/O device directly So that OS must provide a means for controlling I/O devices.

4) File System Interface :- Program need to read & write a file. The OS should Provide permission for the creation & deletion of files by names.

5) Communication :- In certain Situation one process may need to Exchange information with another Process. This Communication may takes place in two ways

* Between the Processes executing on the same computer.
* Between the Processes executing on diff computer that are connected by a network.

⇒ This Communication can be Implemented Via shared memory & by OS.

6) Error Detection :- Errors may occur in CPU, I/O devices & 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.

→ OS with multiple users provide the full Services.

(1) Resource Allocation :- When multiple users logs onto the System & when multiple Jobs are running, resources must be allocated to each of them. The OS manages diff types of OS resources. Some resources may need Some Special allocation codes & others may have Some general request & release code.

(2) 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 & billing. It can also be used to improve System efficiency.

(3) Protection :- 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 multiprocess Environment it is possible that one process may interfere with the other & with the OS, So Protection is required.

(2.2) User Operating System Interface

→ There are 2 fundamental approaches for users to interact with the OS they are.

(1) Command Interpreter :-

→ Some operating System include the command Interpreter in the Kernel, Others Such as Windows XP and Unix, treat the command interpreter as a Special Program that is running, when a Job is initiated or when a user first logs on.

→ Some Systems with multiple command Interpreters to choose from, the interpreters are Known as Shells.

→ The main function of the command interpreter is to get & execute the next user Specified command.

→ Sometimes commands are built in. ~~Sometimes Just names of Programs.~~

→ In the latter, adding new features doesn't require shell Modification.

(2) Graphical User Interface :-

→ A GUI allows provides a mouse based, window & menu System as an interface.

→ A GUI provides a desktop metaphor where the mouse is moved to position its Pointer on images, or icons

on the screen (the desktop) that represent programs, files, directories & System function.

→ Invented at Xerox PARC

* Many Systems now include both CLI & GUI interfaces.
  → Microsoft Windows is GUI with CLI "Command" shell
  → Apple Mac OS X as "Aqua" GUI interface with UNIX Kernel underneath & shells available
  → Solaris is CLI with optional GUI Interfaces (Java Desktop, KDE)

2.3) System Calls :-

→ System calls provides interface b/w the Process & the OS
→ The calls are generally available as assembly language instruction & certain system allow system calls to be made directly from a high level language Program.
→ Several languages have been defined to replace assembly language Program.
→ A System call instruction generates an interrupt & allows OS to gain control of the Processors.
→ System calls occur in diff ways depending on the computer. Some time more information needed

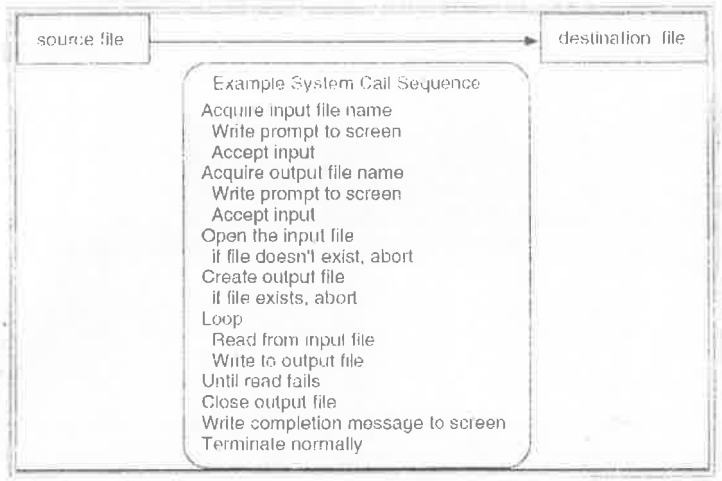
to identify the desired System call. The Exact type & amount of Information needed may vary according to the Particular OS & call.

→ Three most common API's are.

① WIN32 API for Windows

② POSIX API for POSIX based System (UNIX, Linux & MacOSX)

③ Java API for the Java Virtual Machine.



System call Sequence to copy the contents of one file to another file

System call Implementation

→ Typically, a number associated with each System call.

→ These calls are generally available as routines written in C & C++.

→ System call interface maintains a table indexed according to these numbers.

→ The System calls interface invokes intended System call in OS Kernel & returns status of the System call & any return Values.

→ The caller need Know nothing about how the system call is Implemented.

→ Just needs to obey API & understand what OS will do as a result call.

→ Most details of OS interface hidden from Programmer by API.

→ Managed by run-Time Support library (Set of functions built into libraries included with Compiler)

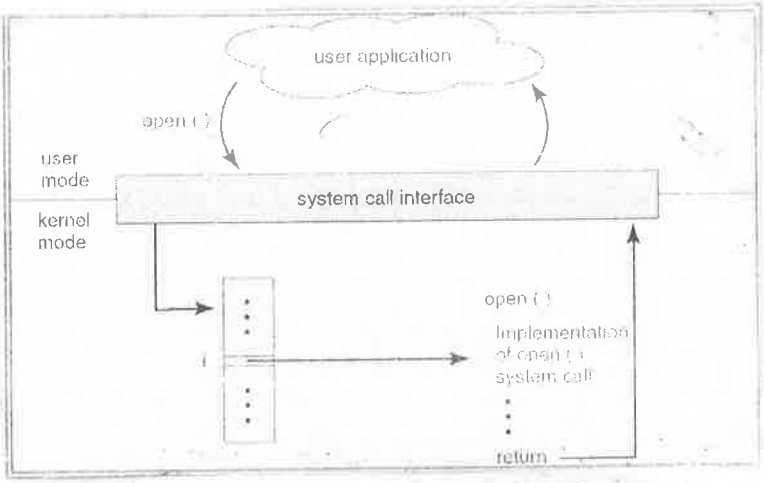


fig:- API - System call OS Relationship

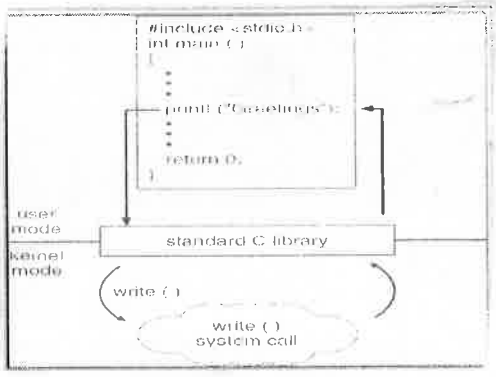
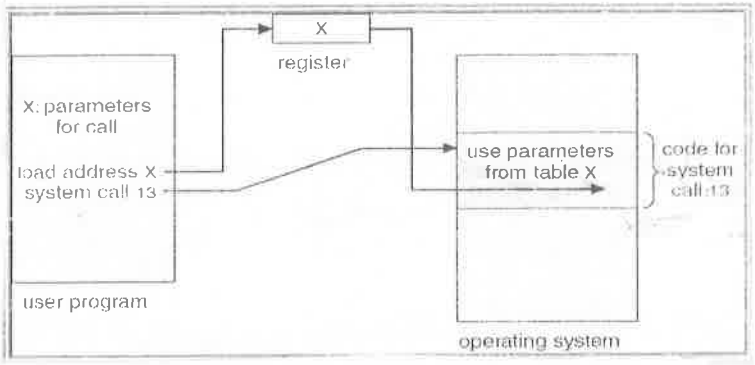


fig:- Standard C Library

Eg:-

→ C Program invoking Printf( ) library call, which calls write( ) System call

PASSING PARAMETERS TO OS



Three general methods are used to pass the Parameters to the OS

(1) → The Simplest approach is to pass the Parameters in registers. In Some there can be more Parameters than register.

(2) → In these the Parameters are generally stored in a block @ table in memory & the address of the block is passed as Parameters in register. This approach used by Linux.

(3) → Parameters can also be placed @ pushed onto stack by the Program & Popped off the stack by the OS.

→ Some OS prefer the block @ Stack methods, because those approaches do not limit the no @ length of Parameters being Passed.

(2.4) Types of System calls

→ System calls may be grouped into 5 categories

(1) Process Control
(2) File Management
(3) Device Management
(4) Information Maintenance
(5) Communication

(1) Process Control :-

* End, abort
* Load, Execute
* Create Process, Terminate Process
* Get Process attributes, Set Process Attributes
* Wait for time
* Wait event, Signal event
* Allocate & Free memory

(2) File Management :-

* Create file, delete file
* Open, close
* Read, Write, reposition
* Get file attributes, Set file attributes.

(3) Device Management :-

* Request device, release device
* Read, Write, reposition
* Get device attributes, Set device attributes
* Logically attach & detach devices

(4) Information Maintenance

* Get time & date, Set time & date
* Get System data, Set System data
* Get Process file, & device attributes
* Set Process file, & device attributes.

[5] Communication :

* Create, delete communication Connection
* Send, receive Messages
* Transfer status Information
* Attach & detach remote devices.

(2.4.1) Process Control

→ A running Program needs to be able to halt its execution either normally (End) & abnormally (abort).

→ A System call can be used to terminate the Program either normally & abnormally. Reasons for abnormal termination are dump of memory, Error Message generated etc.

→ Debugger is mainly used to determine Problem & the dump & return back the dump to the OS.

→ In normal & abnormal situation the OS must transfer the control to the command interpreter System.

→ In batch System the command interpreter terminates the execution of Job & continues with the next Job.

→ A Control Card is a batch System concept. It is a command to manage the execution of a Process.

→ Some System use control cards to indicate the special recovery action to be taken in case of Errors.

→ Normal & abnormal termination can be combined at some Error level. Error level is defined before & The command interpreter uses this Error level to determine next action automatically.

MS-DOS

→ MS-DOS is an Eg of Single tasking System, which has command Interpreter System i.e. invoked when the computer is started. To run a Program MS-DOS uses simple method & does not create a new Process.

→ It loads the Program into memory, writing over most of itself to give the Program as much memory as Possible.

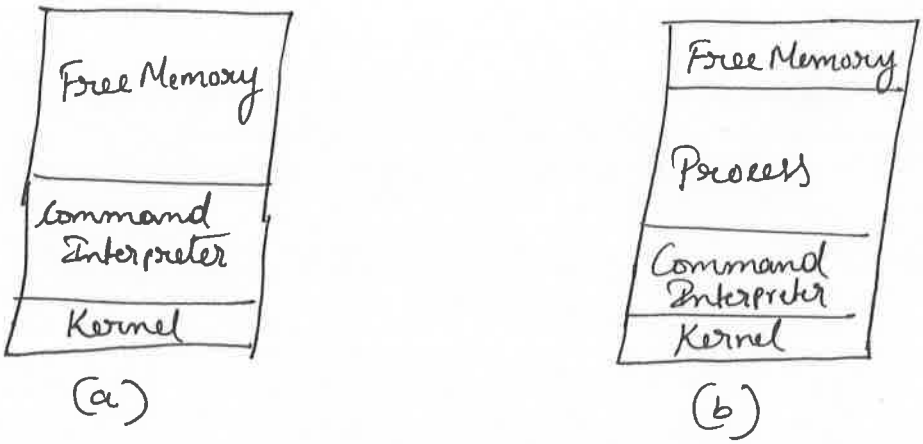


Fig:- MS-DOS Execution (a) At System startup (b) Running a Program

BSD:- - BerKeley S/w Distributio

→ Free BSD is an Eg of multitasking System. In free BSD the command Interpreter may continue running

while other Program is executing.

→ To start a new Process, the shell executes a fork() system call - Then, the Selected Program is loaded into memory via an Exec() system call, & the Program is executed.

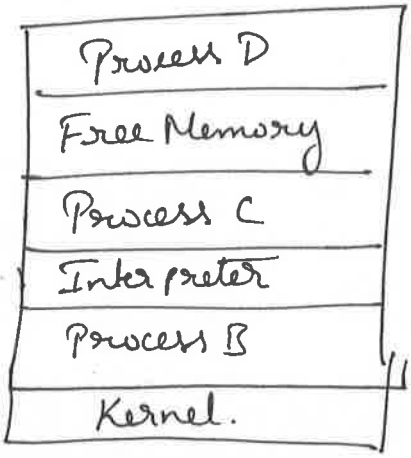


Fig:- FreeBSD running multiple Programs

(2.4.2)

File Management :-

→ System calls can be used to create & deleting of files. System calls may require the name of the files with attributes for creating & deleting of files.

→ Other operation may involve the reading of the file & write & reposition the file after it is opened.

→ Finally we need to close the file.

→ For directories some set of operation are to be Performed Sometimes we require 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.

Device Management :-

→ The System calls are also used for accessing devices.

→ Many of the System calls used for files are also used for devices.

→ In multi user Environment the requirement are made to use the device. After using the device must be released using release System call the device is free to be used by another user. These function 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 services structure. In file & device structure I/o devices are identified by file names.

Information Maintainance :-

→ Many System calls are used to transfer information b/w user program & OS.

Eg:- Most System have the System calls to return the current time & date, number of current users, Version no of OS, amount of free memory & disk space & so on.

→ In addition the OS keeps information about all its processes & there are system calls to access this Information

Communication :-

→ There are two modes of communication.

(1) Message Passing Models :-

→ In this information is exchanged using inter-process communication facility provided by OS.

→ Before communication the connection should be opened.

→ The name of the other communicating Party should be known, it can be on the same computer @ it can be on another computer connected by a computer N/w.

→ Each computer in a N/w may have a host name like IP name Similarly each Process can have a process name. which can be translated into Equivalent identifier by OS.

→ The get host id & process id System call do this translation These identifiers are then passed to the open & close connection System calls.

→ The recipient process must give its Permission for communi-cation to take place with an accept connection call.

→ Most Processes receive the connection through Special purpose System Program dedicated for that purpose called daemons :- The daemon on the Server Side is called Server daemon & the daemon on the client side is called Client daemon.

2) Shared Memory :-

→ In this the processes uses the map memory system calls to gain access to memory owned by another Process.

→ The OS tries to prevent one process from accessing another process memory.

→ In shared memory this restriction is eliminated & they exchange information by reading & writing data in shared areas. These areas are located by these processes & not under OS control.

→ They should ensure that they are not writing to same memory area.

→ Both these types are commonly used in OS & some even implement both.

→ Message Passing is useful when small no of data need to be exchanged. Since no conflicts are to be avoided & it is easier to implement that in shared m/y, It allows maximum speed & convenience of communication as it is done at m/y speed when within a Computer

(2.5) System Programs :-

(1) Provide a convenient Environment of Program development & execution, some of them are simply user interfaces to system calls; others are considerably more complex.

→ They can be divided into 6 categories

1. File Management
2. Status Information
3. File Modification
4. Programming language Support
5. Program loading & Execution
6. Communication

① File Management :- Create, delete, copy, rename, print, dump, list & generally manipulate files & directories.

② Status Information

→ Some ask the system for information — date, time, amount of available memory, disk space, number of users

→ others provide detailed performance, logging & debugging Information.

→ Some systems implement a registry — used to store & retrieve configuration Information.

(3) File Modifications :- Several Text Editors may be available to create & modify the content of files stored on disk.
→ There may also be Special Commands to search contents of files @ Perform transformations of Text.

(4) Programming-Language Support :- Compilers, assemblers, debuggers & interpreters are often provided to the User with OS.

(5) Program Loading & Execution :- Once a Program is assembled @ Compiled, it must be loaded into memory to be executed.
→ The System may provide absolute loaders, relocatable loaders, linkage Editors, & Overlay Loaders.
→ Debugging s/m for higher-level language @ Machine level language are needed

(6) Communications :- Provide the mechanism for creating Virtual connections among processes, users & Computer Systems.
→ They allow Users to Send messages to one another's screens, to browse web Pages, to Send electronic-mail messages, to log in remotely, @ to transfer files from one machine to another.

~~Manipulate files and directories~~

(2) **Status information**
Some ask the system for info - date, time, amount of available memory, disk space, number of users
Others provide detailed performance, logging, and debugging information
Typically, these programs format and print the output to the terminal or other output devices
Some systems implement a registry - used to store and retrieve configuration information

(3) **File modification**
Text editors to create and modify files
Special commands to search contents of files or perform transformations of the text

(4) **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided

(5) **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

(6) **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

2.6 Operating system design and implementation:

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system : - Time shared, Single User, multiuser, distributed, Real time.
- User goals and System goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Important principle to separate

  Policy: what will be done?

  Mechanisms: determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Implementation:

Traditionally os have been written in assembly language now, h/w they are most commonly written in higher level language such as C or C++.

The advantages of using a higher level language:

- ➔ The code can be written faster is more compact and is easier to understand and debug.
- ➔ In addition improvements in compiler technology will improve the generated code for the entire os by simple recompilation.

The disadvantages of implementing os in HLL are reduced speed and increased storage requirements.

2.7 Operating System structure

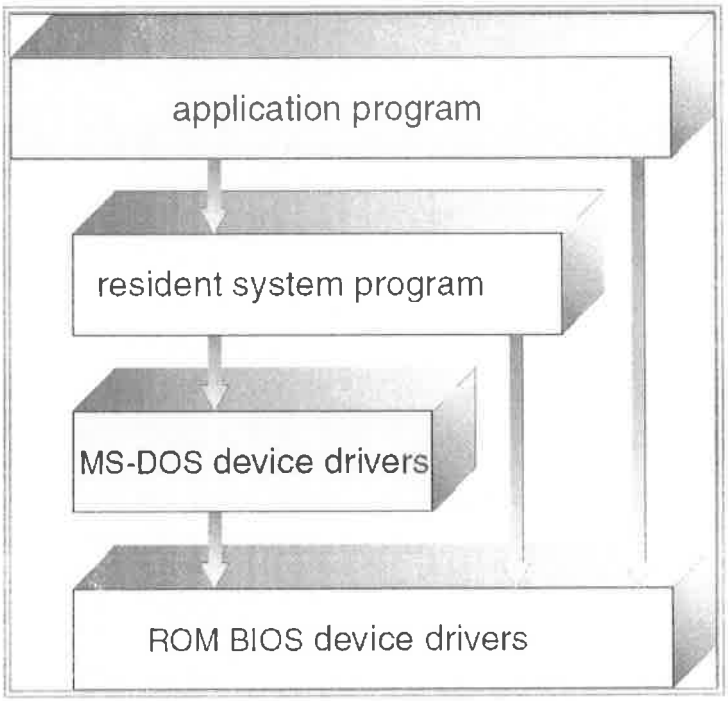
- ➔ Modern OS is large & complex.
- ➔ OS consists of different types of components.
- ➔ These components are interconnected & melded into kernel.
- ➔ For designing the system different types of structures are used. They are,

a. Simple structures.
b. Layered structured.
c. Micro kernels.
d. Modules

Simple Structures

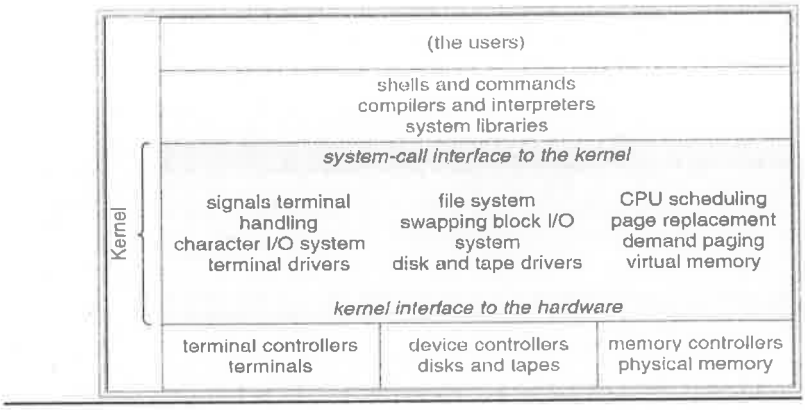
- ➔ Simple structure OS are small, simple & limited systems.
- ➔ The structure is not well defined
- ➔ MS-DOS is an example of simple structure OS.
- ➔ MS-DOS layer structure is shown below

→ MS-DOS written to provide most functionality in the least space.
→ Not divided into modules.
→ Although MS DOS has some structure, its interface & levels of functionality are not well seperated.



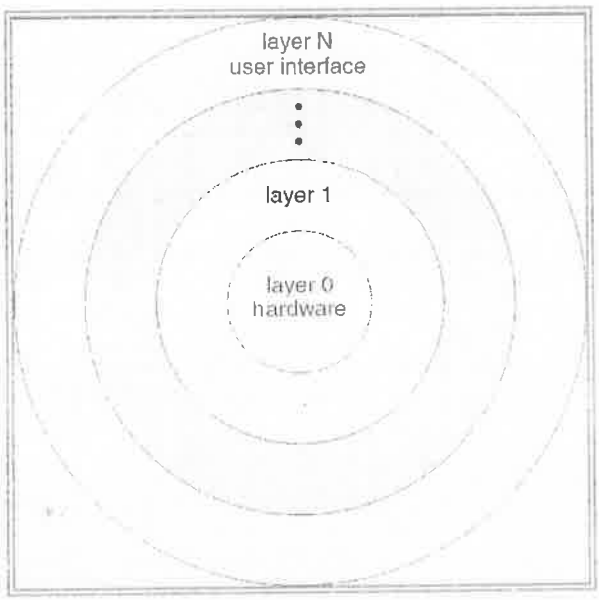
Another example of limited structuring is the original unix os.

- ➔ UNIX consisted of two separate modules
    a. Kernel
    b. The system programs.
- ➔ Kernel is further separated into series of interfaces & device drivers which were added & expanded as the UNIX evolved over years.
- ➔ The kernel also provides the CPU scheduling, file system, m/y management & other OS function through system calls.
- ➔ System calls define API to UNIX and system programs commonly available define the user interface. The programmer and the user interface determine the context that the kernel must support.
- ➔ New versions of UNIX are designed to support more advanced H/w. the OS can be broken down into large number of smaller components which are more appropriate than the original MS-DOS.



Layered Approach:

With proper H/W support, os system can be broken into pieces that are smaller and more appropriate than those allowed by the original ms-dos or UNIX system.



- ➔ In this OS is divided into number of layers, where one layer is built on the top of another layer. The bottom layer is hardware and higher layer is the user interface.
- ➔ An OS is an implementation of abstract object i.e. the encapsulation of data & operation to manipulate these data.
- ➔ The main advantage of layered approach is the modularity 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 it is already debugged.

- ➔ Each layer is designed with only the operations provided by the lower level layers.
- ➔ 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 then the other types.

Micro Kernels:-

- ➔ Micro kernel is a small Os which provides the foundation for modular extensions.
- ➔ The main function of the micro kernels is to provide communication facilities between the current program and various services that are running in user space.
- ➔ This approach was supposed to provide a high degree of flexibility and modularity.
- ➔ This method structure the os by removing all non essential components from the kernel and implements them as system and user level programs..
- ➔ This approach also provides more security & reliability.
- ➔ Most of the services will be running as user process rather than the kernel process.
- ➔ This was popularized by use in Mach OS.
- ➔ Micro kernels in Windows NT provide portability and modularity. Kernel is surrounded by a number of compact sub systems so that task of implementing NT on variety of platform is easy.
- ➔ Micro kernel architecture assigns only a few essential functions to the kernel including address space, IPC & basic scheduling.

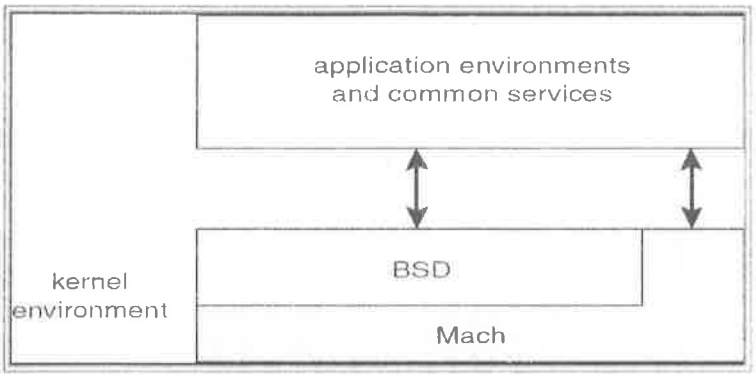


Fig: Mac OS X Structure

Modules:

Perhaps the current methodology for os design invokes using object oriented programming techniques to create a modular kernel.

Here the kernel has a set of core components and dynamically links in addition services either during boot time or run time.

Most modern operating systems implement kernel modules

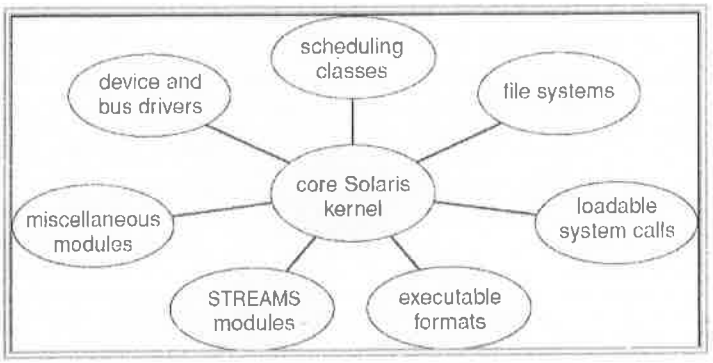
Uses object-oriented approach

Each core component is separate

Each talks to the others over known interfaces

Each is loadable as needed within the kernel

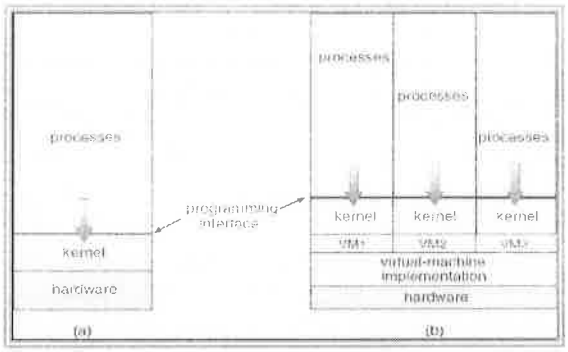
Overall, similar to layers but with more flexible.



Solaris Modular Approach

2.8 Virtual Machines:

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console



(a) Nonvirtual machine (b) virtual machine

Benefits:-

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine

Eg:- Two popular contemporary virtual machines:

→ VM ware

→ Java VM

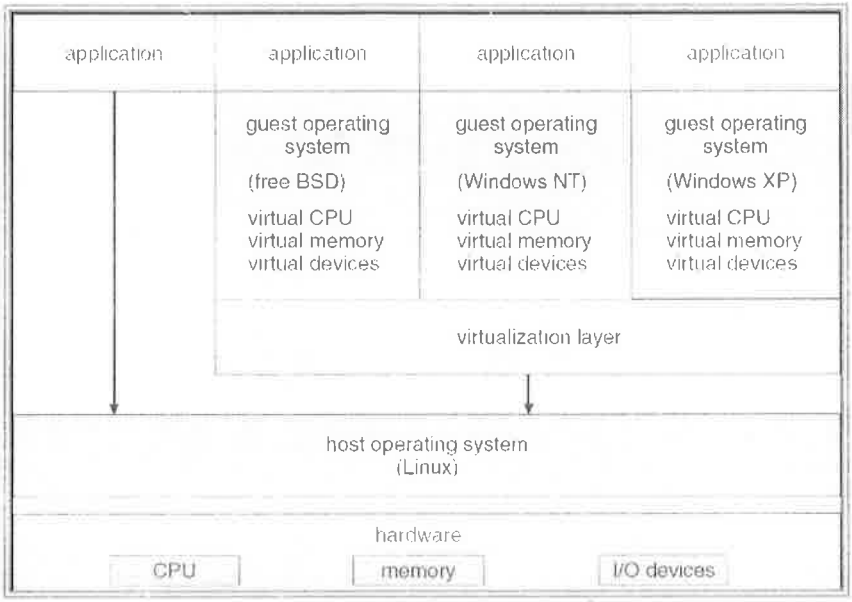
Implementation :- ① The VM provides exact duplicate of the underlying machine
② The Underlying machines has 2 modes @ user mode @ kernel mode

VM ware

VM ware is a popular commercial application that abstracts Intel 8086 H/W into isolated VM.

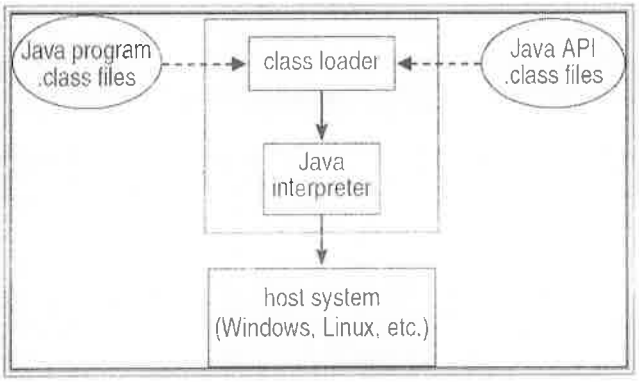
VM ware runs as an application on a host os such as windows or Linux and allows this system concurrently run several different guest as independent VM.

③ The VM Software can run in kernel mode, since it is the OS
④ The VM itself can execute in only user mode.



VM ware Architecture

The Java Virtual Machine



→ Java is popular object oriented Programming language introduced by Sun Microsystem.

→ The Compiler produces an architecture neural byte code O/P (.class) file that will run on any JVM.

- Java objects are specified with the class construct
- A Java program consists of one or more classes. (Java Program, Java API)
- Each Java class the compiler produces an architecture neural byte code output (.class file) that will run on any implementations of the JVM.
- The class loader loads the compiled .class files from both the java program and the java API for execution by the java interpreter.

2.9 Operating system generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site. A process sometimes known as System generation (SYSGEN)
- The os is normally distributed on disk or cd rom to generate system.

→ After a class is loaded, the Verifier checks that the .class file is valid Java byte code & does not overflow or underflow the stack.

→ ~~Java Interpreter~~ JVM also automatically manages memory by performing garbage collection.

- SYSGEN program obtains information concerning the specific configuration of the hardware system ⓒ determine what components are there.
- Booting – starting a computer by loading the kernel
- Bootstrap program – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

The following kind of information must be determined: by Sysgen.

- What CPU is to be used (extended instruction set, floating point arithmetic).
- How much memory is available.
- What devices are available. (address of each device) (device no) (device interrupting)
- what os option are desired, ⓑ what parameter values are to be used. (CPU scheduling)

2.9 System boot:

- After the os is generated, it must be made available for use by the hardware.
- But how does the ~~kernel~~ hardware know where the kernel is or how to load the kernel.
- The procedure of starting a computer by loading the kernel is known as booting the system.
- A small piece of code known as the bootstrap program or bootstrap loader locates the kernel loads it into the main memory and starts its execution.
- Some times two steps process ~~where boot block at fixed location loads bootstrap loader.~~

(1) When power initialized on system, execution starts at a fixed memory location, & execution starts here. At that location is the initial bootstrap program. This program is in the form of ROM.

(2) In which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the Kernel.

Process Concepts

Process:- A process is a program in Execution"

→ A process is more than the program code which is sometimes Known as Text Section

→ A process Includes

(1) Program Counter
(2) Stack
(3) Data Section
(4) Heap

→ Program Counter:- Represent the current Activity

→ Stack:- Contains Temporary Data
[Function Parameter, return addres, local Variables)

→ Data Section:- Contains global Variable

→ Heap:- Is a memory that is dynamically allocated during process run time

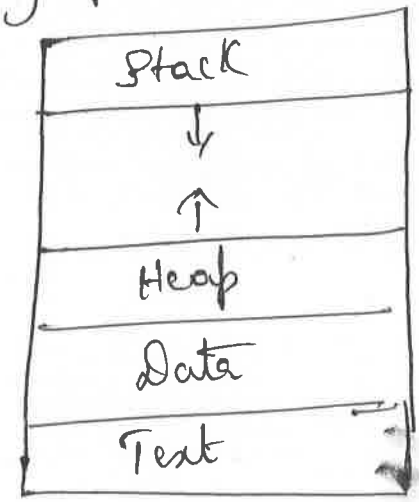


Fig:- Structure of a process in a Memory

→ Program by itself is not a Process

→ A Program is a Passive Entity. Such as a file containing a list of instruction stored on disk. (Executable file)

→ A Process is an active entity, with a Program counter specifying the next instruction to execute & a set of associated resources.

→ A Program becomes a Process when an executable file is loaded into memory.

→ 2 Processes may be associated with the same Program they are nevertheless considered a seperate Execution Sequence.

For Eg:. Several users may be running diff copies of the mail Program @ the same User may invoke many copies of the Web browser Eg. Each of these is a Seperate Process.

* Process State

→ As a Process executes, its changes state.

→ The state of a process is defined by the current activity of that Process. Each Process may be in one of the following states.

① New :- The Process is being created.

② Running :- Instructions are being executed.

(3) Waiting :- The Process is waiting for some event to occur (such as I/o operation)

(4) Ready :- The Process is waiting to be assigned to a Processor

(5) Terminated :- The Process has finished execution.

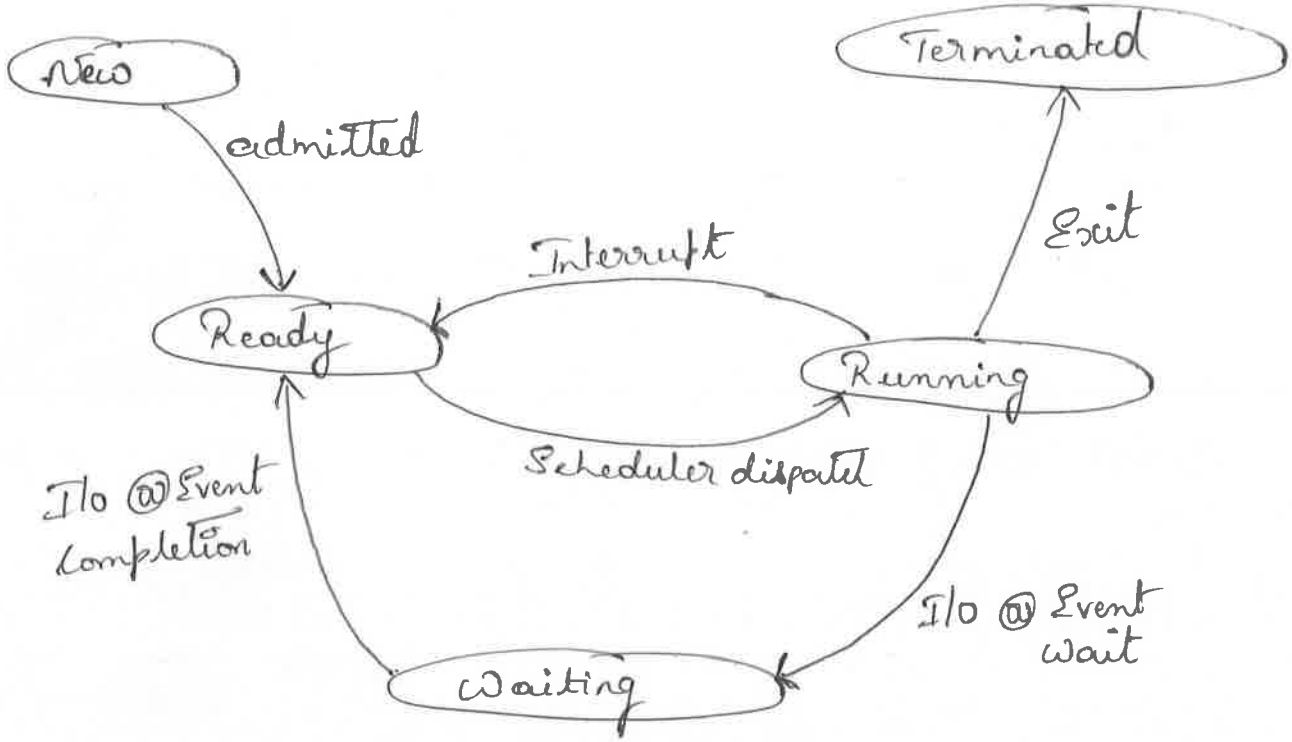


Fig :- Diagram of Process state

* Process Control Block

→ Each Process is represented in the OS by a Process control block (PCB), also called as Task control block.

→ Is a certain store that allows the OS to locate Key Information about Process.

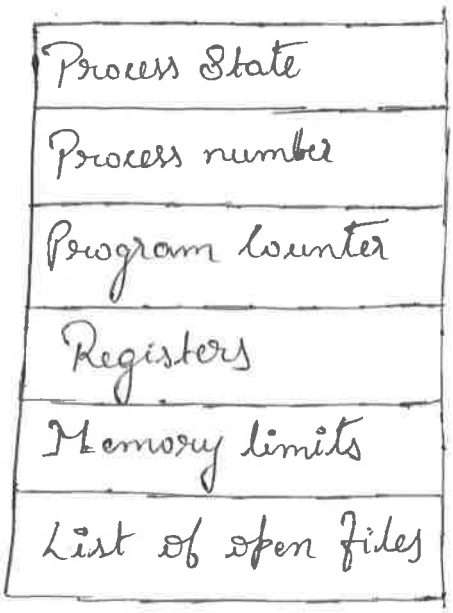


Fig:- Process Control Block (PCB)

→ Process State :- Indicates the current states of process i.e. whether It is new, ready, running, waiting or Terminated.

→ Process number :- Unique identification of the Process in order to track "which is which" Information.

→ Program counter :- Indicates the address of the next instruction to be executed for this Process.

→ Registers :- The register vary in no. & type, They include accumulators, Index registers, Stack Pointers & general Purpose register.

→ Memory Management Information :- This information include a value of the base & limit registers, the page tables, Segment tables, depending on the memory s/m used by the OS.

→ CPU Scheduling Information :- Includes a process priority pointers to Scheduling queues & any other Scheduling Parameters.

→ Accounting Information :- Includes the amount of CPU & real time used, time limits, account nos, Job/process no & so on

→ I/O Status Information :- Includes the list of I/O devices allocated to the Process, list of open files & so on.

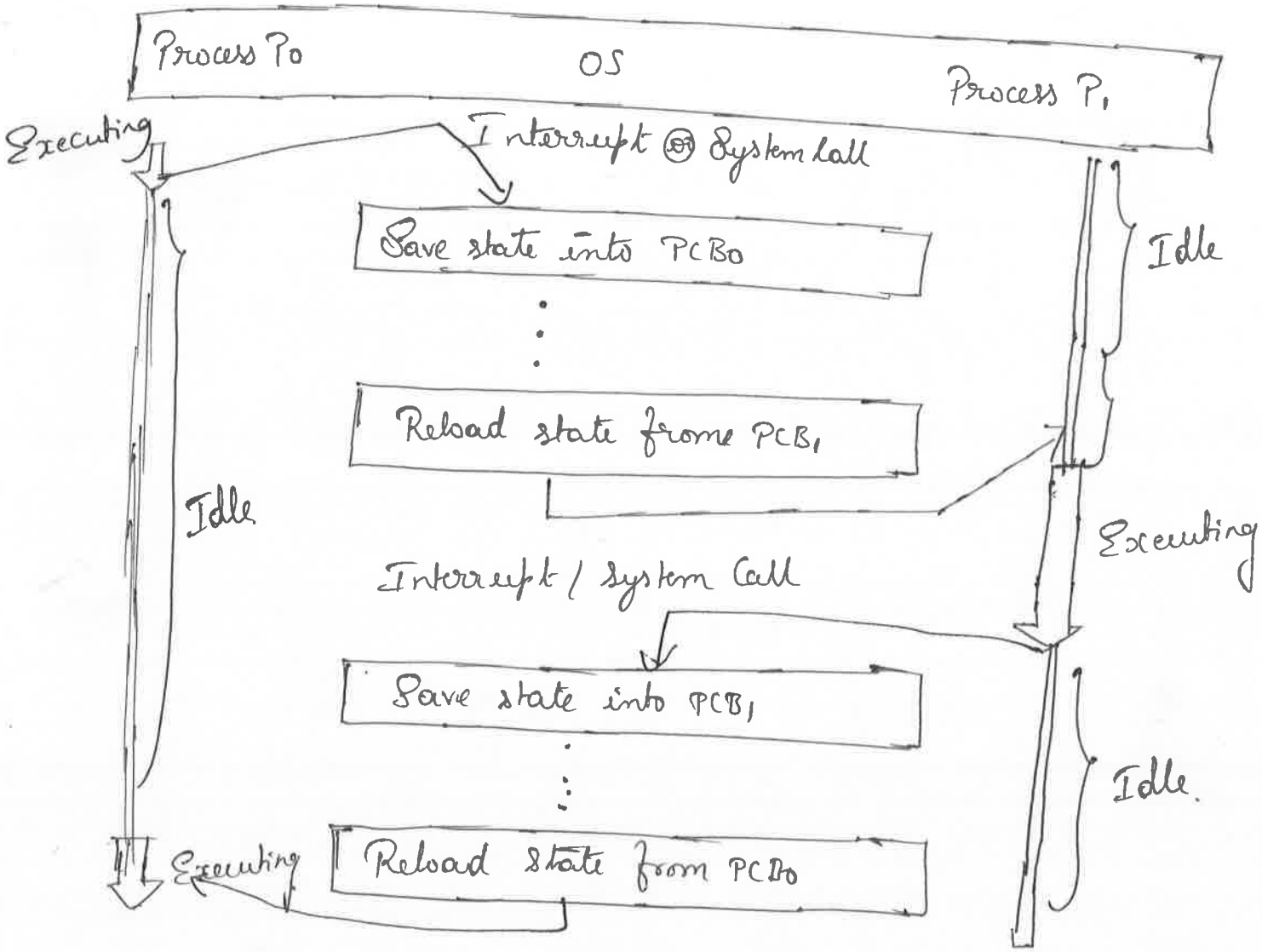


Fig:- Diagram showing CPU Switch from Process P0 to Process P1

3.2 Process Scheduling :-

→ The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization, The objective of time sharing is to switch the CPU among processes so frequently that user's can interact with each program while it is running.

→ To meet these objectives, the Process scheduler selects an available Process for Program execution on the CPU.

3.2.1 Scheduling Queues

→ There are 3 different types of Queues

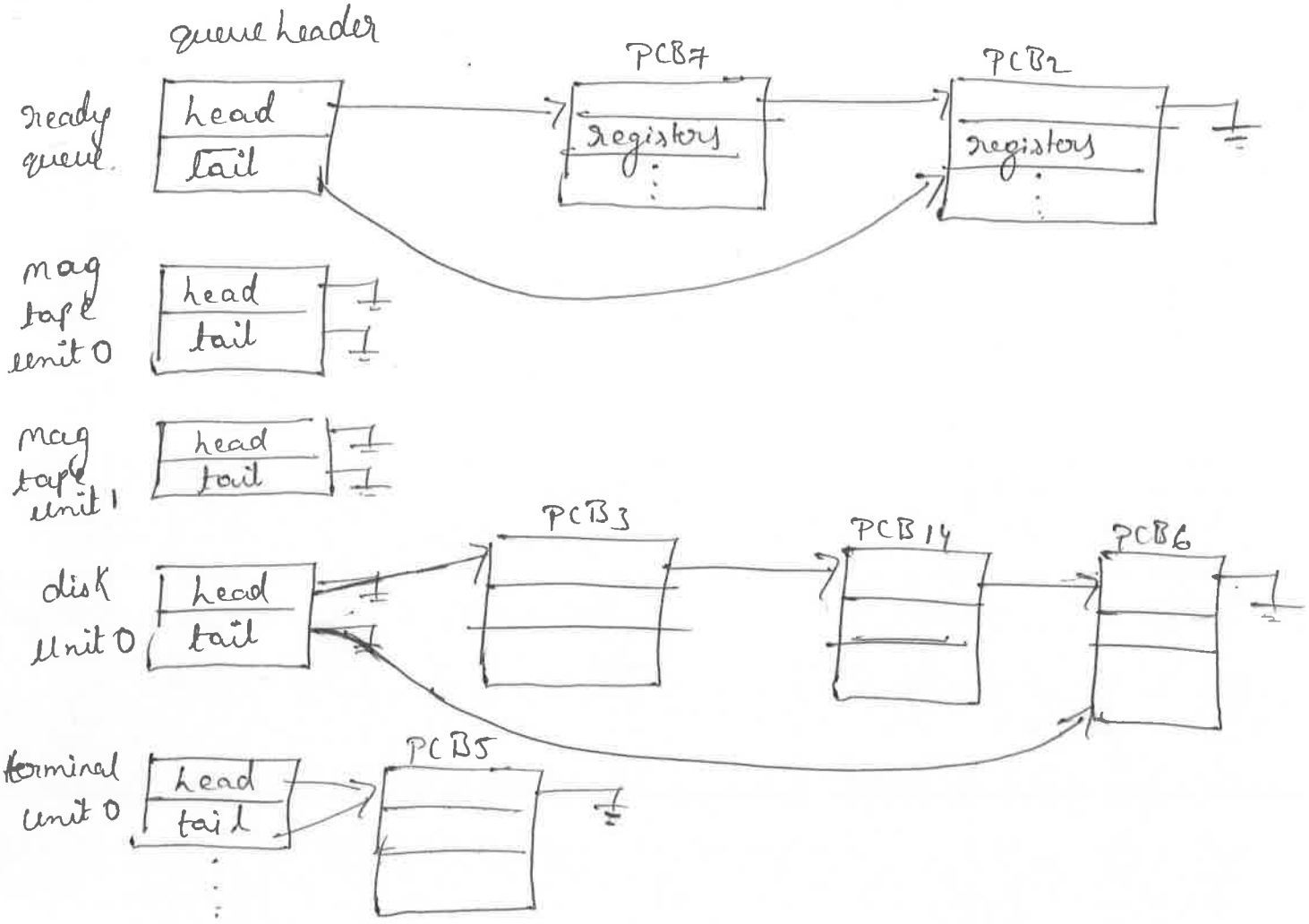
(1) Job Queue :- Set of all Processes in the System (A Processes enters System they are put in a Job Queue)

(2) Ready Queue :- The Processes that are residing in main memory & are ready & waiting to execute are kept on a list.

(3) Device Queue :- The list of Processes waiting for a Particular I/O device is called device queue.

→ Each device has its own queue.

Fig:- The ready Queue & Various I/O device queues



→ These queue is generally stored as a linked list.

→ A ready queue header contains pointers to the first & final PCB's in the list.

→ Each PCB includes a pointer field that points to the next PCB in the ready queue.

→ The list of Processes waiting for a particular I/O devices is called device queue. When the CPU is allocated to a Process it may execute some time & may quit @ interrupted @ wait for the occurence of a particular event like completion of an I/O request.

→ But the I/o may be busy with some processes. In this case the Process must wait for I/o. This will be placed in device queue. Each device as its own device queue.

→ The Process scheduling is represented in a Queuing diagram, Queues are represented by the rectangular box & resources they need are ref by circles.

→ It contain 2 types of Queues

① Ready Queue

② Set of device queue.

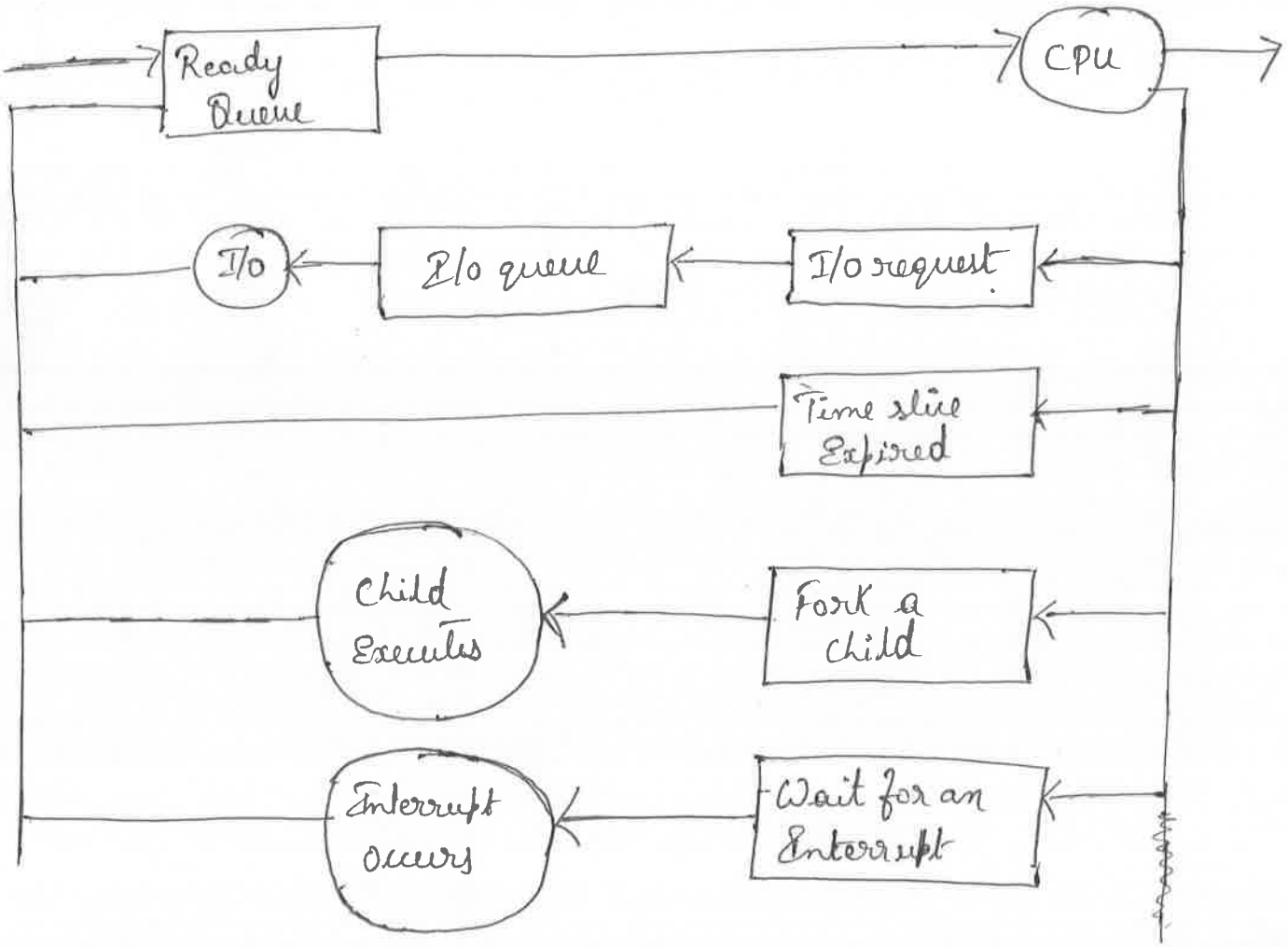


Fig:- Queuing diagram representation of Process Scheduling

→ A new Process is initially put in the ready queue It waits there until it is selected for execution @ is dispatched. Once the Process is allocated to CPU & is executing one of the events could occur.

(1) The Process could issue an I/O request & then be placed in an I/O queue.

(2) The Process could create a new Sub Process & waits for the Sub Process termination.

(3) The Process could be removed from the CPU as a result of an interrupt & be put back in the ready queue.

→ The Process continues this cycle until it terminates it is removed from all queues & has its PCB & resources are deallocated.

(3.2.2) Schedulers :- "Selecting the Process"

→ A Process migrates among the various Scheduling queues throughout its lifetime. The OS must select for Scheduling purposes, Processes from these queues. The Selection process is carried out by appropriate Scheduler.

Types of Scheduler

(1) Short term Scheduler [CPU Scheduler]
(2) Long term Scheduler [Job Scheduler]
(3) Medium term Scheduler.

(1) Short term Scheduler :

→ Selects which process should be executed next & allocates the CPU to one of them
→ It Schedules very frequently
→ Very fast
→ It is Processed at every 10ms
→ For Every 100ms Schedulers are invoked.

(2) Long term Scheduler :

→ Selects which Processes should be brought to Job Pool & loads them into memory for execution.
→ It is invoked Very Infrequently
→ May be Slow
→ Takes more time to Select a Process for execution.
→ This Scheduler is invoked only when a Process leaves the System.
→ Controls the degree of Multiprogramming (The no of Processes in memory)

* Two types of Processes

① I/o bound Process :- Performs I/o operations i.e. Spends more time doing than computation.

② CPU bound Process :- Performs computations i.e Spends more time doing computation but generates I/o request Infrequently.

(3) Medium term Scheduler

→ Some OS such as time sharing system introduce intermediate level of Scheduling

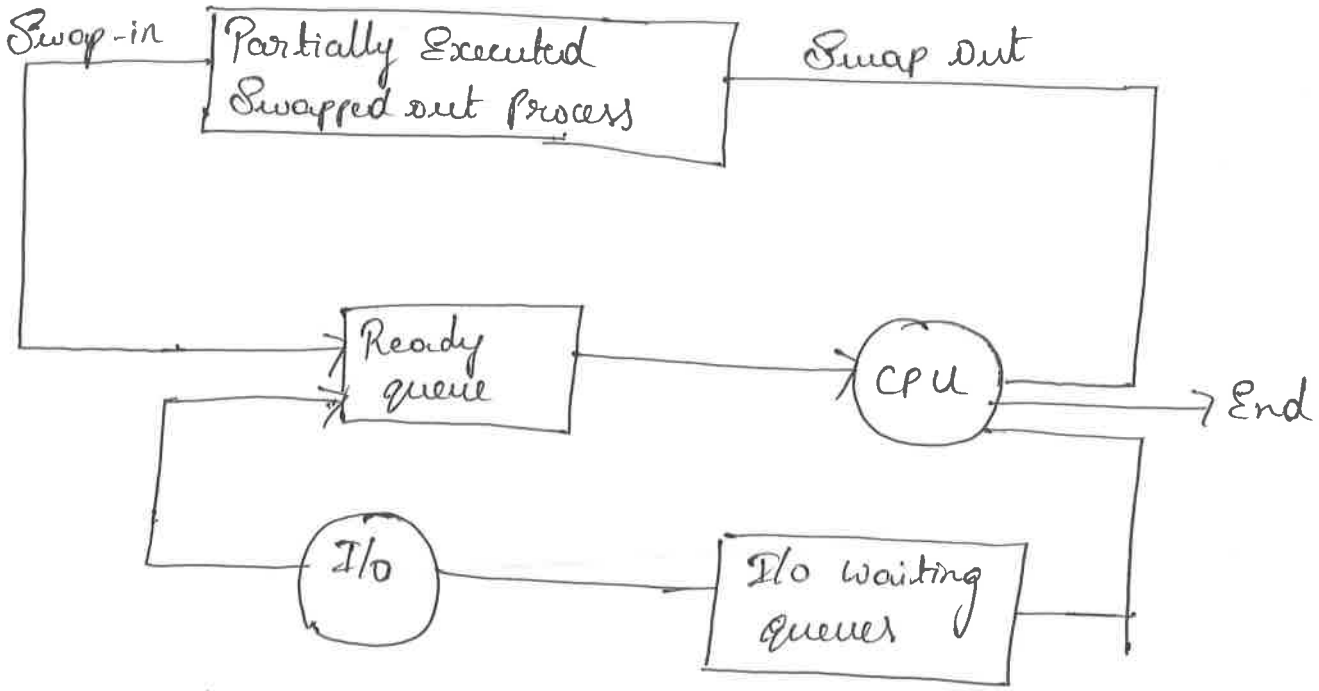


Fig: Addition of medium term Scheduling to the Queuing diagrams

→ Medium term Scheduling is that sometime advantageous to remove processes from memory & thus reduces the degree of multiprogramming.

→ Later this Process can be Introduced into memory & its execution can be Continued where it is left off. This Scheme is called Swapping

→ The process is Swapped out & is later Swapped in by the medium term Scheduler.

→ Swapping is necessary to improve the Process mix.

3.2.3 Context Switch :

→ When CPU Switches to another process, the System must Save the state of the old process & load the Saved Saved state for the new process. This task is known as a context Switch.

→ The Context of a Process is represented in the PCB of a Process it includes the Value of the CPU register, process state & Memory Mgt Information.

→ Context Switch time is pure Overhead → Because the System does no useful work Switching & are highly dependent on the Hardware Support.

→ If the OS is more complex, more work must be done during the Context Switch.

3.3 Operation on Processes

→ Two operations are Performed.

1. Process Creation
2. Process Termination

1. Process Creation :

→ A Process may create new processes Via a create process System call. during the course of execution.

→ Creating process is called a parent process & the new processes are called the children of that Process.

Each of these new processes may inturn create other processes forming a tree of processes.

→ In general, a process will need certain resources to accomplish its task [cpu time, memory files, I/O devices]

→ When a process creates a Sub process, that Sub process may be able to obtain its resources, directly from the OS (or) it may be from Parent process.

→ The Parent may have to partition its resources among its children (or) It may be able to share some resources [Memory/files] among several of its children.

* When a process creates a new process, 2 possibilities exist in term of Execution:

(1) The Parent continues to execute concurrently with its children

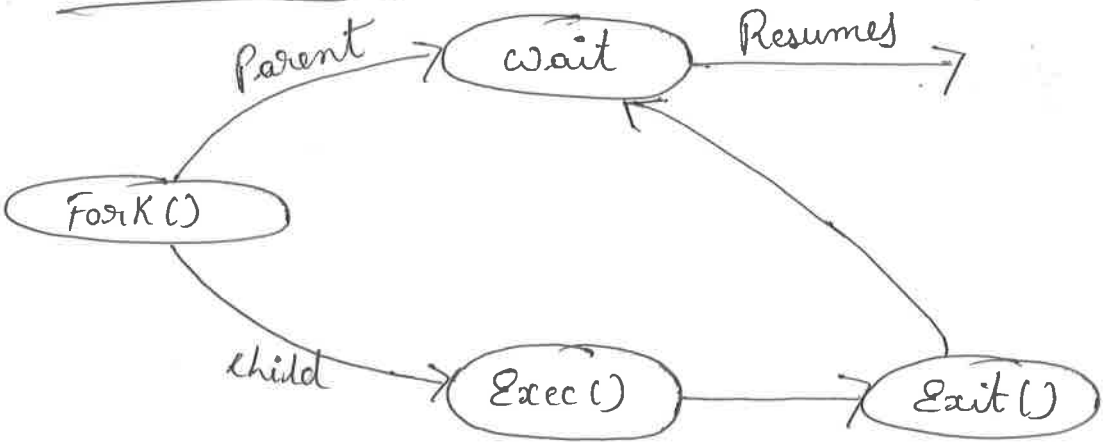
(2) The Parent waits until Some (or) all of its children have terminated.

→ There are also 2 possibilities in terms of address space of the new process

(1) The child process is a duplicate of the Parent process (It has the same program & data as the Parent)

(2) The child process has a new program loaded into it.

Fig:- Process creation



→ A new process is created using ForK() System Call

→ The exec() system call loads a binary file into memory & starts its execution

→ Wait() System call to move itself off ~~fr~~ the ready queue until the termination of child

→ The parent waits for the child process to complete with the Wait() system call. ~~The Parent~~

→ When ~~the~~ child process completes (by either ~~normally~~ ~~or~~ implicitly or explicitly) invoking exit() Sys call the parent process resumes from the wait(). where it completes using the exit() System call.

(2) Process Termination :-

fig :-

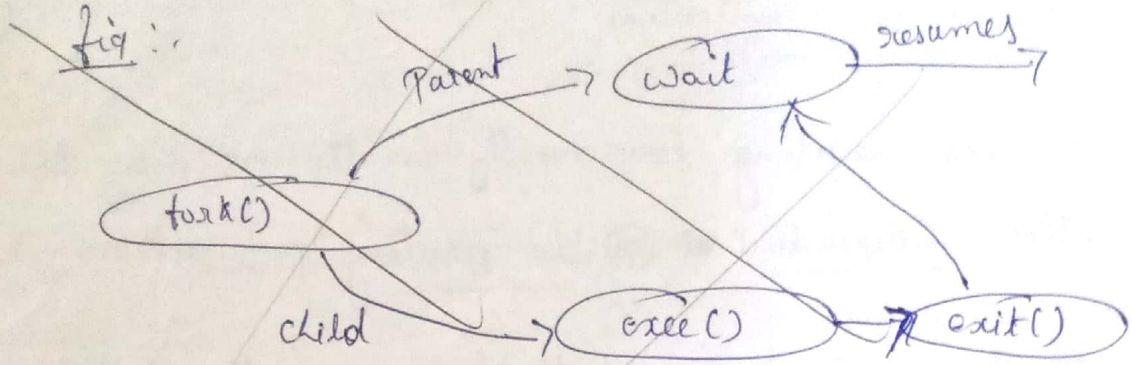


fig: Process creation

2/ Process Termination :-

→ A process terminates when it finishes executing its final stmt & asks the OS to delete it by using the exit() System call.

→ All the resources of the process [physical & Virtual memory, open file & I/O buffers] are deallocated by the OS.

→ A Parent may terminate the execution of one of its children for a variety of reasons such as.

(1) The child has exceeded its usage of some of the resources that it has been allocated.

(2) The task assigned to the child is no longer required.

(3) The Parent is exiting, & the OS does not allow a child to continue if its Parent terminates.

Some System do not allow a child to exist if its parent has terminated. In such System, if a process terminates (either normally or abnormally) then all its children must also be terminated. This phenomenon is referred to as Cascading termination.

Scanned with CamScanner

Inter Process Communication :-

" Processes executing concurrently in the OS may be either independent or co-operating 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 co-operating if it can affect or be affected by other processes executing in the system.
i.e. any process that shares data with other processes is a co-operating process.

There are several Reasons for Providing an environment that allows process cooperation.

① Information sharing :- Since several users may be interested in the same piece of information (for instance a shared file), we must provide an environment to allow concurrent access to such Information

② Computation Speed up :- If we want a particular task to run faster, we must break it into subtasks, each speed up can be achieved only if the computer has multiple processing elements [such as CPU / I/O channels]

Scanned with CamScanner

(3) Modularity :- We may want to construct the system in a modular fashion, dividing the system function into seperate process / threads.

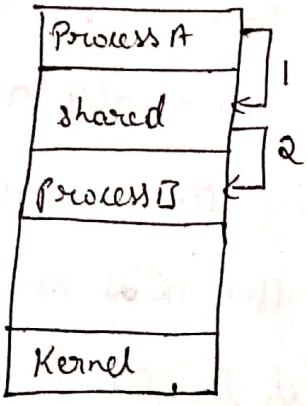
(4) Convenience :- Even an individual user may work on many tasks at the same time, ie user may be editing, printing & compiling in parallel.

→ Co-operating Processes require an IPC mechanism that will allow them to exchange data & Inf.

There are 2 fundamental models of IPC

1. Shared Memory
2. Message Passing.

(1) Shared - Memory System



→ IPC using memory requires communicating processes to establish a region of shared memory

→ Processes can then exchange info by reading & writing data to the shared region.

→ Shared memory allows maximum speed & convenience of communication.

→ In shared memory system, System calls are required only to establish shared memory region. Once shared memory is established, all accesses are treated as

Scanned with CamScanner

routine memory accesses & no assistance from Kernel is required.

→ To clearly illustrate the concept of co-operating processes, Let's consider the Producer-Consumer problem.

→ A producer process produces info that is consumed by a Consumer Process.

→ The producer-Consumer problem also provides a useful metaphor for the client-Server Paradigm. We generally think of a Server as a producer & a client as a Consumer.

Eg:- Web Servers produces HTML files & images, which are consumed by a client Web-browser.

→ One Soln to the Producer Consumer problem uses a shared memory. To allow producer & Consumer Processes to run Concurrently, We must have available a buffer of items that can be filled by the producer & emptied by the Consumer. This buffer will reside in a region of memory that is shared by the Producer & Consumer Processes.

→ A Producer can produce one item while the Consumer is consuming another item. The P & C must be Synchronized, so that the Consumer does not try

Scanned with CamScanner

to consume an item that has not yet been produced.

→ 2 types of buffers can be used.

① Unbounded → Places no practical limit on the size of the buffer,

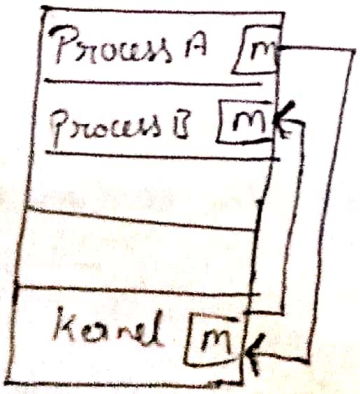
~~Bounded~~ → The consumer may have to wait for new items, but the producer can always produce new items.

② Bounded :→ Assumes a fixed buffer size

Here consumer must wait if the buffer is empty & the producer must wait if the buffer is full.

→ Buffer may either be provided by the OS (or) by explicitly coded by the application ~~prog~~ with the use of shared memory.

② Message Passing System



→ Another way for co-operating processes to communicate with each other via a msg passing facility.

→ This method allows the processes to exchange information

→ The responsibility for providing communication rest with the OS itself

Scanned with CamScanner

→ This mechanism allows Processes to communicate & to Synchronize their actions without sharing the address space.
Eg:- chat pgm used on word wide web

→ A msg passing facility provides at least 2 operations
* Send (message)
* receive (message)

→ A msg sent by a Process can either be fixed @ Variable Size.

→ If Process P & Q want to communicate, they must Send msgs to & receive msgs from each other a communication link must exist b/w them.

This link can be implemented in variety of ways here we are logically implementing a link & the Send() / receive() operations.

① Direct / Indirect comm (Naming)
② Synchronous / Asynchronous comm (Synchronous)
③ Automatic / Explicit buffering (Buffering)

Naming:- Processes that want to communicate must have a way to refer to each other they can use either direct / Indirect communication.

Under direct communication → Each Process that wants to communicate must explicitly name the recipient/ Sender of the communication

Scanned with CamScanner

→ In this scheme, Send() & receive() primitives are defined as:

① Send (P, message) → Send a message to process P

② Receive (Q, message) → Receive a message from process Q

→ A comm link in this scheme has the following properties

① A link is established automatically b/w every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.

② A link is associated with exactly two processes.

③ Between each pair of processes, there exists exactly one link.

→ This scheme exhibits symmetry in addressing i.e. both the sender & receiver process must name the other to communicate.

→ In Assymmetric only the sender names the recipient the recipient is not required to name the sender.

① Send (P, msg) → Send a msg to Process P

② Receive (id, msg) → Receive a msg from any process the variable id is set to the name of the process with which comm has taken place.

Scanned with CamScanner

Indirect communication:- The messages are sent to & received from mailbox @ Ports.

→ In this scheme, a process can communicate with some other process via a no ∧ diff mail boxes.

→ 2 Processes can communicate only if the processes have a shared mailbox.

Send (A, msg) → Send a msg to mail box A

Recieve (A, msg) → Receive a msg from mailbox A

→ In this scheme, a Comm link has the full properties

(1) A link is established b/w a pair of processes only if both ~~member~~ members of the pair have a shared mail box.

(2) A link may be associated with more than 2 Processes.

(3) B/w each pair of communicating processes, there may be no ∧ diff links with each link corresponding to one mail box.

The operating system then must provide a mechanism that allows a Procen to do the full

(1) Create a new mailbox

(2) Send & receive msg through the mailbox

(3) Delete a mailbox

Scanned with CamScanner

2] Synchronization :-

Communication btw Processes takes place through calls to Send() & receive() primitives. There are diff design options for implementing each primitive. Message passing may be either blocking / nonblocking also known as Synchronous / Asynchronous.

① Blocking Send :- The Sending process is blocked until the msg is received by the receiving process or by the mailbox

② Non blocking Send :- The Sending process sends the msg & resumes operation

③ Blocking receive :- The receiver blocks until a msg is available

④ Non blocking receive :- The receiver retrives either a valid msg or a null.

3] Buffering :-

Whether the communication is direct or Indirect msgs exchanged by communicating processes reside in a temporary queue, Such queue can be implemented in 3 ways.

① Zero capacity :- The queue has a maximum length of zero, thus, the link cannot have any messages waiting in it. Here Sender must block until the recipient receives the msg.

Scanned with CamScanner

(2) Blocking capacity:- The queue has finite length n; thus at most n messages can reside in it. If the queue is not full when a new msg is sent, the msg is placed in the queue, & the sender can continue execution.

(3) Unbounded capacity: The queues length is potentially infinite; thus any no of msg can wait it. The sender never blocks.

Scanned with CamScanner