UNIT - I
OPERATING SYSTEM OVERVIEW

COMPUTER SYSTEM OVERVIEW.

An operating system is a software program designed to act as an interface between the user and the computer. It is a collection of program and utilities. It controls the computer hardware, manages the system resources & supervises the interaction between the system & the user.

eg) DOS, LINUX, windows etc.

BASIC ELEMENTS.

Processor:
- Controls the operation of the computer
- Performs the data processing function.
- Referred to as the central processing unit (CPU)

Main Memory.
- Volatile
- Contents of the memory is lost when the computer is shut down.
- Referred to as real memory or primary memory.
- Programs to be executed is taken into the main memory.

I/O Modules:

Moves data between the computer & external environments such as
- storage
- Communications equipment
- terminals.

System Bus.

Provides communication among processor, main memory and I/O modules.

INSTRUCTION EXECUTION

Consists of 4 steps.

Fetch : fetches an instruction from memory
Decode : Operands to be fetched from mry.
Execute : Execution of instruction
Write : After execution, result is written to storage or memory.

INTERRUPTS

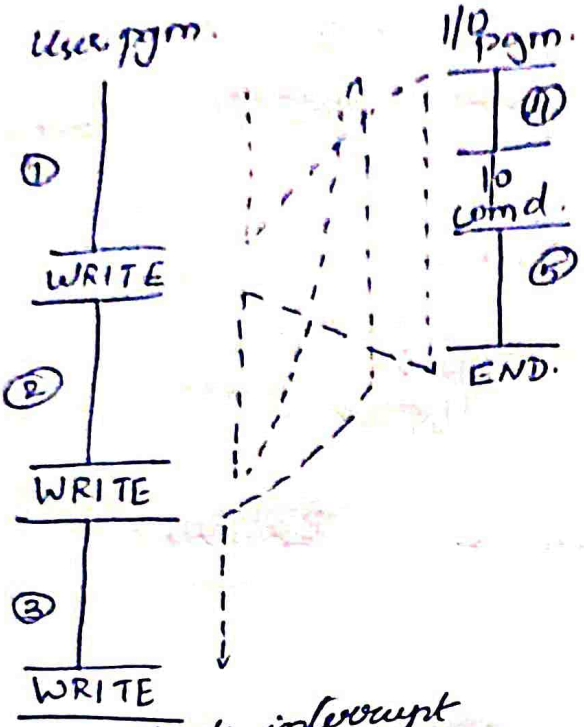
- Interrupt the normal working of the processor.
- This may occur by hardware & by software.

H/w trigger an interrupt by sending a signal to CPU through the bus.
S/w trigger an interrupt by a system call.

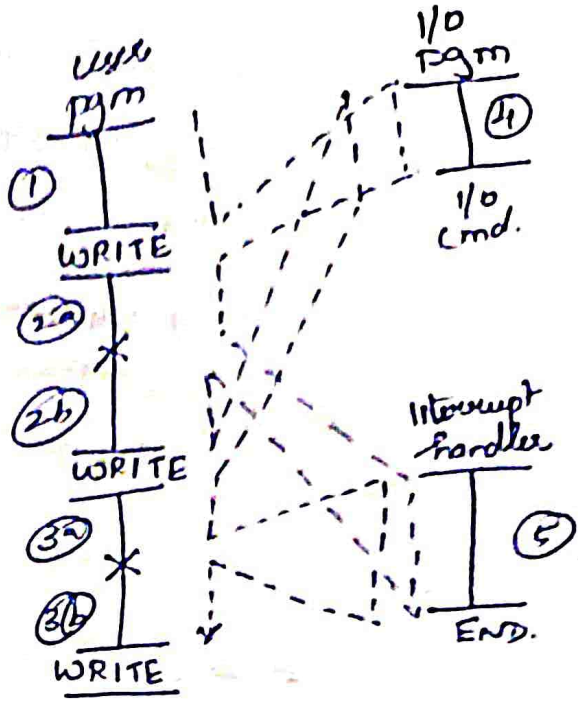
Classes of interrupts.

1) Program: Generated by some condition that occurs as a result of an instruction execution, like arithmetic overflow, / by zero. etc.
2) Timer: Generated by a timer within the processor.
3) I/O : Generated by I/O controller.
4) Hardware failure: failure such as power failure. etc.

Flow of control without interrupts



(a) without interrupt



(b) with interrupt.

A user pgm performs a series of WRITE calls with processing. Code segments 1, 2 & 3 does not involve I/O routine. and codes 4 & 5 are I/O routine.

In (a) fig. there is no interrupt. once the program encounters a WRITE cmd, the I/O operations are performed and after completion of I/O again the program is executed.

In fig (b). the program is interrupted at 2a. Once interrupt is occured, the interrupt handler is executed. After that execution again the main program will be executed.

At the time of I/O program execution the main pgm might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.

After the completion of I/O, it sets a flag indicating the success or failure of the operation.

Interrupt handler :- is a program that determines the nature of the interrupt and performs whatever actions are needed.

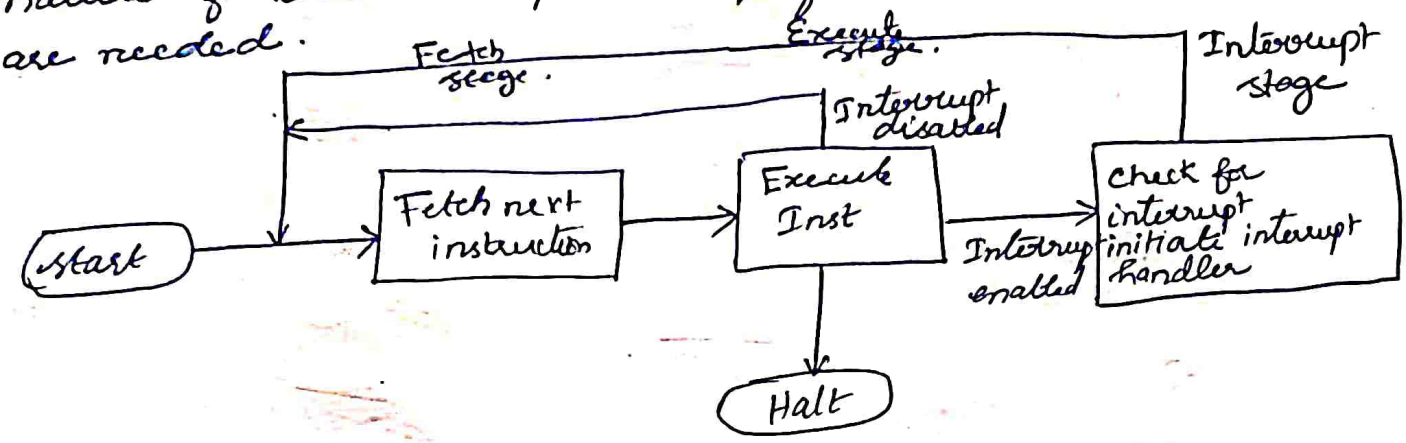


fig:- Instruction cycle with interrupts.

Steps :

1. Device issues signal to processor
2. Processor finishes execution of current instruction.
3. Processor signals acknowledgement of interrupt.
4. Processor pushes PSW & PC onto control stack.
5. Processor loads new PC value based on interrupt.
6. Interrupt handler saves all information in the registers on the stack.
7. Processes the interrupt.
8. After completion of interrupt, the saved register values are retrieved from the stack & restored to the register.
9. Restore the PSW & PC values from the stack.

Multiple interrupts.

An interrupt occurs while another interrupt is being processed. eg). receiving data & printing result at the same time.

Handled by two approaches.

- disable interrupts while an interrupt is being processed.
- Use priority scheme.

MEMORY HIERARCHY

The memory hierarchy is according to speed and cost. The higher levels are expensive and fast.

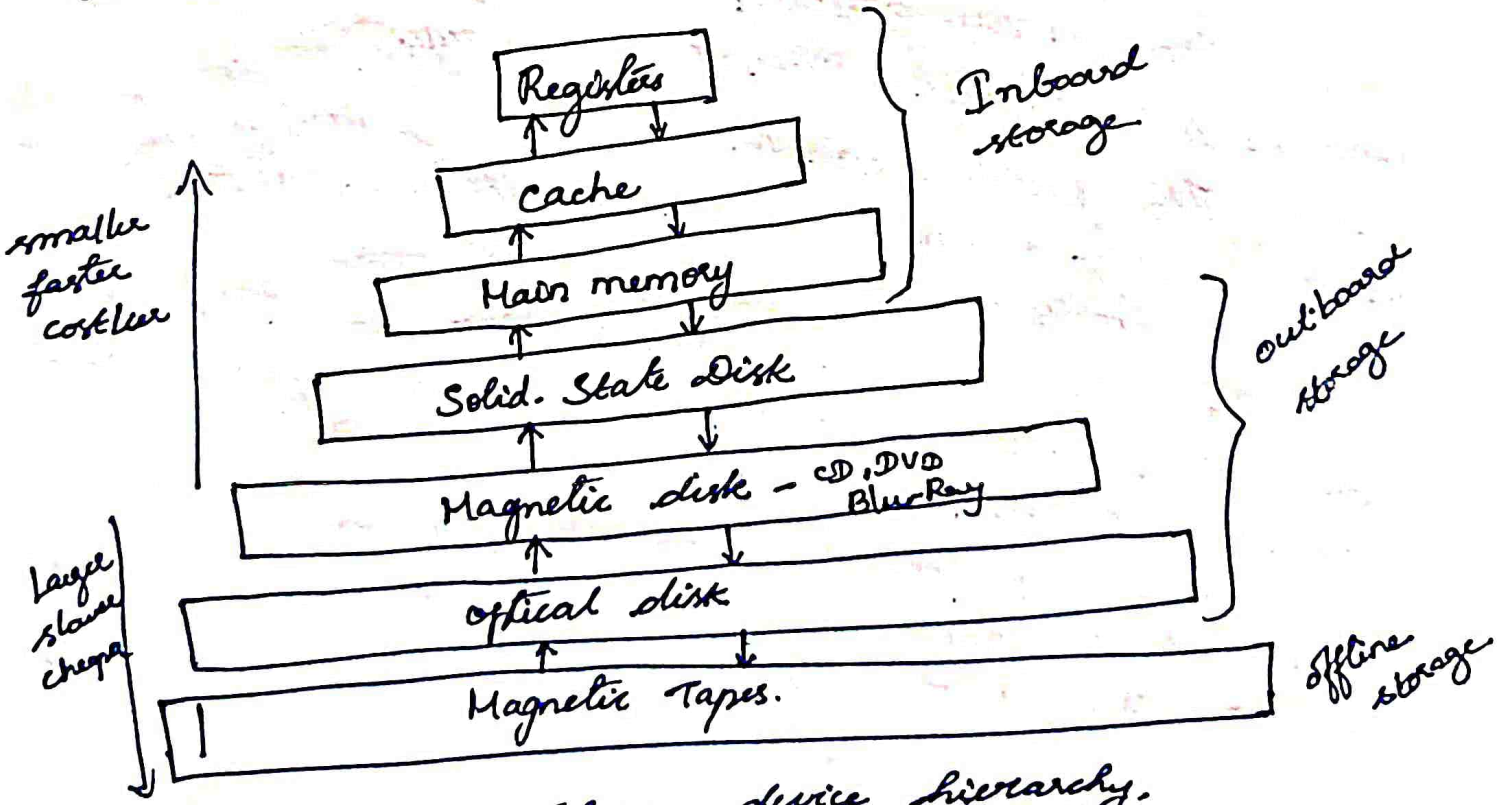


fig:- Storage-device hierarchy.

Top three level of memory are constructed using semiconductor memory.

Storage system above the electronic disk are volatile and that are below are non volatile.

volatile :- Loses its contents when power to the device is removed.

Non volatile :- Data is not lost even if the power is removed.

CACHE MEMORY.

Processors are generally able to perform operations on operands faster than the access time of large capacity main memory. This problem can be alleviated by introducing a small block of high speed memory called a cache between the main memory & the processor.

The idea of cache memory is that some active portion of a low speed memory is stored in duplicate in a higher speed cache memory.

When a my request is generated, the request is first presented to the cache memory, and if the cache cannot respond, the request is then presented to the main my.

cache miss :- If the requested data is not present in the cache it is referred to as cache miss.

cache hit :- If the data is present in the cache may it is referred to as cache hit.

When a read operation results in cache miss, the data is retrieved from the main memory & then it is copied to the cache. For a read operation, average cycle time is,

$$t_{eff} = t_{cache} + (1-h)\, t_{main}.$$

h - probability of a cache hit.

(1-h) - probability of a cache miss.

TAGS	DATA							
0117X	35	72	55	30	64	23	16	14
7620X	11	31	26	22	55	+++		
3656X	71	72	44	50	---			
⋮								
1741X	33	35	07	66	...			

01173
Address

30
Data

fig:- A typical cache memory (Direct mapping)

The tag 0117X matches address 01173, so the cache returns the item in the position X=3 of the matched block.

Cache memory organizations:

- Fully associative mapping
- Direct mapping
- Set-associative mapping
- Sector mapping.

Fully associative mapping:

Here both memory address and data are stored in the cache together.

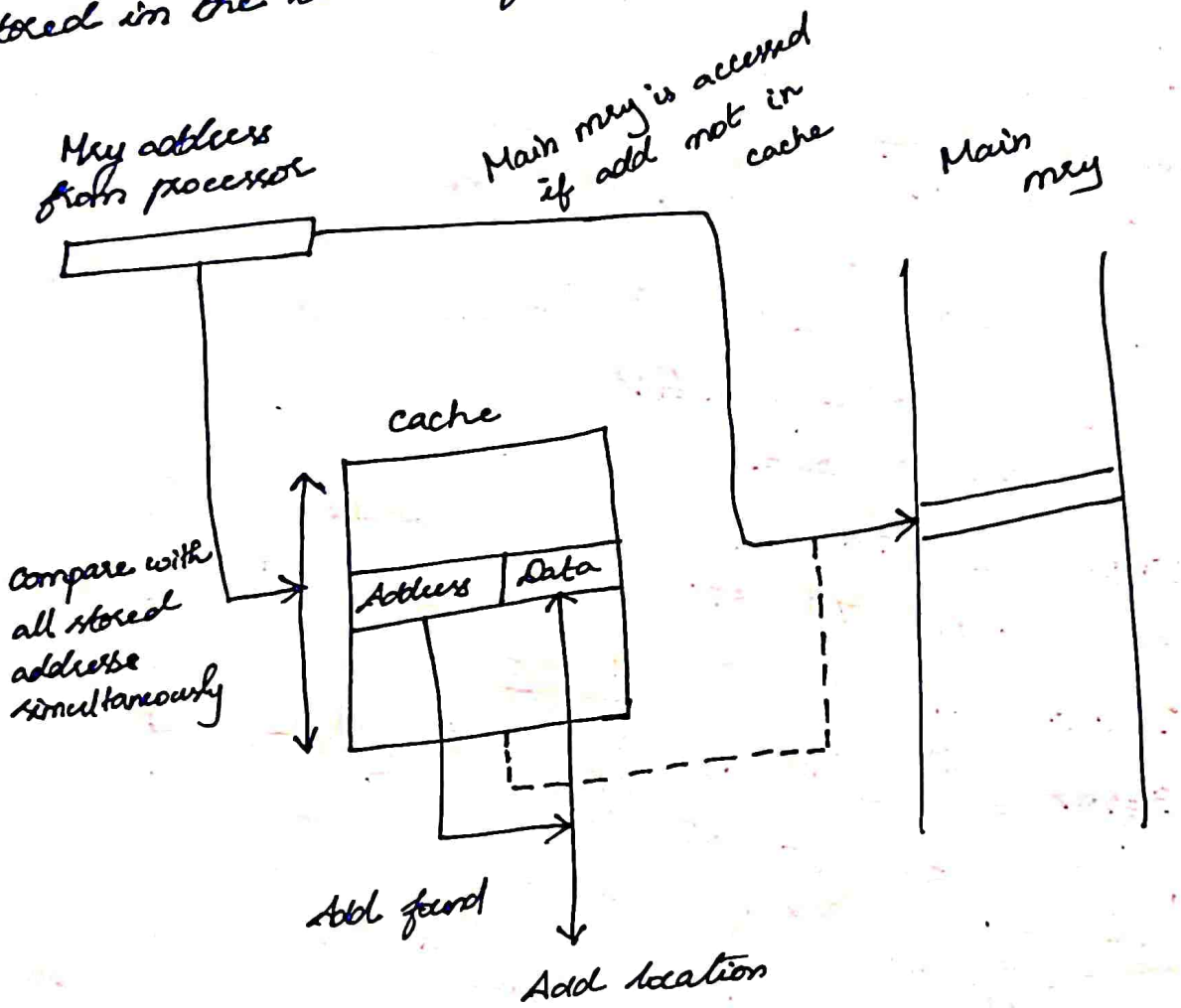


fig:- Cache with fully associative mapping.

① Cache will store address and data simultaneously.

② Processor will search for the data along with its address in the cache.

③ If there is an address match, the corresponding data will be fetched.

④ If not the processor will search for the data from the main memory.

Direct mapping:

The fully associative cache is expensive to implement because of requiring a comparator with each cache location.

In direct mapping, the address from the processor is divided into 2 fields, a tag & an index. Here, the cache consists of normal high speed random access memory & each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address.

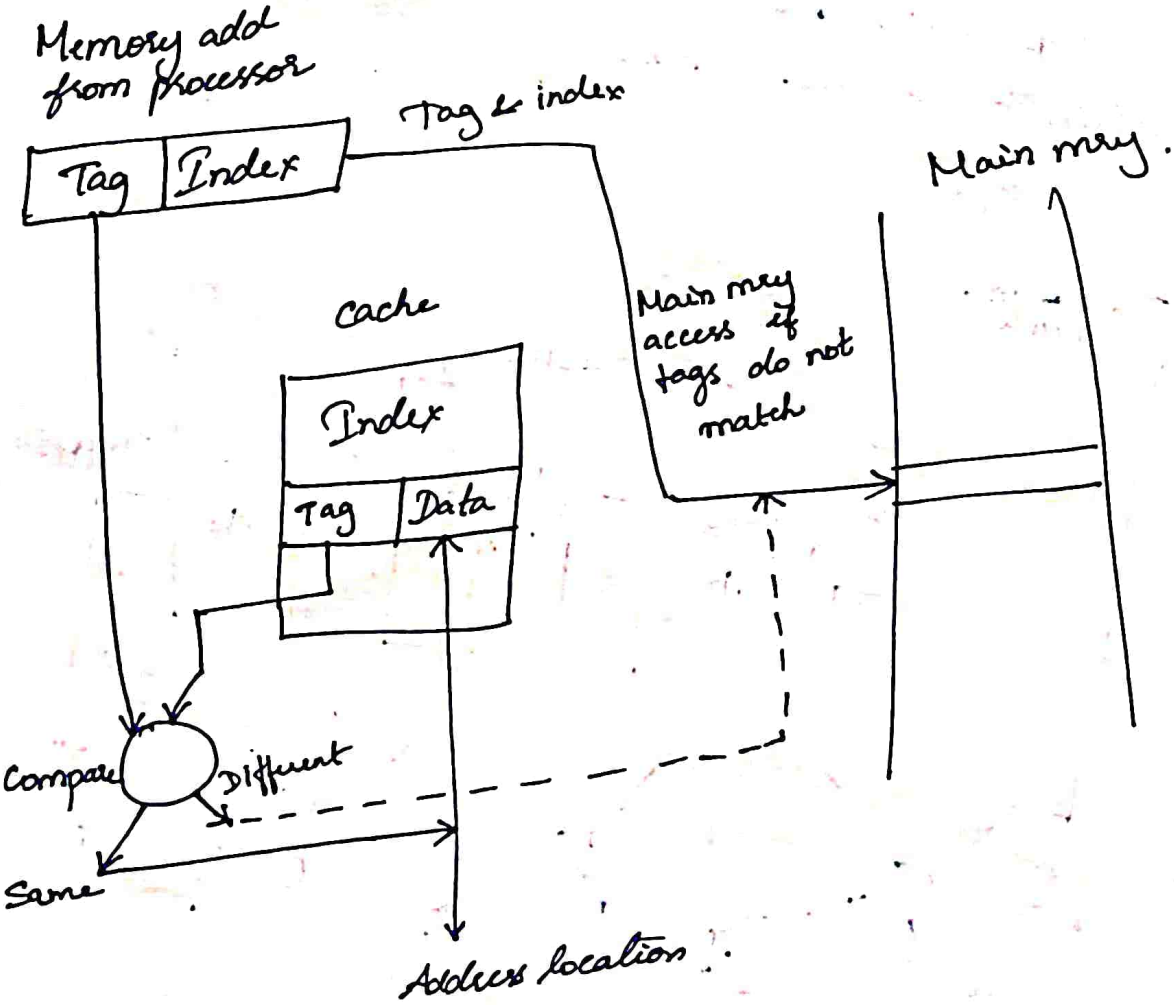
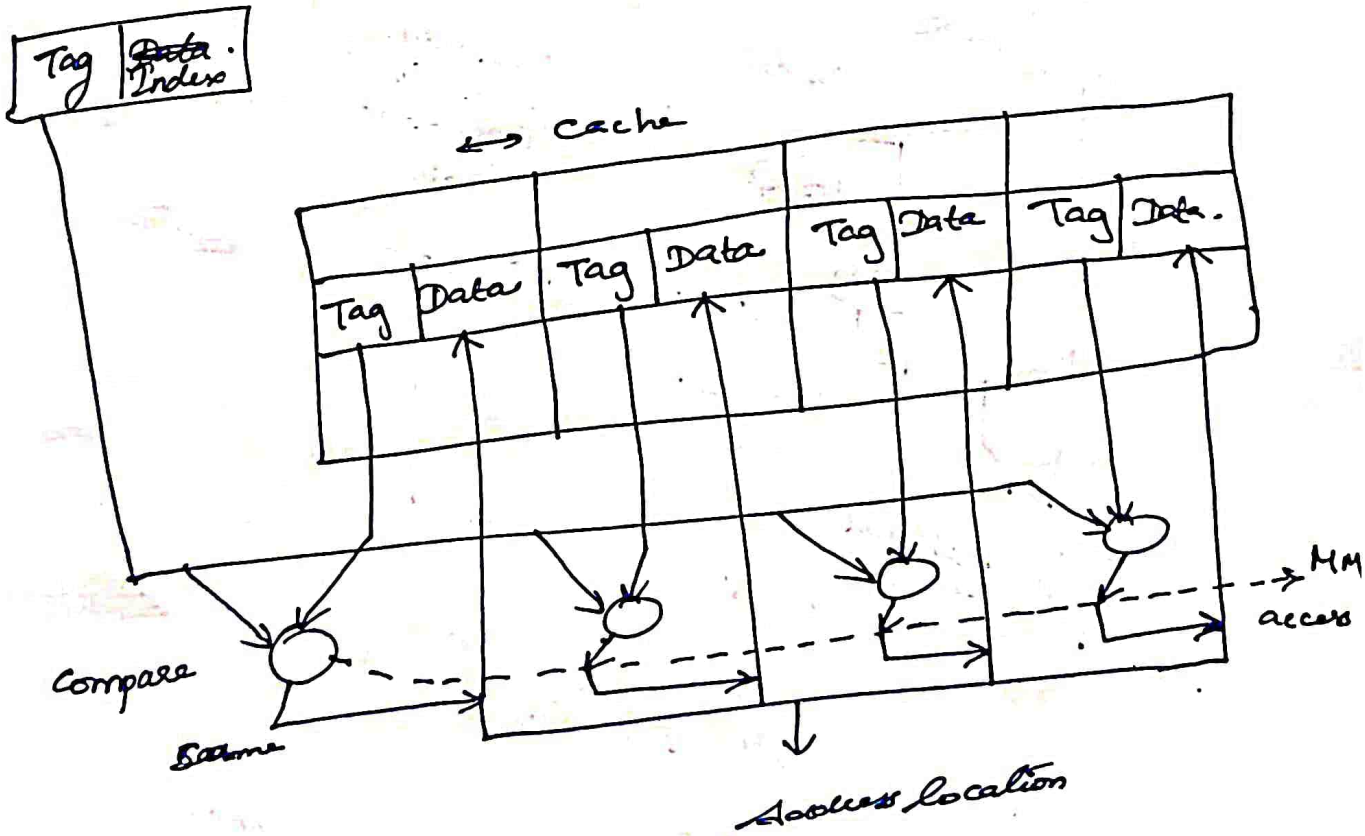


fig:- Cache with direct mapping.

Set associative mapping:

It is combination of direct & associative mapping. In direct scheme, all words stored in the cache must have different indices. The tags may be same or different. In fully associative scheme, blocks can displace any other block so can be placed anywhere, but cost is

Set associative mapping allows limited no. of blocks, with the same index & different tags. The cache is divided into set of blocks. Each block in each set has a stored tag which, together with the index, completes the identification of the block.



Sector mapping:

The main memory & the cache are divided into sectors. Each sector is composed of number of blocks. Any sector in the main memory can map into any sector in the cache and a tag is stored with each sector in the cache to identify the main memory sector address.

Cache performance:

The performance of a cache can be quantified in terms of the hit and miss rates, the cost of a hit & the miss penalty.

Access time = (hit cost) + (miss rate) * (miss penalty)

= (fast mry access time) + (miss rate) * (slow mry access time).

miss penalty :- additional cost of replacing a cache line with the desired data.

Fetch & write mechanism:

Fetch policy:

3 strategies for fetching bytes or blocks of data from the main memory to the cache memory.

① Demand fetch ② Selective fetch ③ Prefetch.

Instruction & data caches:

The main memory holds both program instruction and data together. Likewise the cache can be organized in the same fashion. This is called a unified cache.

Write Operations:

A read operation does not affect the data present in the cache or in the main memory. But a write operation need to change some data in the main memory and its copy should be updated to the cache memory. There are 2 mechanisms used for this.

① write through mechanism

② write-back mechanism.

write through mechanism:

Every write operation to the cache is repeated to the main memory, at the same time. The additional write operation to the main memory take much longer than to the cache & will dominate the access time for write operations. The average access time of write through with transfer from main mry to the cache on all misses (read & write) is,

$$t_a = t_{cache} + (1-h)\,t_{trans} + W(t_{main} - t_{cache})$$

$$= (1-W)\,t_{cache} + (1-h)\,t_{trans} + W t_{main}$$

w - fraction of write references.

t_{trans} - time to transfer block to cache

Write back mechanism.

The write operation to the main memory is only done at block replacement time. The block displaced by the incoming block might be written back to the main mem irrespective of whether the block has been altered.

Average time is,

$$t_a = t_{cache} + (1-h)\, t_{trans} + (1-h)\, t_{trans}$$

① $(1-h)\, t_{trans}$ term is due to fetching a block from main mem.

② $(1-h)\, t_{trans}$ writing back a block..

Replacement policy of cache memory.

A replacement algorithm is used to replace the cache memory with another block of data.

i) Random replacement algorithm
ii) First in first out replacement algm
iii) Least recently used algorithm

Direct MEMORY ACCESS (DMA)

DMA is a feature of computer systems that allows certain H/w subsystem to access main memory (RAM), independent of the central processing unit (CPU).

A direct memory access is an operation in which data is copied from one resource to another resource in a computer system without the involvement of the CPU.

The task of DMA controller is to execute the copy operation of data from one resource to another. The copy operation can be performed from

- I/O device to mry
- memory to IO
- mry to mry
- I/O to I/O device.

A DMAC is an independent resource of a computer system added for the concurrent execution of DMA operations. The first two operation modes are 'read from' and 'write to' transfers of an I/O device to the main memory, which are the common operation of a DMAC. The other two operations are slightly more difficult to implement and most DMAC do not implement device to device transfers.

The DMAC replaces the CPU for the transfer task of data from the IO device to the main mry (vice versa) which otherwise would have been executed by the CPU using the programmed IO (PIO) mode. PIO is realized by a small instruction sequence executed by the processor to copy data. The 'memcpy' fn supplied by the system is such a PIO operation.

DMAC is a master/slave resource on the system bus, because it must supply the address for the resources being involved in a DMA transfer. It requests the bus whenever a data value is ready for transport, which is signaled from the device by the REQ signal

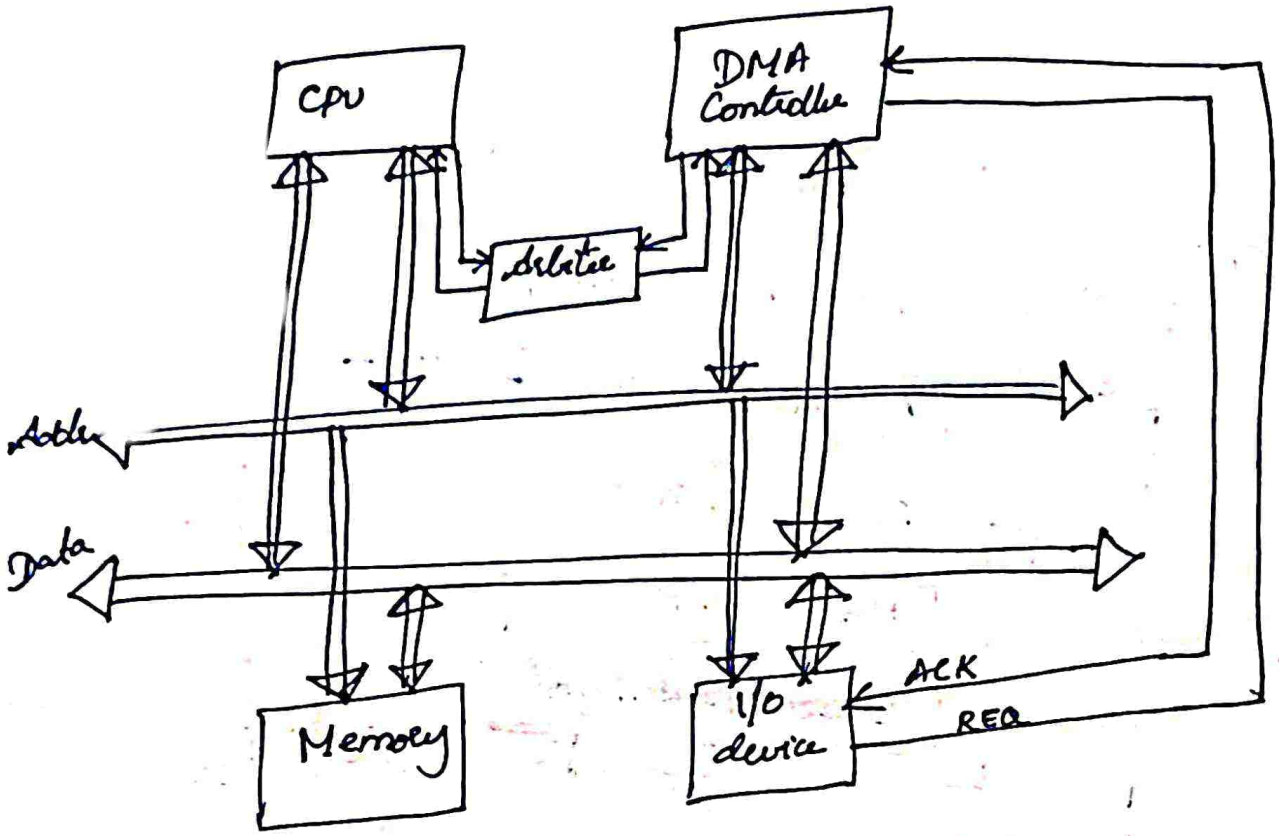


Fig:- Logical structure of a system with DMA

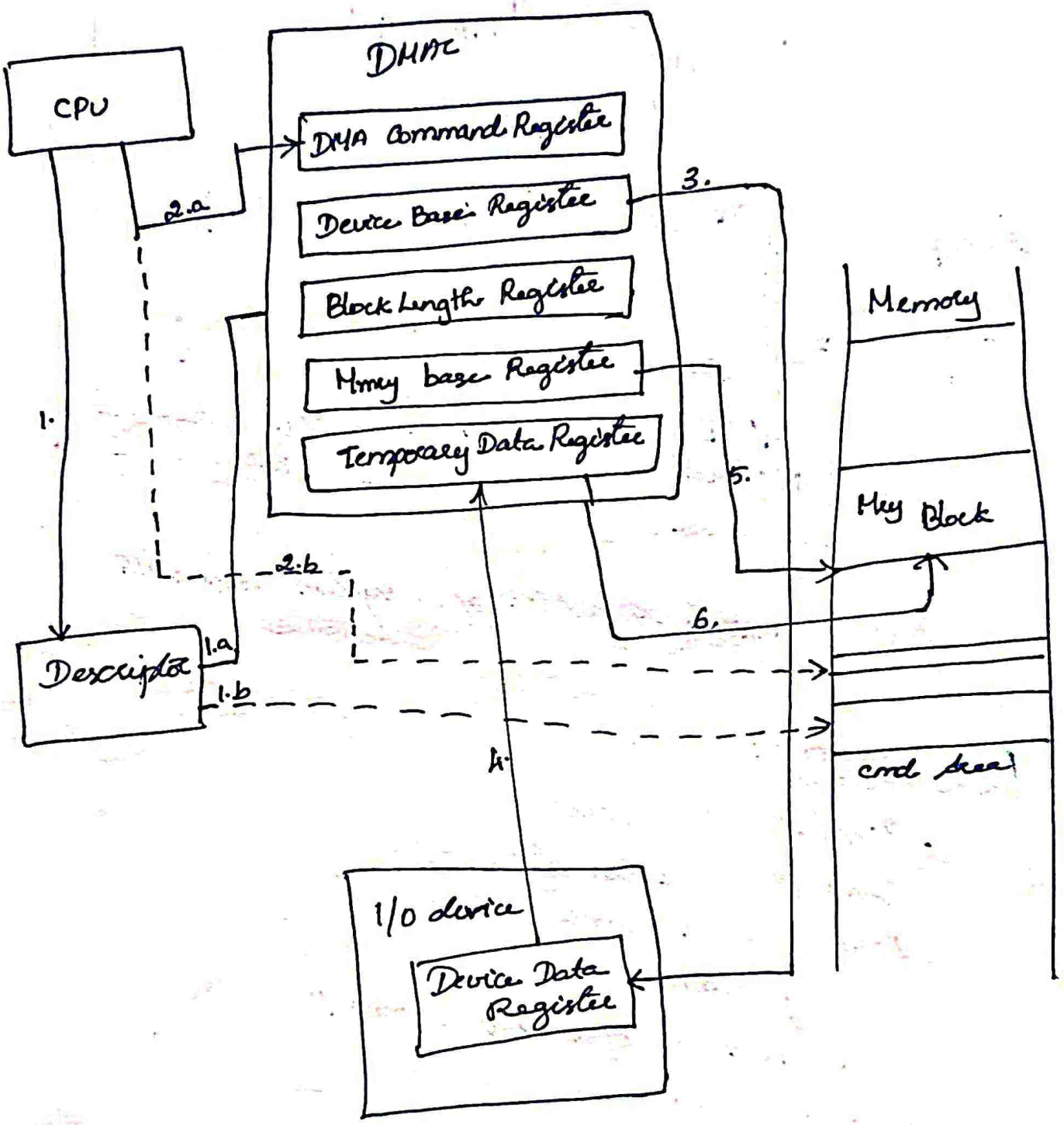
DMA Operations:

- Single block transfer
- Chained block transfer
- Linked block transfer
- Fly-by transfers

All these modes of operations normally access the block of data in a linear sequence.

Execution of a DMA operation (single block transfer):

1. The CPU prepares the DMA operation by the construction of a descriptor, containing all necessary information for the DMAC to independently perform the DMA operation.



2.a) The CPU initializes the operation by writing a cmd to a register in the DMAC & (2.b) to a special assigned mem area, where the DMAC can poll for the command & the descriptor

3) Then the DMAC addresses the device data register

4) Then it reads the data into the temporary data register.

5) In another bus transfer cycle, it addresses the memory block.

6) Then writes the data from the temporary data register to the mem block.

The DMAC increments the mem block address & continue this loop until the block length is reached.

The completion of the DMA operation is signalled to the processor by sending an IRQ signal or by setting a mem semaphore variable, which can be tested by the CPU.

MULTIPROCESSOR & MULTICORE ORGANIZATION

A single-processor system have only one main CPU. A multiprocessor systems have more than one processor in close communication, sharing the computer bus, clock, mem & peripheral devices.

Advantages of multiprocessor:

1. Increased throughput.

By increasing the number of processors, more work can be done in less time. The speed up ratio with N processors is not N, it is less than N.

2. Economy of scale:

Share peripherals, mass storage, power supplies and thus save more money. If several pgms operate on the same data set, it is cheaper to store those data on one disk and to have all the processors share them.

3. Increased reliability :

If fns can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. In 10 processors if one fails other 9 will work to complete the process.

The ability to proceed the service providing proportional to the level of surviving hardware is called graceful Degradation. Such systems go beyond graceful degradation are called fault tolerant. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed & corrected.

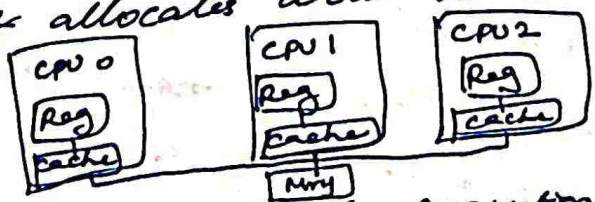
Types.

* Symmetric multiprocessing (SMP)

Each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

* Asymmetric multiprocessing (ASMP)

Each processor is assigned a specific task. A master processor controls the system, the other processor either look to the master for instruction or have predefined tasks. This defines a master-slave relationship. The master processor schedules & allocates work to the slave processors.



(SMP)

MULTICORE SYSTEMS.

A multicore processor is a single computing component that has two or more independent cores or processing unit. These core read & perform program execution on CPU.

A multicore processing unit can execute multiple instructions at the same time.

The multiple cores are integrated into a single IC die, or onto multiple dies but in a single chip package.

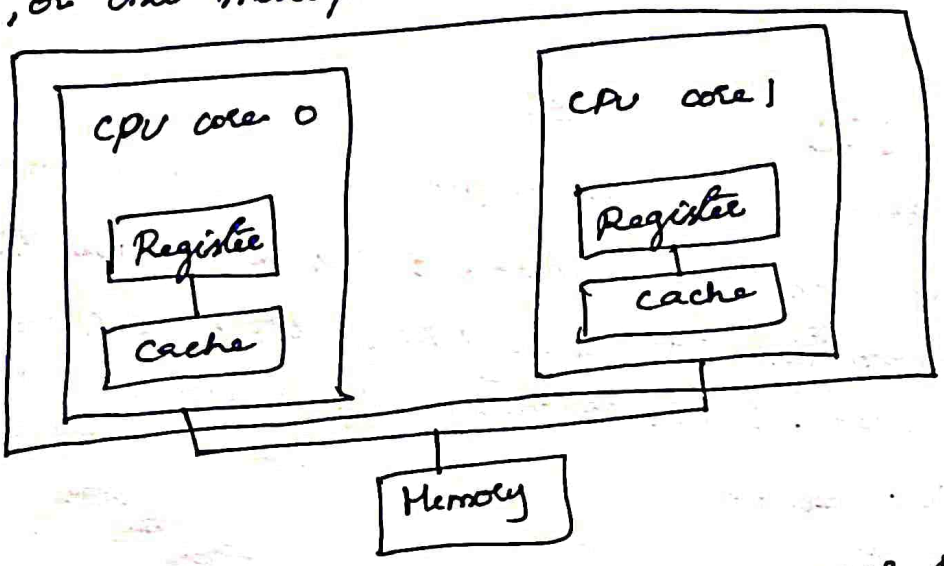
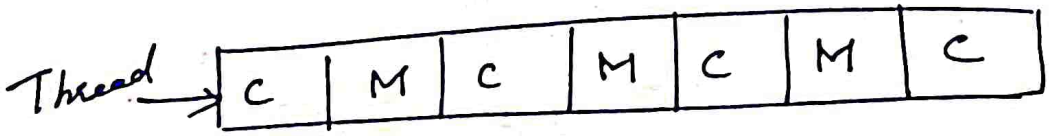


fig:- A dual core design with 2 cores placed on same chip.

Usually SMP systems have allowed several threads to run concurrently by providing multiple physical processors, also recent practice in a system hardware has been place to multiple processor cores on physical chip, result is multicore processor.

Each core contains & maintains its architecture state and thus appears to the OS to be separate physical processor.

When a processor access memory, it spends a significant amount of time waiting for data to be available. This process is called as memory stall.



C - Compute cycle

M - Mky stall

To overcome this, two hardware threads are assigned to each core. If one thread waits for memory, the core can switch to another thread.

Types:

* Coarse-grained:

A thread executes a processor until a memory stall occurs, then the processor switch to another thread to begin execution. Cost of switching b/n threads is high. After new thread begin execution it begin the pipelining with its instruction.

* Fine-grained:

Multithread switches b/n threads at a much finer level of granularity, at boundary of an instruction cycle. The architectural design includes logic for thread switching. cost of switching b/n threads is less.

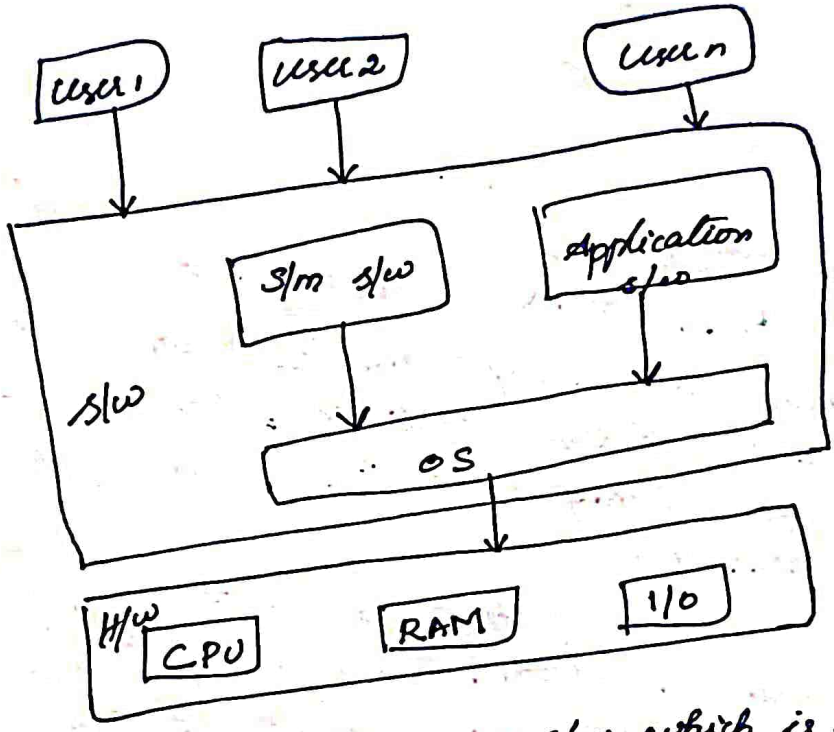
A multithreaded multi core processor actually requires two different levels of scheduling.

① Runs on each hardware thread logical processor.

② How each core decides which hardware thread to run.

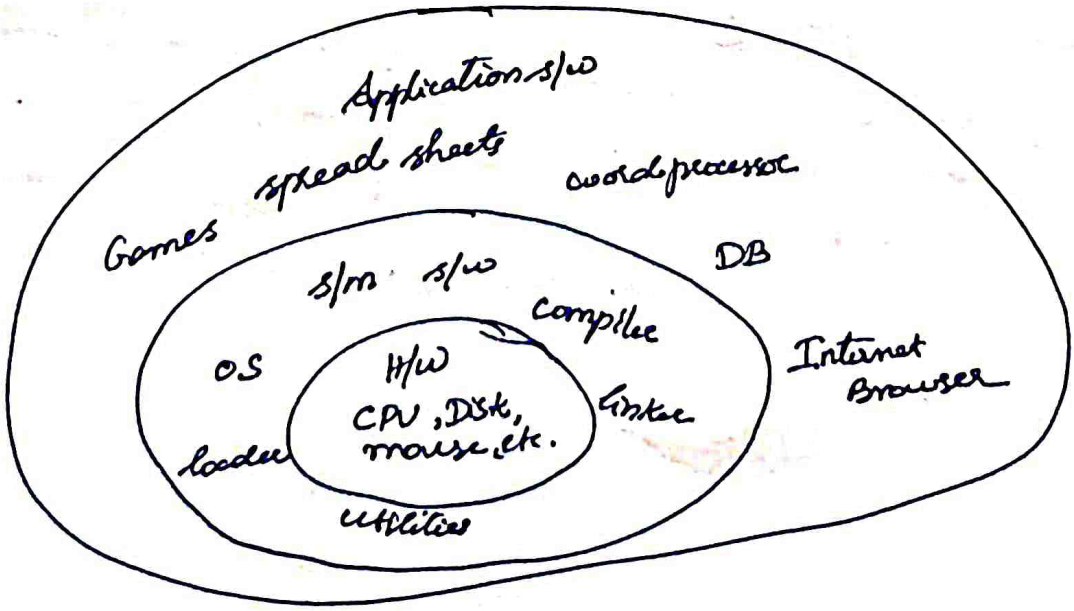
OPERATING SYSTEM OVERVIEW

An OS is an interface b/n a user & computer h/w. An OS is a s/w which performs tasks like file management, memory mgmt, process mgmt, handling I/P & O/P & controlling peripheral devices such as disk drives & printers.



System software: is general purpose s/w which is used to operate computer h/w. It provides platform to run application s/w.

Application s/w: Special purpose s/w used by user for performing specific task.



OPERATING SYSTEM OBJECTIVES AND FUNCTIONS.

Objectives of OS:

1. Convenience
2. Efficiency
3. Ability to evolve.

Functions of OS:

1. Memory management.

Main memory is a large array of words or bytes where each word or byte has its own address. Main mry provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must be loaded to the main mry.

- Keeps track of primary memory ie, which part of mry is in use by whom, and which part is not in use.
- In multiprogramming, the OS decides which process will get mry when and how much.
- Allocates the mry when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

2. Process management.

- Keeps track of processor and status of processor. The pgm responsible for this task is known as traffic controller.
- Allocates the processor to a process.
- Deallocates processor when a process is no longer required.

3. Device management.

An OS manages device communication via their respective devices.

- Keeps tracks of all devices. Program responsible for this task is I/O controller.
- Decides which process gets the device when and for how much time
- Allocates the device in the efficient way.
- De-allocates devices.

4. File management.

- Keeps track of information, location, user, status etc. The collective facilities are known as file system.
- Decides who gets the resources
- Allocates & De-allocates resources.

Other activities

- security
- Control over system performance
- Job accounting.
- Error detecting aids
- Coordination b/n other S/w & users.

EVOLUTION OF OPERATING SYSTEM.

There are two basic types of OS. single user and multiusers. A multiuser OS handles multiple users as well as multiple peripheral devices simultaneously.

Serial Processing:

In 1950's programmer directly interact with h/w.

To execute a pgm

① Type the pgm on punch card
② Connect the punch card to card reader
③ Submit to computing machine. If any error is detected, it will be indicated by a light.
④ The programmer examines the register and main mry to identify the cause of the error.
⑤ Take the o/p on the printer
⑥ The system is ready to execute next program.

Drawback: It consumes more time, next pgm should wait for the completion of previous task. The pgms are executed one after other so is called as serial processing.

CPU is idle, when the currently running process is waiting for some I/O.

Mainframe System:

This type of computer system is used for commercial and scientific application. The main frame system are large in size, has huge storage capacity, good processing power and has high level of reliability. This may process its workload serially or concurrently. The resources are dedicated to a single program until its completion or they may be dynamically reassigned among a collection of active programs in different stages of execution.

Batch Systems

The user of a batch OS do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

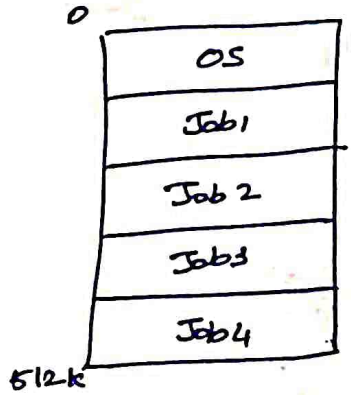
Disadvantages:

- Lack of interaction b/n the user & the job
- CPU is often idle, because the speed of the mechanical I/O device is slower than the CPU.
- Difficult to provide the desired priority.

Multiprogrammed Systems.

When two or more programs are in mry at a same time that shares the processor is referred to the multiprogramming OS. This assumes a single processor that is being shared. It increases CPU utilization by organizing jobs so that the CPU always has one job to execute.

Job entering the system is kept into the mry. OS picks the job & begins to execute one of the job from the mry. If the running process waits for IO, the CPU switches from the job to another job on the job pool.



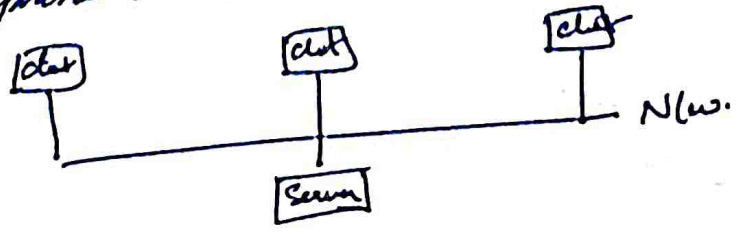
Mry of multiprogramming system.

Advantages.
Can get efficient my utilization
CPU is never idle, so increase in performance.
waiting time is limited.

Time Sharing:
Supports interactive users. This is also called as multitasking. Here each user is given a time slice for executing his job. This can run several programs at the same time. The difference b/n multiprogrammed system & time sharing system is that multiprogrammed sys maximizes processor use and time sharing system minimize response time.

Desktop System
Desktop O.S were neither multiuser nor multitasking. The goal of desktop is maximizing user convenience and responsiveness.

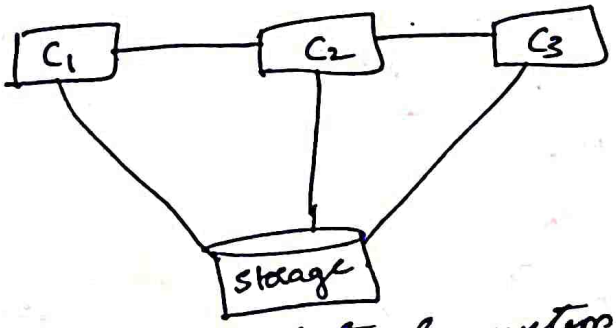
Distributed System:
A distributed system looks to its user like an ordinary OS but it runs on multiple independent CPU. Distributed systems depends on networking for their functionality These share computational tasks & provide a rich set of features to users.
Loosely coupled system - each processor has its own local memory. Each processor communicate with one another through various communication lines like high-speed buses or telephone lines.



Advantages:
- Resource sharing
- Higher reliability
- Better price performance ratio

Clustered System

Clustered systems are created by two or more individual computer system merged together. They have independent computer system with a common storage & the systems work together.



Each node in the clustered system contains the cluster software. This s/w monitors the cluster system and makes sure it is working as required. If any one node in the clustered system fails, then the rest of the nodes take control of its storage & resources and try to restart.

There are two types of clusters.

Asymmetric cluster:

One of the node in the cluster is on hot stand by mode and all the others run the required application. The hot standby mode continuously monitors the server & if it fails, the hot standby node takes its place.

Symmetric cluster

Two or more nodes run application as well as monitors each other.

Real Time System.

The response time is very less as compared to online processing. Real time systems are used when there are rigid time requirements on the operation of a processor or the flow of data & real time systems can be used as a control device in a dedicated application. This should have a well defined fixed time constraints, otherwise the system will fail. eg) Scientific experiments, medical imaging systems etc

Two types.

Hard real-time system:
Critical tasks complete on time. Here secondary storage is limited and so the data is stored on ROM.

Soft real-time system:
A critical real-time task gets priority over other tasks & retain the priority until it completes.
eg) multimedia, virtual reality etc.

Handheld Systems.

Handheld systems include Personal Digital Assistants such as palm pilots, cellular phones with connectivity to a network such as internet.

COMPUTER SYSTEM ORGANIZATION

OPERATING SYSTEM STRUCTURE

Storage Structure:

Computer programs must be in main mry to be executed. Main mry is the only large storage area that the processor can access directly.

A typical instruction execution cycle has fetch & execute cycle.

* Main mry is too small to store all needed pgms & data permanently.
* Main mry is volatile storage.

Computer systems provide secondary storage as an extension of main mry to hold large quantities of data permanently.

Magnetic disk a storage device provides storage for both program & data.

Main memory:

Main mry is physically organized as a large number of cells that are capable of storing one bit each. Logically they are organized as groups of bits called words that are assigned an address.

Memory mapped I/O provide convenient access to I/O devices. Ranges of mry addresses are set aside, and are mapped to the device registers.

Magnetic Disks:

Magnetic disk provide the bulk of secondary storage for modern computer systems.

Each disk platter has a flat circular shape, like a CD. Two surfaces of a platter are covered with a magnetic material. Information are magnetically recorded on the platters.

A read-write head flies just above each surface of every platter. The heads are attached to a disk arm, which moves all the heads as a unit. The platter surface is logically divided into circular tracks, which are sub-divided into sectors. The set of tracks that are at one arm position forms a cylinder. The storage capacity of common disk drives is measured in gigabytes.

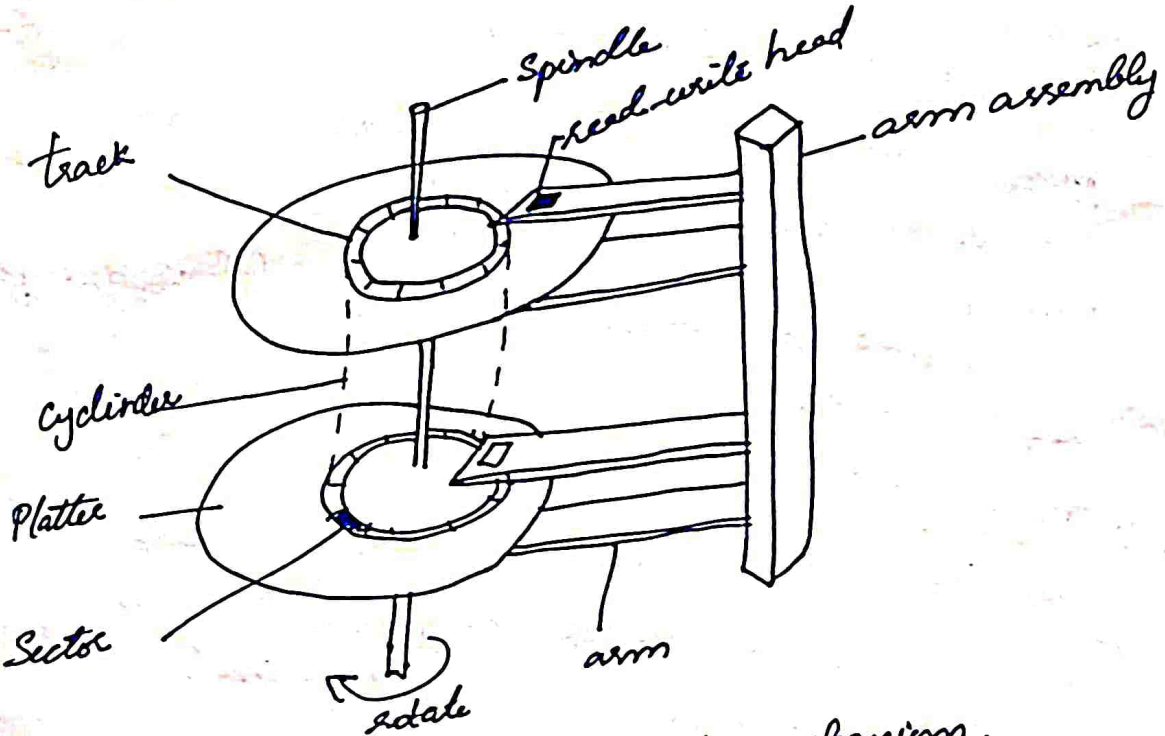


fig: Moving head disk mechanism.

When the disk is in use, a drive motor spins it at high speed.

Transfer rate :- The rate at which data flow b/n the drive & the computer.

Positioning time / Random access time: Time to move the disk arm to the desired cylinder, called the seek time and time for the desired sector to rotate to the disk head, called rotational latency.

Magnetic Tapes:

Magnetic tapes was used as an early secondary storage medium. It is permanent & can hold large quantities of data. Its access time is slow when compared to main memory. Random access to magnetic ~~disk~~ tape is much slower than random access to magnetic disk, so tapes are not much useful for secondary storage. Tapes are mainly used for backup. for infrequently used information.

OPERATING SYSTEM STRUCTURE

System Component.

Process mgmt
- Creating & deleting both user & system processes
- Suspending & resuming process
- Providing mechanism for process synchronization, deadlock handling.

Main memory mgmt:
- Keep track of mey
- Decide which processes are loaded to mey
- Allocating & deallocating mey space as needed.

File mgmt:
- creating & deleting file & directories
- manipulation of files & directories
- Mapping files onto secondary storage.
- Backing up files on stable storage media.

I/O System mgmt.

- A general device-driver interface
- Drivers for specific hardware devices
- A mem mgmt component that includes buffering, caching & spooling.

Secondary storage mgmt.

- Free space mgmt
- Storage allocation
- Disk scheduling.

Operating System Services:

- Program execution:
  The system must load a pgm into memory to execute. That pgm must end its execution either normally or abnormally.
- I/O operations:
  A running pgm may require I/O. This I/O may involve a file or an I/O device.
- File-system manipulation:
  Programs need to read & write files. Also they need to create and delete files by name.
- Communications:
  Communication takes place b/n two processes executing on same computer or on different computers. Communication may be implemented by shared mem or by message passing techniques.

Error detection

Error may occur in CPU, memory h/w, I/O devices, or in user pgm. The OS should take appropriate action to ensure correct & consistent computing.

Resource allocation:

When multiple users are logged on the s/m or multiple jobs are running at the same time, resources must be allocated to each of them.

Accounting:

Keeps track of which users use how many & which kind of computer resources.

Protection:

Protection ensures that all access to system resources is controlled.

SYSTEM CALLS

System calls provide the interface b/n a process and the operating system. These calls are available as assembly language instruction.

A computer program makes a system call when it makes a request to the OS's kernel. System call provides the services of the OS to the user programs via Application Program interface (API). It provides an interface b/n a process & OS to allow user-level processes to request services of the OS. System calls are the only entry points into the kernel system.

Three general methods are used to pass parameters between a running program and the OS.

- Pass parameters in registers.
- Store the parameters in a table in mem and the table address is passed as a parameter in a register.
- Push the parameters onto the stack by the program, & pop off the stack by OS.

System calls can be classified into 5 types.

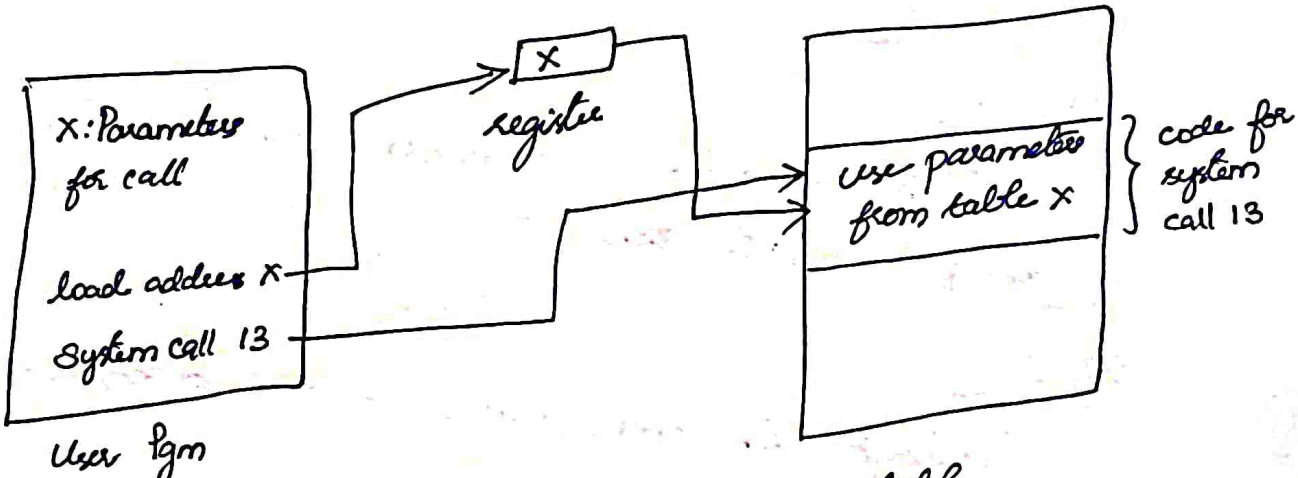


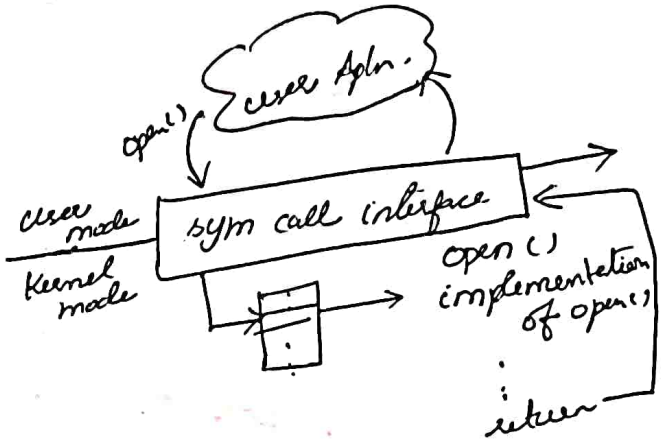
fig: Passing of parameters as a table

Process Control.

A running pgm either halts its execution normally or abnormally. If a sys call is made to terminate the currently running pgm abnormally, a dump of mem is taken and an error msg is generated. The dump is written to disk and may be examined by a debugger to determine the cause of the problem.

Under either normal or abnormal circumstance, the OS must transfer control to the command interpreter. The cmd interpreter then reads the next command.

Process Control:
- end, abort
- load, execute
- create process, terminate process
- get process attribute, set process attributes
- wait for time
- wait event, signal event
- allocate & free memory.



File management:
- create file, delete file
- open, close
- read, write, reposition
- get file attribute, set file attribute

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

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

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

Communication can be done through either message passing or by shared memory.

Message passing :- Information is passed through interprocess communication. If process of same computer then a connection is established by ~~making~~ sharing the process name of each process. If process of different compiles host name like IP address is transfered b/n them to make a connection before communication.

The system calls used here are gethostid, get processid, open, close, open connection, close connection, accept connection. The source of communication is known a client & the other is called as server. Then msgs are passed b/n them.

Shared mry model.

Processes use map memory system calls to gain access to regions of mry owned by other processes. Processes can exchange information by reading & writing data in these shared areas. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

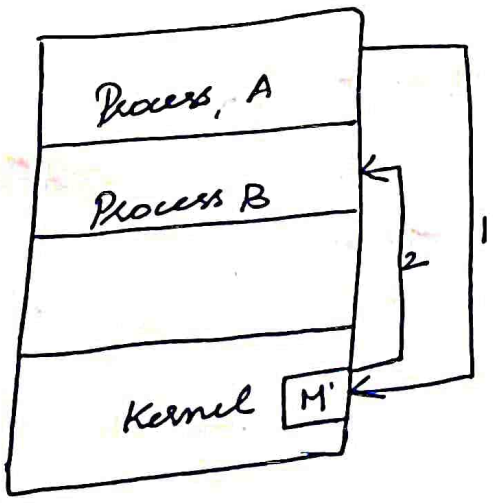
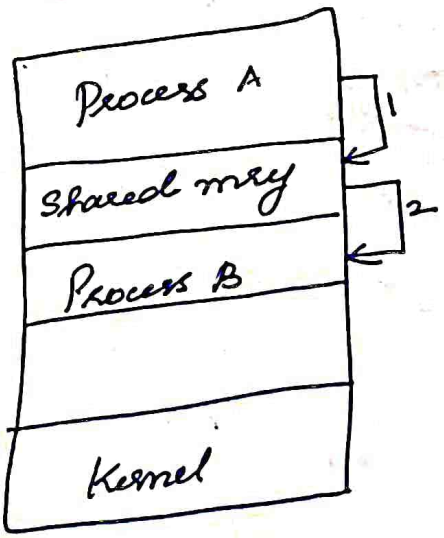


fig:- Msg passing.

- smaller amount of data



Shared mry.

- Large data are shared.

System Programs

File management:
These programs create, delete, copy, rename, print, dump, list & generally manipulate files and directories.

Status information:
These pgms ask system for date, time, amount of available mey or disk space, no. of users etc.

File modification:
Text editors may be available to create & modify the content of files.

Programming language support:
Compilers, assemblers, & interpreters for common programming languages are provided to the users with OS.

Program loading & execution:
Provides absolute loaders, relocatable loaders, linkage editors and overlay loaders.

Communications:
Provide mechanism for creating virtual connections among processes, users & different computer systems. They allow users to send msgs to one another's screens to browse web pages, send e-mail, to login remotely, or to transfer files from one machine to another.

OSystem structure:

A system as large & complex as a modern OS must be engineered carefully. A common approach is to partition the task into small components or modules, rather than have one monolithic system. Each modules should be a well defined portion of the system with carefully defined I/p, O/p & functions.

①Simple structure

Many OS dont have well defined structures. Frequently, such systems started as small, simple & limited s/m.

eg) MS-DOS

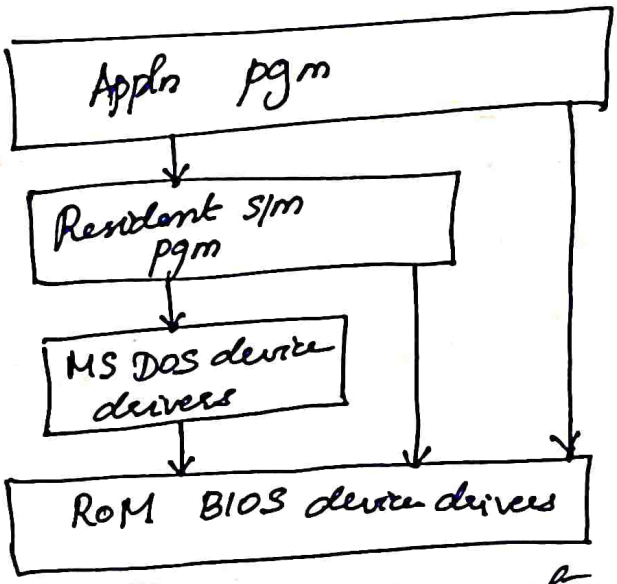


fig:- MS-DOS Layer structure.

In this structure the interface & levels of functionality are not well separated. For instance application pgms are able to access the basic I/O routines to write directly to the display & disk drives.

So MS-DOS are vulnerable to errant progeams causing the entire system to crash when user pgms fail.

Another example is UNIX OS. Like MS-DOS, UNIX initially was limited by h/w functionality. It consists of two parts
① Kernel ② System pgm.

The Kernel is further separated into a series of interfaces and device drives, which have been added and expanded over the years as UNIX has evolved.

② Layered approach

With proper h/w support, OS can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS & UNIX sys. The OS can then retain much greater control over the computer & over the application.

In layered approach the OS is broken into no. of layers. The bottom layer is the h/w (level 0) & the highest (layer N) is the user interface.

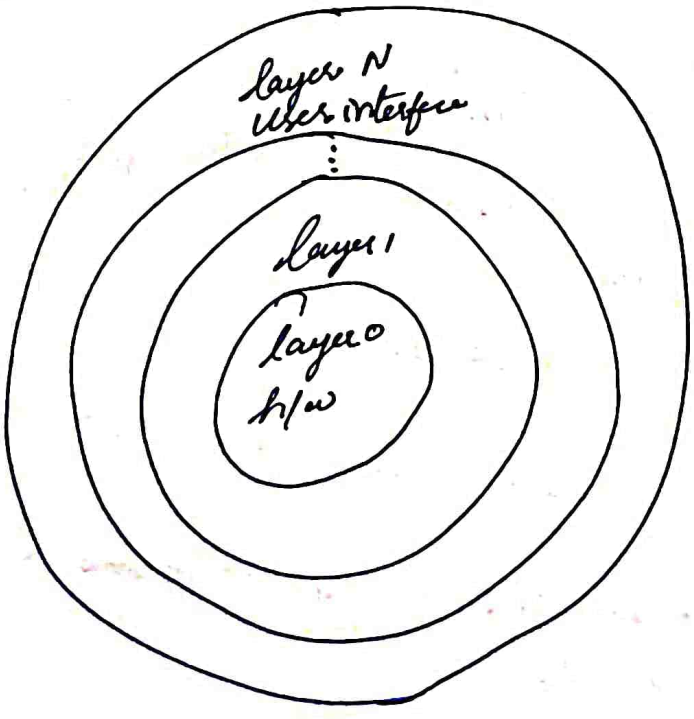


fig: A layered OS.

A OS layer consists of data structures & set of routines that can be invoked by other layers.

The advantage of layered approach is simplicity of construction and debugging. One by one the layeres can be debugged.

③ Microkernels:

As UNIX expanded the OS became large & difficult to manage. So microkernels was introduced. This approach structured the OS by removing all non essential components from the kernel and implementing them as system & user-level programs. The result is a smaller kernel.

The main fn of the microkernel is to provide communication b/n the client pgm & the pgm or services running in user space. Communication is by msg passing mechanism.

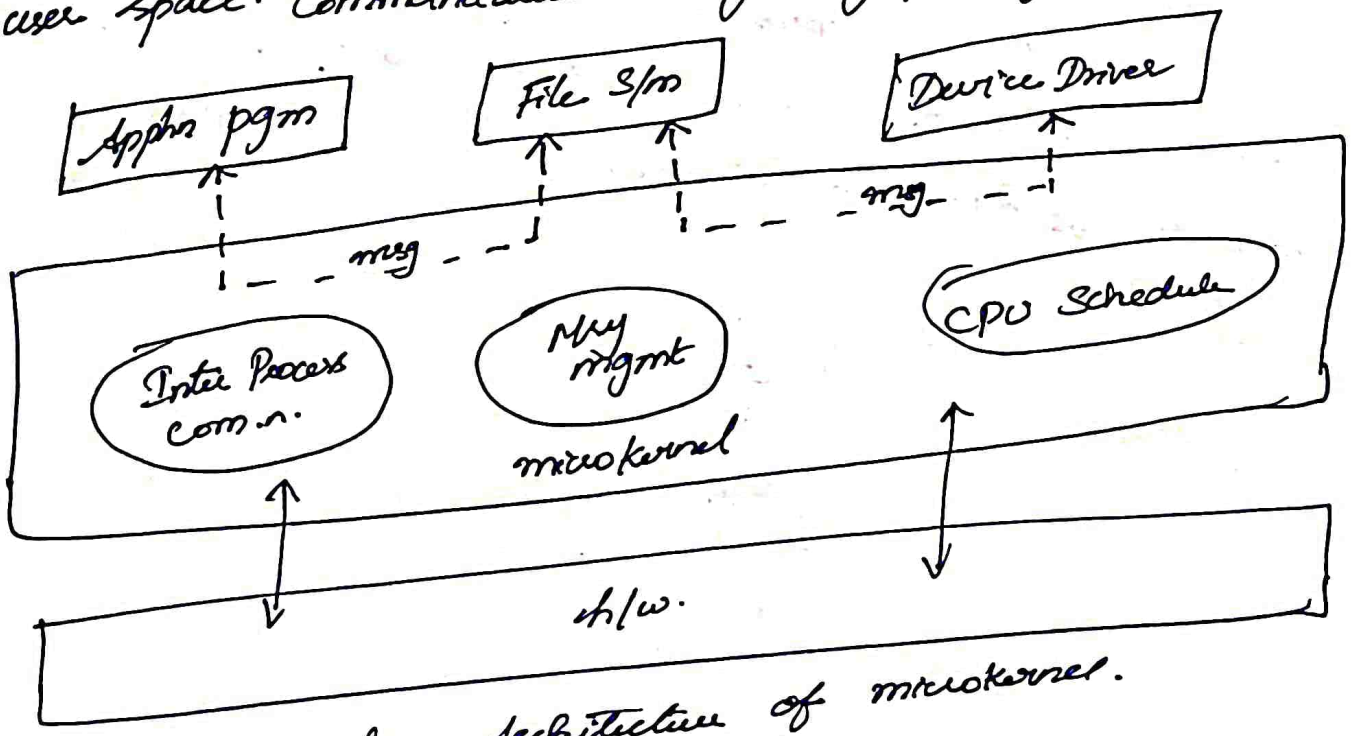


fig:- Architecture of microkernel.

If the client program wishes to access a file, it must interact with the file server. The client pgm & the service never interact directly.

Benefit of microkernel approach is that it makes extending the OS easier. All new services are added to user space & consequently don't require modification of the kernel. When the kernel does have to be modified the changes tends to fewer because the microkernel is a smaller kernel. The resulting OS is easier to port from one h/w design to another. This also provide more security and reliability.

④ Modules:

The current methodology for OS design involves using loadable kernel modules. The idea of the design is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running.

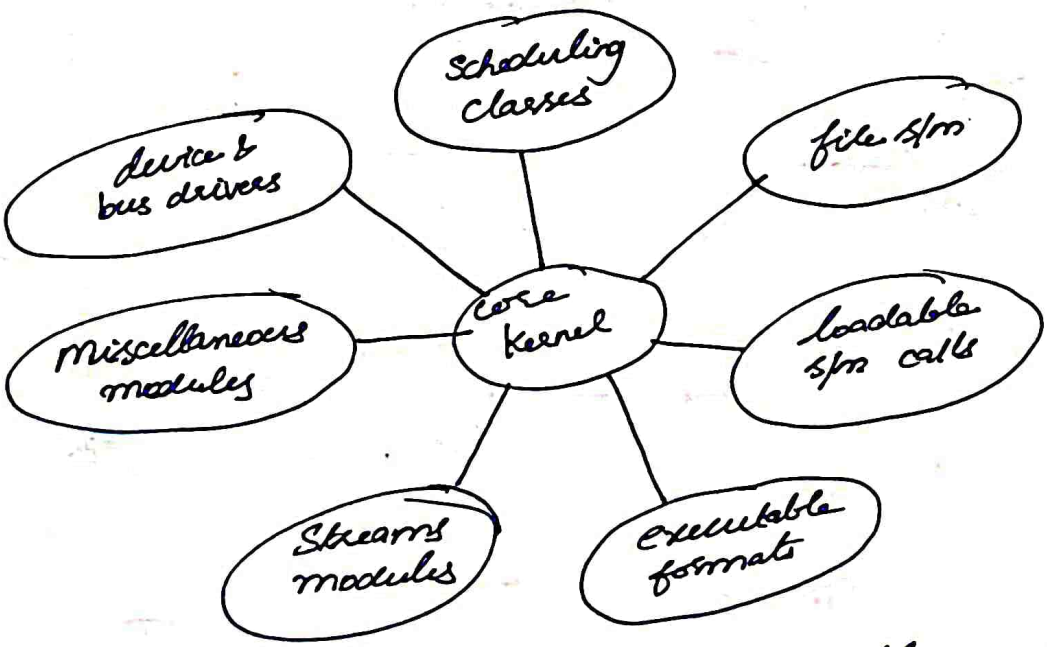


fig:- Solaris loadable modules.

Here any module can call any other module

linking services dynamically is preferable to add new features directly to the kernel, which require recompiling the kernel every time a when a change was made

⑤ Hybrid Systems:

Different structures (OS) are combined to form a hybrid system that address performance, security, usability issues.

① Mac OS X
② iOS
③ Android

Mac OS X

The Apple Mac OS X OS uses a hybrid structure.

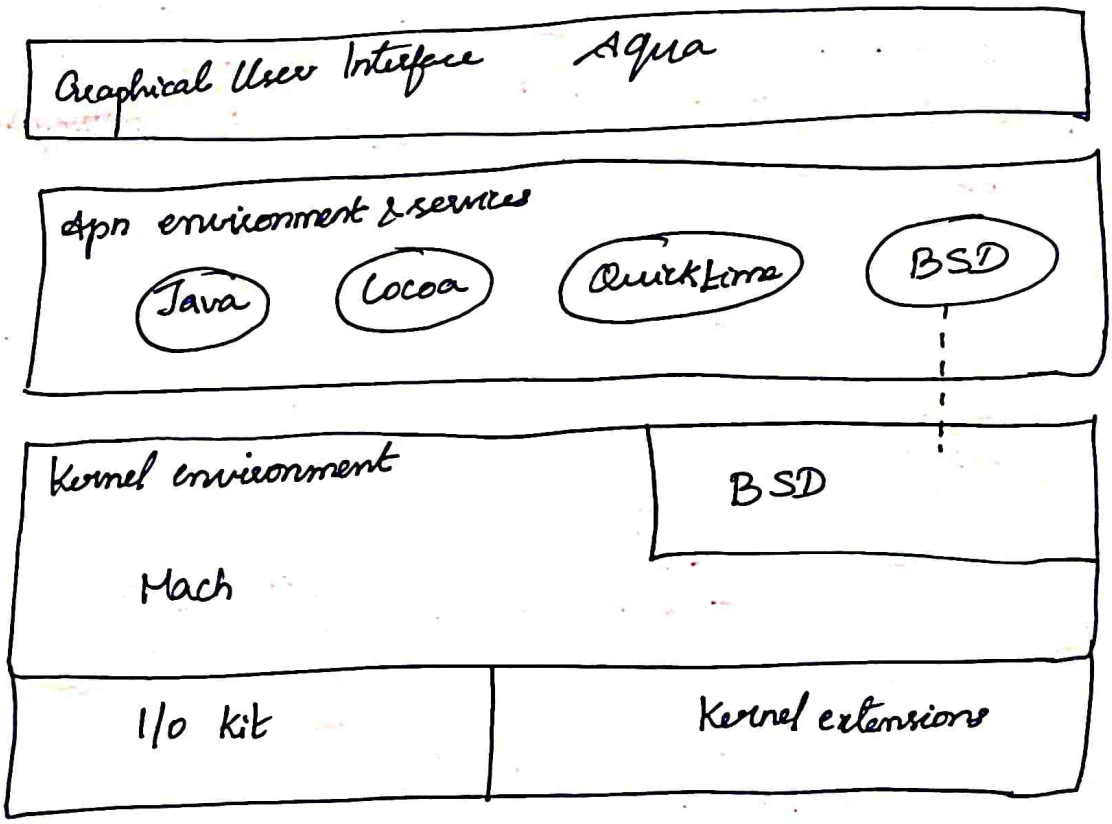


fig:- Mac OS structure.

Top layers include Aqua user interface and set of appln environment & services.

Cocoa – API for objective C programming used for writing Mac OS X appln.

Kernel environment contains Mach microkernel & BSD UNIX kernel.

Mach provides
- mem mgt
- Remote procedure calls
- Inter process communication
- msg passing &
- Thread scheduling.

BSD provides
- Command line interface
- networking
- file system.

iOS:

iOS is mobile OS for iPhone & iPad. iOS is structured on the Mac OS X OS, with added functionality pertinent to mobile devices, but does not directly run Mac OS X applications.

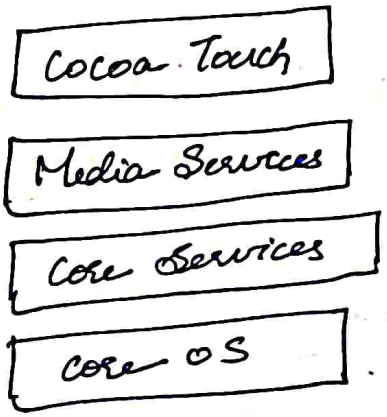


fig:- Apple iOS

Cocoa Touch - provides support for h/w features unique to mobile devices.
media service - provide service for graphics, audio, video
Core services - support for cloud computing & databases.

Android

Bottom level is the LINUX kernel
Runtime environments include core set of libraries and Dalvik virtual machine.
Android is developed in JAVA.

Dalvik virtual machine runs the java bytecode. This VM for Android was designed to optimize mobile device with limited mry & CPU processing capabilities.

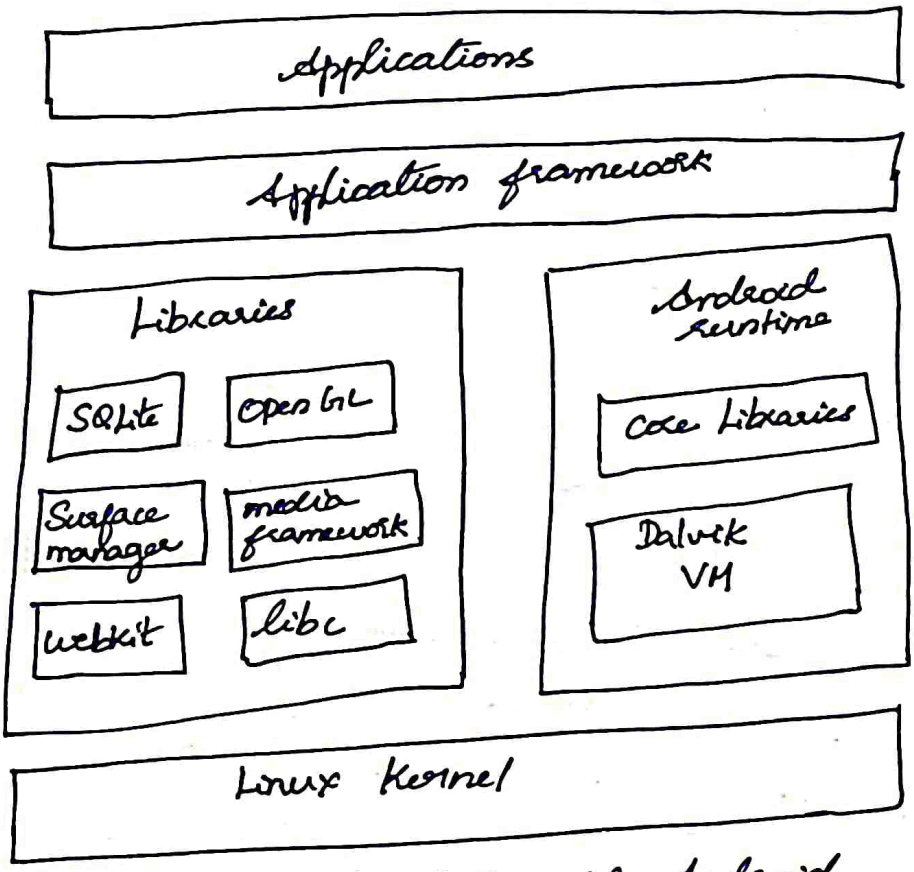


fig:- Architecture of Android.

OPERATING SYSTEM GENERATION & SYSTEM BOOT

Operating System generation:

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

The following kind of information must be determined to generate a system.

- what CPU is to be used?
- what options are installed?
- How will the boot disk be formatted?
- How many sections or partitions will be separated & what will go into each partition?
- How much mem is available?
- what devices are available?
- what OS options are desired & what parameters values are to be used?

Once this info is determined, a system administrator can use it to modify a copy of the source code of the OS. Then the OS is compiled. Data declarations, initializations, & constants along with conditional compilation, produce an o/p object version of the OS that is tailored to the system described.

System boot:

After an OS is generated, it must be made available for use by the h/w. The procedure of starting a computer by loading the kernel is known as booting. A program known as bootstrap program or bootstrap loader locates the kernel, loads it into main mem, and starts its execution.

When a CPU is reset or rebooted, the instruction register is loaded with a predefined mem location and execution starts there. The bootstrap pgm is in the form of (ROM) Read Only Memory.

One task of bootstrap program is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps. Then it initialize all aspects of the system, from CPU registers to device controllers & the contents of main mem.

The OS is stored in ROM for small OS. The problem here is that changing the bootstrap code requires changing the ROM H/w chips. This problem is resolved using erasable programmable read-only mem (EPROM). EPROM read & write is possible.

All form of ROM are known as firmware (permanent s/w programmed into a ROM). Problem with firmware is, executing code there is slower than executing code on RAM. Firmware is expensive

For large OS or for systems that change frequently, the bootstrap loader is stored in firmware, & the OS is on the disk.

OPERATING SYSTEM OPERATION

OS are interrupt driven. If there are no process to execute, no I/O devices to service, no user to whom to respond, a OS will sit quietly wait for something to happen. The interrupt nature of the OS defines that systems structure.

Dual mode & multimode operation:

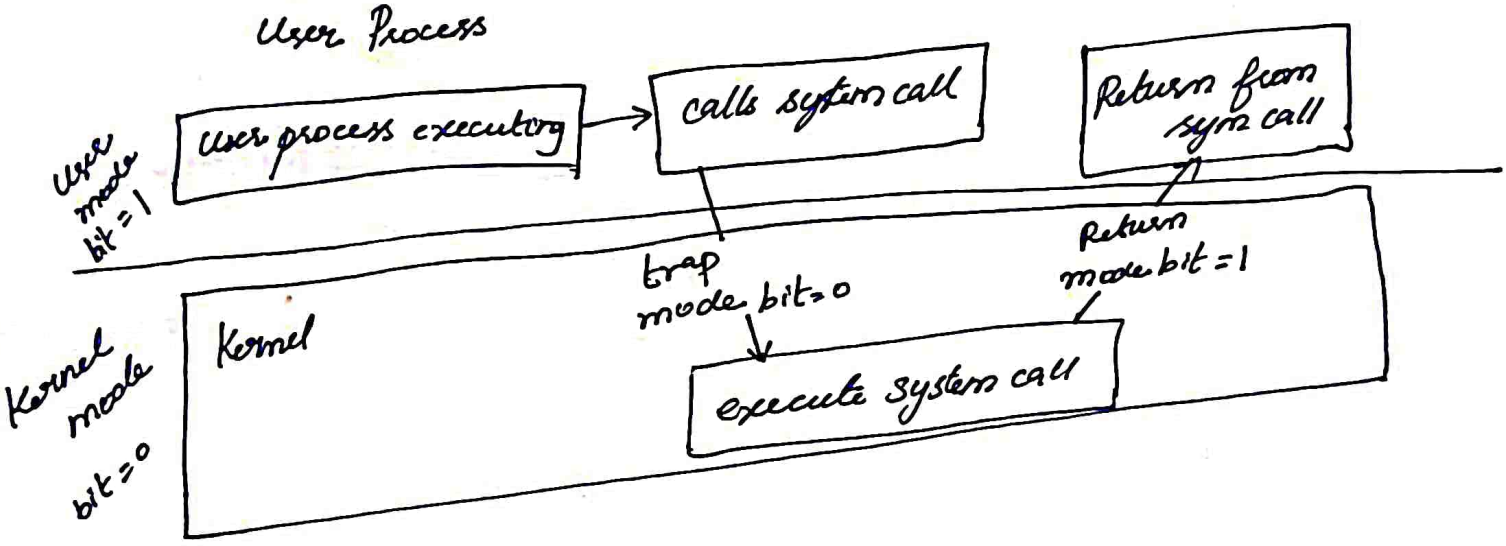
There are 2 modes of operation user mode & kernel mode. A bit called the mode bit in the H/w indicate the current mode. Kernel mode = 0

User mode = 1.

At s/m boot time, the h/w starts in Kernel mode. The OS is then loaded and starts user applns. in user mode. During a trap or interrupt the h/w switches to Kernel mode.

The dual mode of operation ensures security. The h/w allows privileged instruction to be executed only in Kernel mode.

CPU support virtualization have separate mode to indicate when the virtual machine manager executes.



Timer:

Timer ensure the OS maintains control over the CPU. A user pgm cannot stuck in an infinite loop or to fail to call sys services & never return to the OS. To accomplish this timer was introduced. A timer can be set to interrupt the computer after a specified period. The period may be fixed or variable.

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 turning over control to the user, the OS ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the OS.

We can prevent a user pgm running too long using a timer.

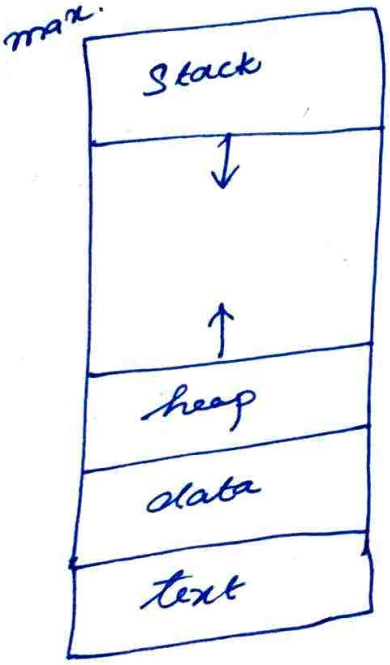
UNIT- II

PROCESS MANAGEMENT

PROCESS CONCEPT

Process: A process is a pgm in execution. Process also includes the current activity, as represented by the value of the pgm counter & the content of the processor's registers.

A process includes process stack, data section, & a heap.



stack - fn parameters, return addresses, local variables.

data - global variables

heap - mem dynamically allocated during run time.

fig:- Process in memory.

A process is an active entity with a pgm counter and associated resources. A program becomes a process when it is loaded into mem.

Process state:

As a process executes, it changes state. The different states are,

New :- The process is being created

Running :- Instructions are being executed

Waiting : The process is waiting for some event to occur.

Ready : The process is waiting to be assigned to a processor.

Terminated : The process has finished execution.

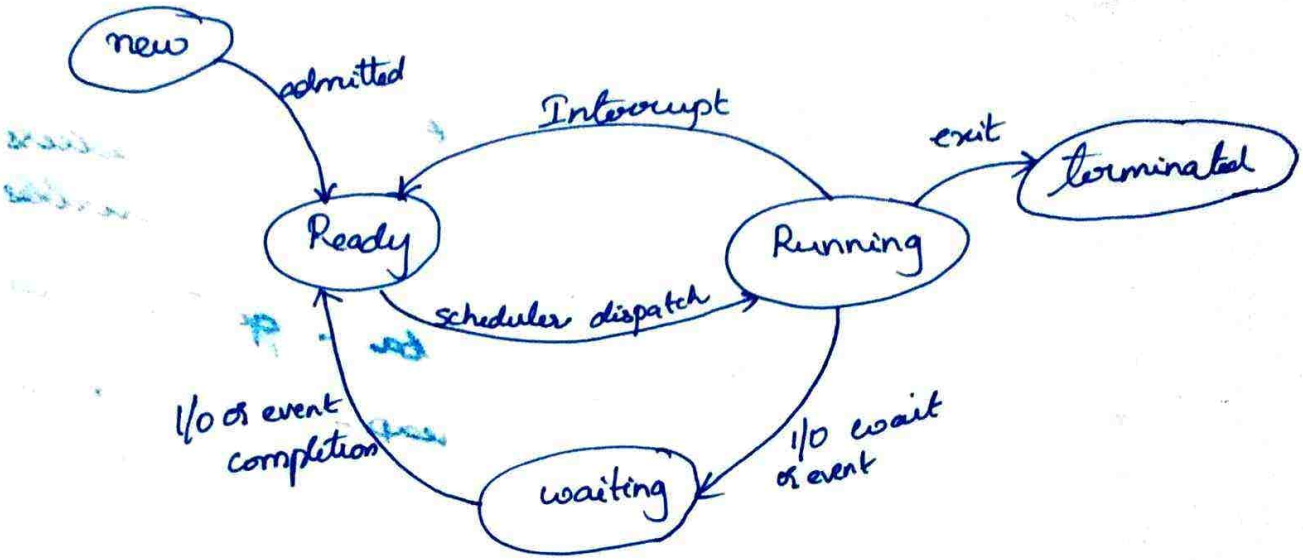


fig :- State process.

Process Control Block.

Each process is represented in the OS by a process control block (PCB) or task control block.

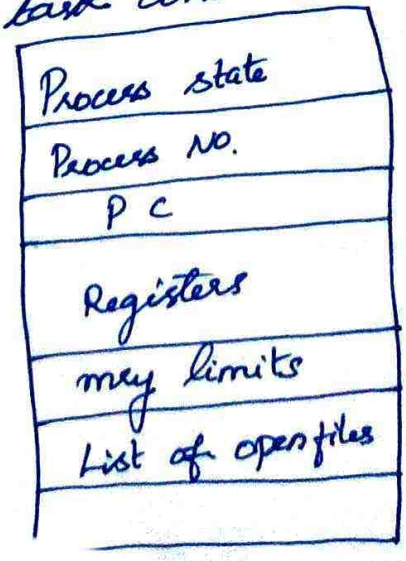


fig :- PCB

Process State:
The state may be new, ready, running, waiting, halted.

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

CPU registers:
The registers vary in no. & type depending on the architecture. They include accumulator, index reg., stack pointer, general purpose reg., any conditional code information. State information must be saved in case of interrupt.

CPU scheduling information:
This includes process priority, pointers to scheduling queues, other scheduling parameters.

Mry mgmt info.:
Value of the base and limit registers, page table or the segment table.

Accounting information:
Includes amount of CPU & real time used, time limits, account numbers, job or process no.

I/O status information:
List of I/O devices allocated to the process, list of open files etc.

Threads:

A process is a pgm that performs a single thread of execution. Modern OS have multiple threads of execution to perform more than one task at a time. In multicore system, multiple threads can run in parallel. A PC is expanded to include information for each thread.

Process scheduling:

The aim of multiprogramming is to have some process running at all times to maximize CPU utilization.

The aim of time sharing is to switch the CPU among processes so that users can interact with each pgm while it is running. To meet these objectives, the process scheduler selects an available process for pgm execution on the CPU.

Scheduling Queues:

Job Queue :- As processes enter the sym. they are put into Job queue.

Ready Queue :- The process that are ready & waiting to execute are kept on ready queue. This queue is stored as a linked list.

Device queue :- The process waiting for a I/O device is put to a device queue. Each device has its own queue.

A new process is initially put in the ready queue. It waits there until it is selected for execution or dispatched. Once the process is allocated the CPU & is executing, one of the event could occur.

- The process could issue an I/O request & be put onto an I/O queue.
- The process could create a new child & wait for the child's termination.
- The process could be removed from the CPU as a result of interrupt and be put back into the ready queue.

In first 2 cases, the process switches from waiting state to ready state & then it is put to the ready queue.

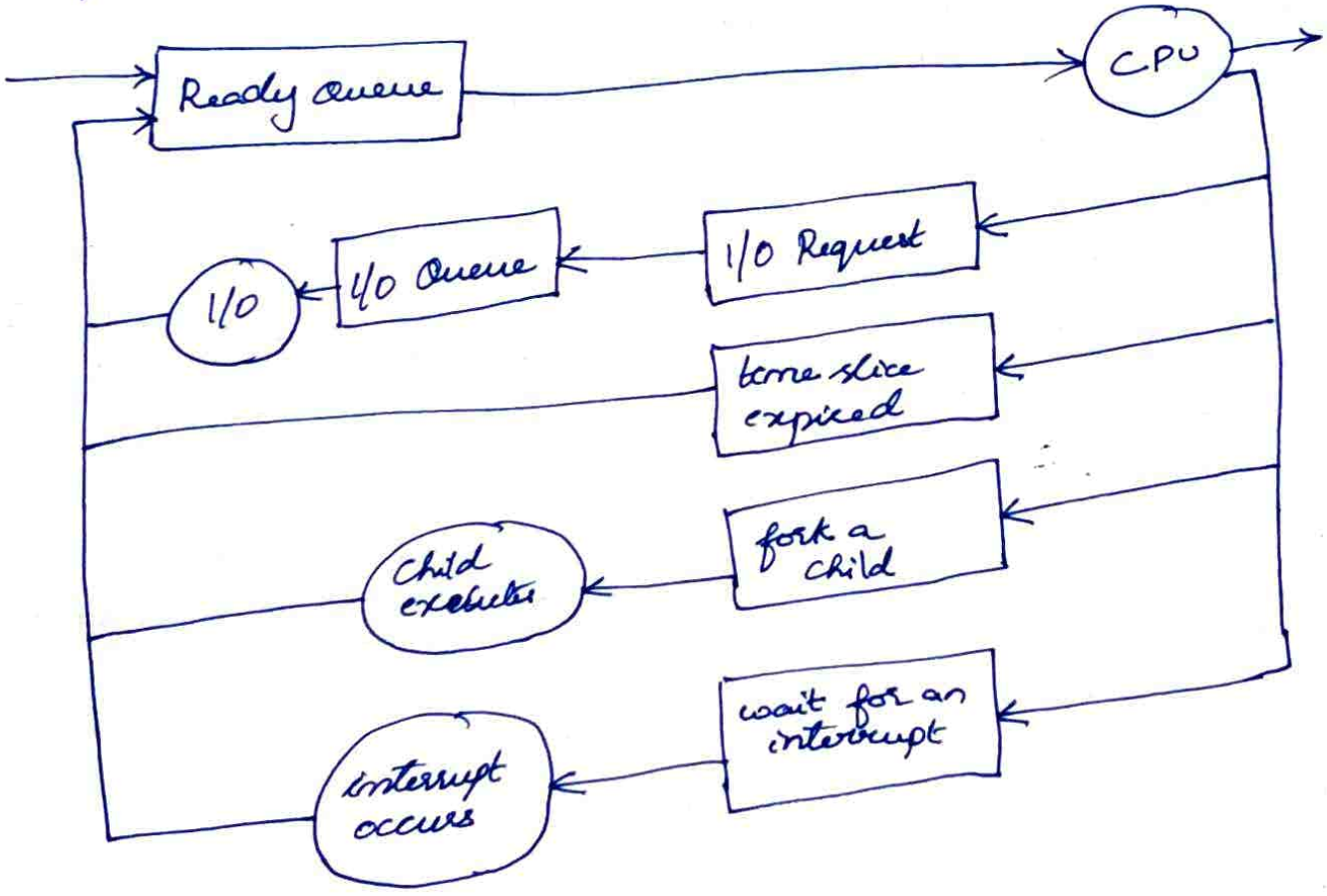


fig:- Queueing diagram representation of process scheduling.

Schedulers:

A process migrates among the various scheduling queues throughout its lifetime. The OS must select the process from different queues in some fashion. The selection process is done by the scheduler.

Three types:

1) Long term scheduler
2) Short term scheduler
3) Medium term scheduler.

Long term scheduler:

- Also called as job scheduler.
- Selects process and loads them into mry for execution.

Short term scheduler

- Also called as CPU scheduler.
- selects from among the processes that are ready to execute and allocates the CPU to one of them.

Medium term scheduler

- It removes a process from mry and reduces the degree of multiprogramming. Later the process can be reintroduced into mry and its execution can be continued. This scheme is called swapping.

I/O bound process:
Is one that spends more of its time doing I/O than it spends doing computations.

CPU bound process:
Generates I/O requests infrequently and use more time for computations.

A long term scheduler should select a good process mix of I/O bound & CPU bound processes.

If all the process are I/O bound, the ready queue will always be empty, and the short term scheduler will have little to do.

If all processes are CPU bound, I/O queue will always be empty, devices will go unused. The s/m will be unbalanced. To produce a best performance s/m both CPU bound & I/O bound processes should be scheduled evenly.

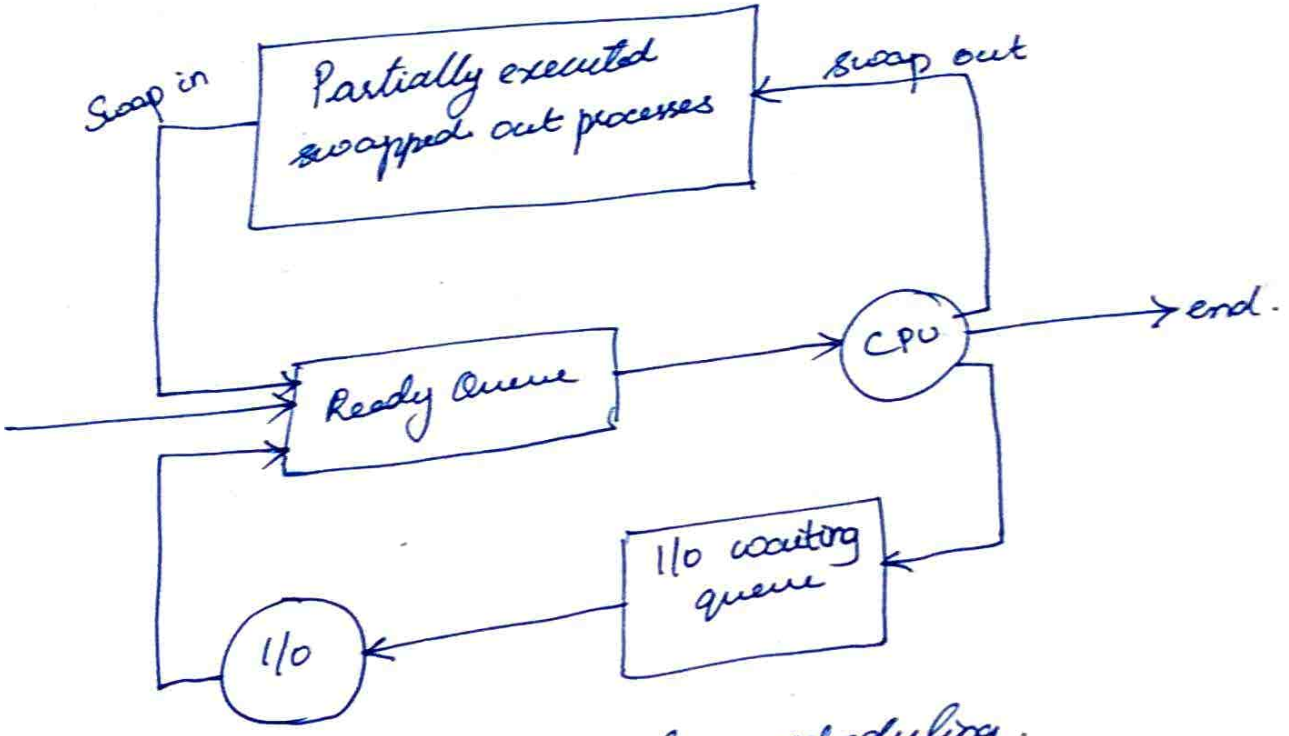


fig:- medium term scheduling.

Context switch:

When an interrupt occurs, the s/m save the current context of the process running on the CPU so that it can restore that context when its processing is done. The context includes the value of CPU registers, process state, and mry mgmt information.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as context switch.

OPERATIONS ON PROCESSES

The processes can execute concurrently and they can be created and deleted dynamically.

Process Creation

A process may create several new process. The new process is the child and the other is the parent. Each of child process can inturn create other process forming a tree of processes.

Each process has its own unique process identifier (Pid), an integer no.,

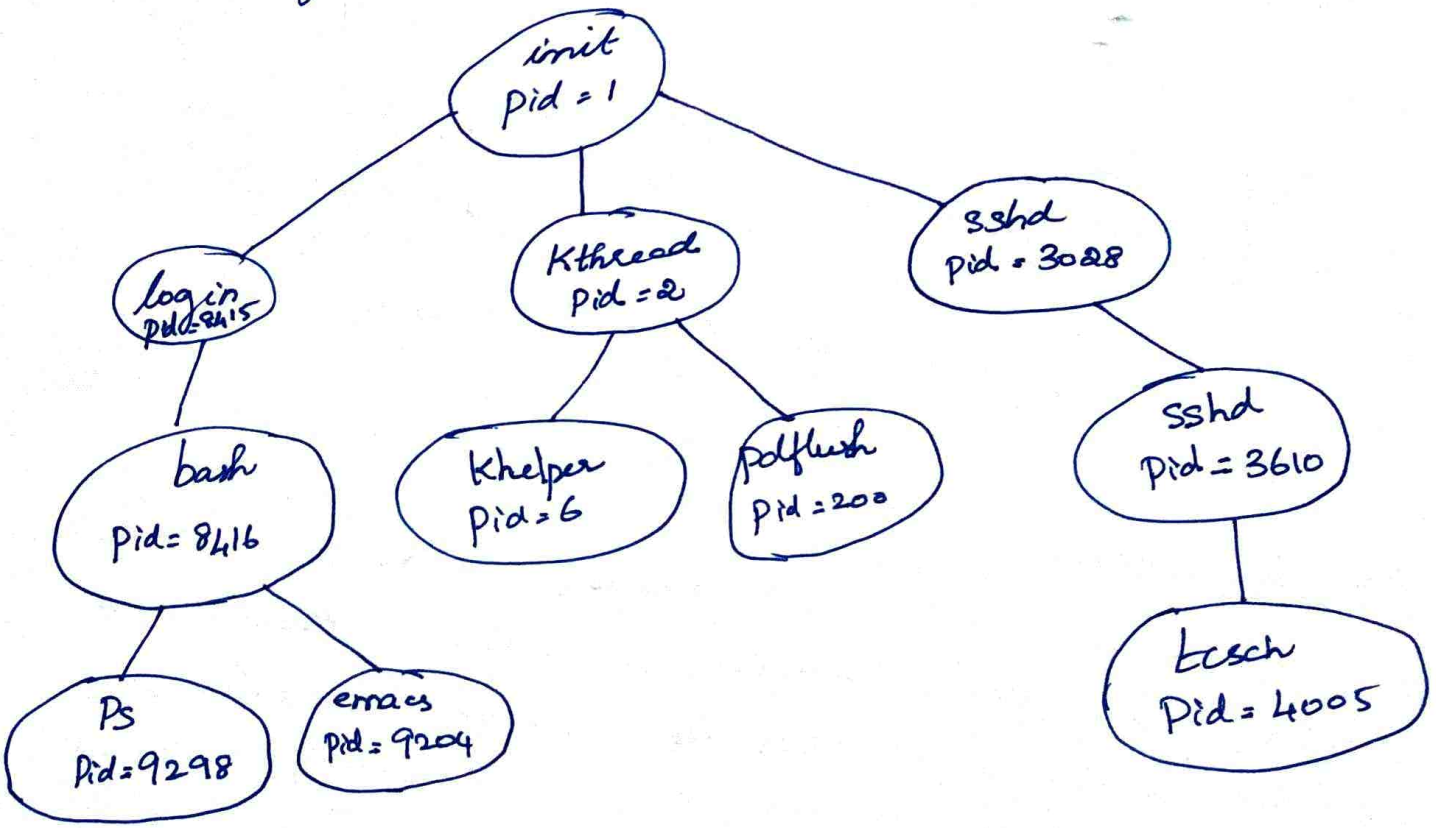


fig:- Tree of processes on LINUX s/m.

The init process serves as the root parent process for all user processes. Init process can create various user processes.
- Kthread process is responsible for creating additional processes that performs tasks on behalf of the kernel.
- Sshd process manages clients. (Secure shell) to the s/m by ssh.
- Login - manages clients that directly log onto the system.

When a process creates a child process, the child process will need of some resources. Those resources will be given either by the OS or by its parent process.
When a process creates a new process, 2 possibilities of execution exist.

① parent & child concurrently execute
② Parent waits for some child to terminate.

Two address space possibilities are.

① Child process is a duplicate of the parent process
② Child has new pgm loaded into it.

Sym calls - fork(), CreateProcess(), exec(), exit(), wait()

Process termination

A process terminates when it finishes executing its final statement & asks the OS to delete it by using the exit() system call. All the resources of the process including physical & virtual mem, open files. I/O buffers are deallocated by the OS.

A Parent process can terminate another child process by TerminateProcess() sym call. Reasons for termination may be,

- Child exceeds the resource usage
- Task of child is not required
- If parent exit child has to exit.

If a process terminates, then all its children must also be terminated. This phenomenon is called as cascading termination.

exit() sys call is called to exit the process execution directly.

wait() - A parent waits for a child to terminate

A process that has terminated, but whose parent has not yet called wait() is known as a zombie process.

If a parent did not invoke wait() and instead terminated, thereby leaving its child process as orphans, the OS itself will assign the init process as the new parent to the orphan processes.

INTER PROCESS COMMUNICATION

Processes executing concurrently in the OS may be independent or cooperating processes.

Independent Process :- Cannot affect or be affected by other processes. i.e, it does not share data with others.

Cooperating processes :- can affect or be affected by other processes. It share data -

Reasons for providing process cooperation are,

- Information sharing :
  Several users need the same data
- Computational speed up :
  The tasks are broken into pieces and executed parallely to speed up computation.
- Modularity :
  Construct a sys in a modular fashion, dividing the sys fn into separate process or threads.

• Convenience

Individual user may work on many tasks at the same time.

Cooperating processes require an interprocess communication mechanism to exchange data & information. (IPC)
2 models of IPC are

- Shared memory
- Message passing.

* Shared memory:

A shared my region resides in the address space of the process creating the shared my segment. Other process that wish to communicate using this shared my segment must attach it to their address space. Then they can exchange information by reading & writing data in the shared area. The data & location for sharing are determined by these processes. The processes ensure that they don't write to the same location simultaneously.

eg) Producer Consumer problem.

A producer process produces information that is consumed by a consumer process. eg) Server is the producer and client is the consumer. One solution to producer consumer problem is shared my. To allow concurrent working of producer & consumer, a buffer of items is made available where the producer fill data and consumer empties the data. A producer can produce one item while the consumer is consuming another item. A consumer should not try to consume an item that has not yet produced.

Two types of buffers can be used.

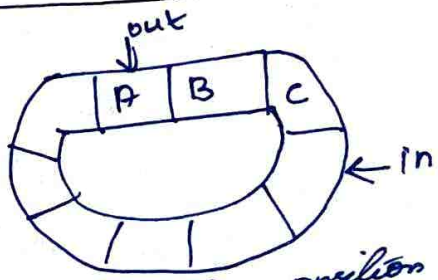
Unbounded buffer: There is not limit on the size of the buffer. The consumer may wait for new items, producer can always produce new item.

Bounded buffer: Has fixed size buffer. Consumer waits if buffer is empty and producer waits if the buffer is full.

The shared buffer is implemented as a circular array in & out are pointers.

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

Producer process using shared mem.



in → next free position in the buffer

out → first full position in the buffer.

in == out, buffer is ~~full~~ empty

((in+1) % BUFFER_SIZE) == out

buffer is full.

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

Consumer process.

* Message-Passing Systems

Message passing provides a mechanism to allow processes to communicate & to synchronize their actions without sharing the same address. It is useful in a distributed environment.

This provides 2 operations

* send (message)
* receive (message)

Message size are either fixed or variable. Variable size is more preferable.

If process P & Q wants to send msg first a communication link should be established b/n them. Connection links can be

* direct or indirect communication
* Synchronous or asynchronous communication
* Automatic or explicit buffering.

Naming:

Processes to communicate should have a name to refer each other.

* Direct communication: Each process should name the recipient or sender of the communication.

send (P, msg) send msg to P

receive (Q, msg) Receive msg from Q.

Properties of communication link

- A link is established automatically b/n every pair of processes that want to communicate. The process need to know only each other's identity to communicate.
- A link is ~~established~~ associated with exactly 2 processes.
- B/n each pair of processes, there exist exactly one link.

Symmetry addressing:

Both sender & receiver must name the other to communicate.

Asymmetry addressing:

Only sender names the recipient & the recipient is not required to name the sender.

send (P, msg) send a msg to P

receive (id, msg) Receive a msg from any process.
id – name of the process.

* Indirect communication:

The msgs are sent to and received from mailbox or ports. Each mailbox has a unique identification. A process can communicate with another process via a no of different mailbox, but 2 processes can communicate only if they have a shared mailbox.

send (A, msg) send msg to mailbox A

receive (A, msg) Receive msg from mailbox A.

Properties of link:

- A link is established only if both members of the pair have a shared mailbox
- A link may be associated with more than 2 processes.
- B/n each pair of communicating process, a no. of different links may exist, with each link corresponding to one mailbox.

A mailbox can be owned by a process or an OS. If a process owns a mailbox, and, that process terminates then the communication ends.

If a mailbox is owned by an OS, it should allow any process to

- Create a new mailbox
- send & receive msg through the mailbox
- Delete a mailbox.

* Synchronization:

Msg passing may be

Blocking send: The sending process is blocked until the msg is received by the receiving process or by the mailbox

Non blocking send: The sending process sends the msg & resumes operation.

Blocking receive: The receiver blocks until a msg is available

Non blocking receive: The receiver retrieves either a valid msg or a null.

* Buffering:

The exchanging msgs reside in a temporary queue.

3 ways of implementation.

Zero capacity:

The queue has a maximum length of zero 0.

Thus the link cannot have any msges waiting in it.

Bounded capacity:

The queue has finite length n, thus atmost n msgs can reside in it.

Unbounded capacity:

Length of the queue is infinite. So sender never blocks.

CPU scheduling.

Basic Concepts :

CPU – I/O Burst cycle:

Process execution consists of a cycle of CPU execution and I/O wait. Execution stats with a CPU burst followed by I/O burst, which is followed by another CPU burst and so on

CPU scheduler

The short term scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Preemptive scheduling & Non Preemptive scheduling.

Scheduling works when,

1. When a process switches from the running state to the waiting state.
2. Running state to the ready state
3. waiting state to ready state
4. When a process terminates.

For 1 & 4 there is no scheduling needed.
1 & 4 conditions are called as non preemptive or cooperative scheduling.

The 2 & 3 refers to preemptive scheduling.

The resources are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

Dispatcher:

The dispatcher is the module that gives control of the CPU to the process selected by the short term scheduler. This fn involves

- switching context
- switching to user mode
- jumping to the proper location in the user pgm to restart that pgm.

The time taken for the dispatcher to stop one process and start another running is known as the dispatch latency.

SCHEDULING CRITERIA

CPU utilization:

CPU should be busy always. CPU utilization varies from 0 to 100%. For a real sys, it should be 40% to 90%.

Throughput:

No. of processes completed per time unit is called as throughput.

Turnaround time:

The interval from the time of submission of a process to the time of completion is the turnaround time. This is the sum of periods spent waiting to get into mry, waiting in the ready queue, executing on the CPU and doing I/O.

Waiting time:

Waiting time is the sum of the periods spent waiting in the ready queue.

Response time:

Time from the submission of a request until the first response is produced. This is the amount of time to start responding but not the time that it takes to output that response.

We want to maximize CPU utilization & throughput and to minimize turnaround time, waiting time & response time.

SCHEDULING ALGORITHMS

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU.

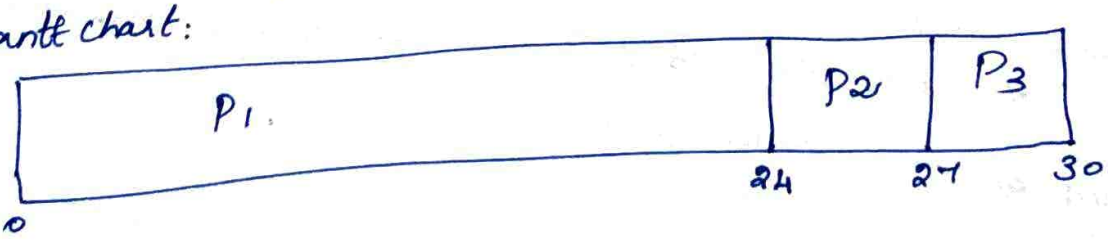
① First come, First served scheduling: (FCFS)

The process that requests the CPU first is allocated the CPU first. This is managed with a FIFO queue. When the process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

- Average waiting time is long.

Process	Burst Time
P_1	24
P_2	3
P_3	3

Gantt chart:



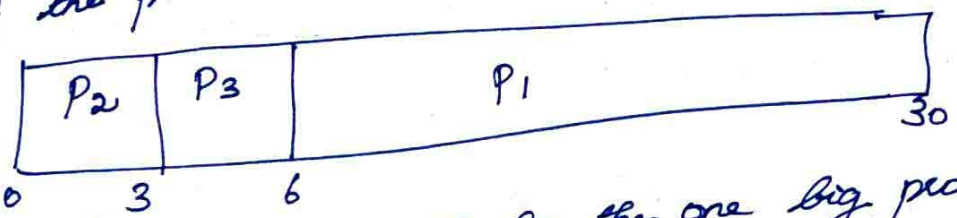
~~Average~~ waiting time of $P_1 = 0$

$P_2 = 24$ millisec.

$P_3 = 27$ millisec.

Average WT = 17 millisec.

If the process arrive in the order P_2, P_3, P_1, the AWT = 3 millisec

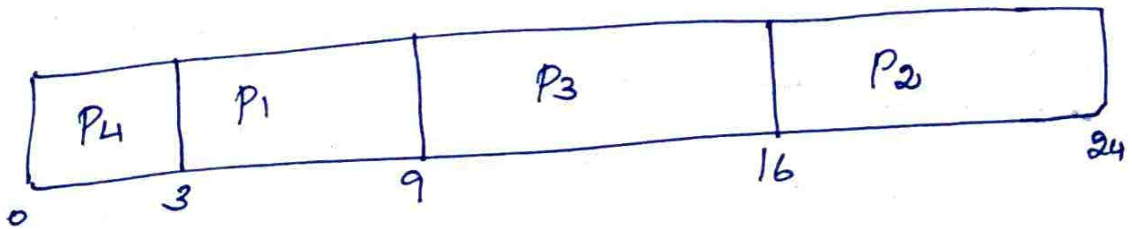


Convoy effect:
All the other processes wait for the one big process to get off the CPU.

2. Shortest Job-First Scheduling (SJF):

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of 2 processes are the same, FCFS scheduling is used to break the tie.

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



WT of $P_1 = 3$ msec

$P_2 = 16$ msec

$P_3 = 9$ msec

$P_4 = 0$ msec

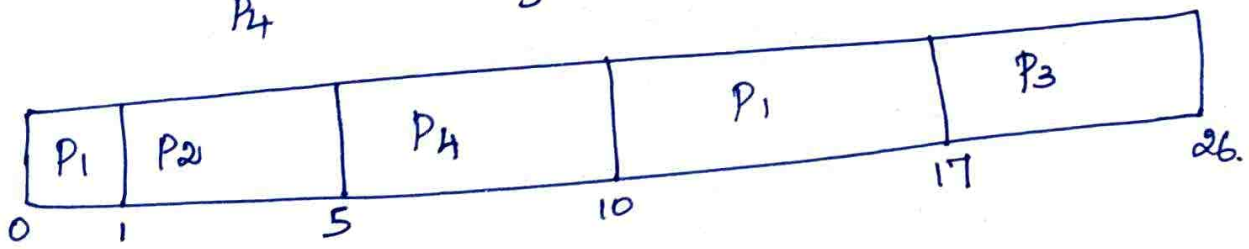
AWT = 7 milliseconds

Advantage:
- Average waiting time (AWT) decreases.

Disadvantage:
- Knowing the length of the next CPU request.

The SJF may be either preemptive or nonpreemptive. A preemptive SJF algm will preempt the currently executing process, whereas a nonpreemptive SJF will allow the currently running process to finish its CPU burst. SJF is also called as shortest-remaining-time first scheduling.

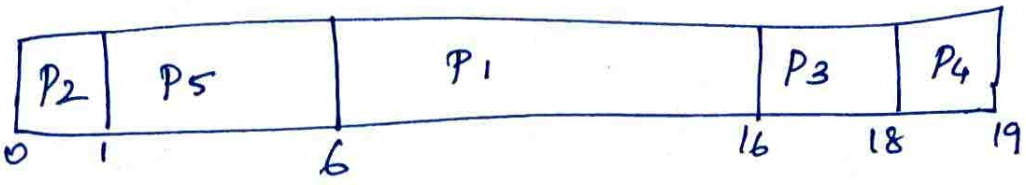
Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



3. Priority Scheduling:

A priority is associated with each process, & the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



AWT = 8.2 ms.

Priorities can be defined internally or externally. Internally priority are set by time limit, they requirements, no. of open files etc. Externally priority are set by the importance of the process, type and amount of funds being paid for computer use etc.

Priority scheduling can be either preemptive or non preemptive. When a process arrives at the ready queue, its priority is compared with the running process. A preemptive scheduling will preempt the CPU if the priority of the newly arrived process is higher than the currently running process. A non preemptive algm will simply put the new process at the head of the ready queue.

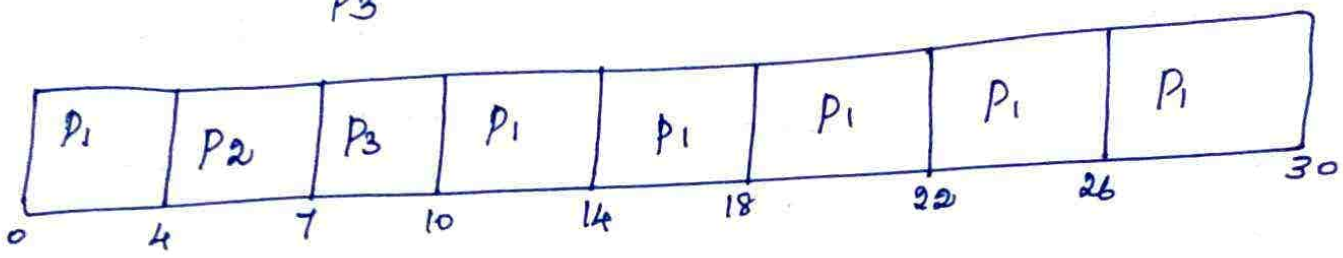
Disadvantage:
- Starvation: Lowest priority process will have to wait for a long time if any higher priority process enters the ready queue leading the lowest priority process into starvation.

4. Round Robin scheduling:

A Round robin scheduling is designed for time sharing systems. It is similar to FCFS scheduling, but preemption is added to switch b/n processes. A small unit of time called time quantum is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Process	Burst time
P_1	24
P_2	3
P_3	3

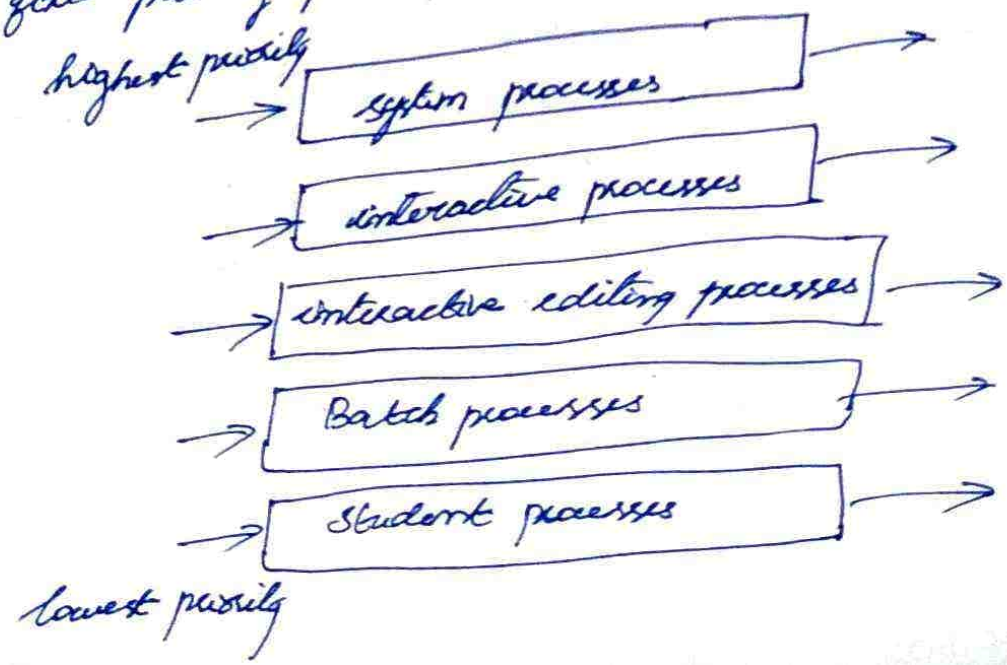
Time quantum = 4 ms.



The performance of the RR algm depends heavily on the size of the time quantum. If the time quantum is too large, then RR works similar to FCFS. If the time quantum is very small, the RR approach is called as processor sharing.

5. Multilevel Queue Scheduling.

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. The process are permanently assigned to one queue based on some property of the process like memory, size, process priority or process type. Each queue has its own scheduling algm. Also each queues should be scheduled. This is normally implemented as fixed priority preemptive scheduling.



6. Multilevel Feedback Queue Scheduling:

In multilevel queue scheduling, the processes cannot be switched b/n the queues once they are scheduled. Multilevel feedback queue scheduling, allows processes to move b/n queues. If a process uses too much CPU time, it will be moved to a lower priority queue. Similarly, a process that waits too long in a lower priority queue may be moved to a higher-priority queue. This prevents starvation.

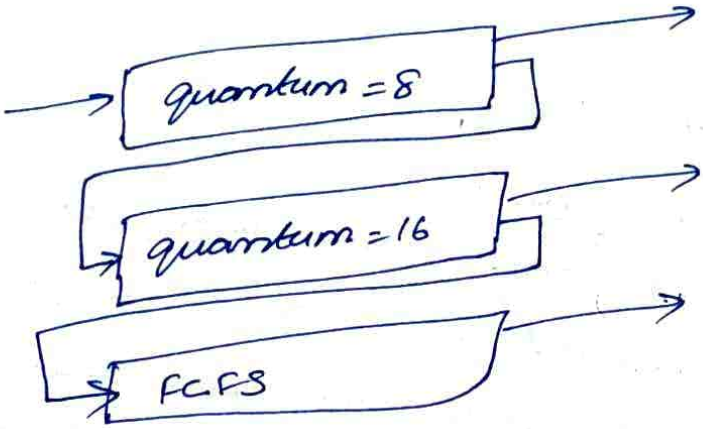


fig:- multilevel feedback queues.

MULTIPROCESSOR SCHEDULING

If multiple CPUs are available, the scheduling problem is more complex.

Homogeneous:- processors are identical in terms of functionality here any processor can run any processes in the queue.

Heterogeneous: Different processors.

Only programs compiled for a given processor's instruction set could be run on that processor.

If several identical processors are available, then load sharing can occur. Each processor has a separate queue. A situation may occur like one processor could be idle

while other processors may be busy. To prevent this situation, a common ready queue can be used. So all processes go into one queue and are scheduled onto any available processor.

Two scheduling approaches may be used.

① Each processor is self scheduling. Two processors do not choose the same process and that processes are not lost from the queue.

② One processor is ~~scheduled for~~ appointed as a scheduler for the other processors, thus creating a master-slave struct

REAL-TIME SCHEDULING

Hard real-time system:

Within a time the critical task must be completed. The process ~~to~~ be executed is submitted along with the time limit. The ~~proce~~ scheduler schedules the process to a processor only if it can complete the execution within the time limit; if not it is not scheduled. This process of allocation is called as resource reservation.

Soft real-time system:

No condition to be executed within the time limit. This may result in longer delays or starvation.

To implement a soft real time sys. 2 aspects are there,

① sm/ should have priority scheduling and real time processes must have the highest priority.

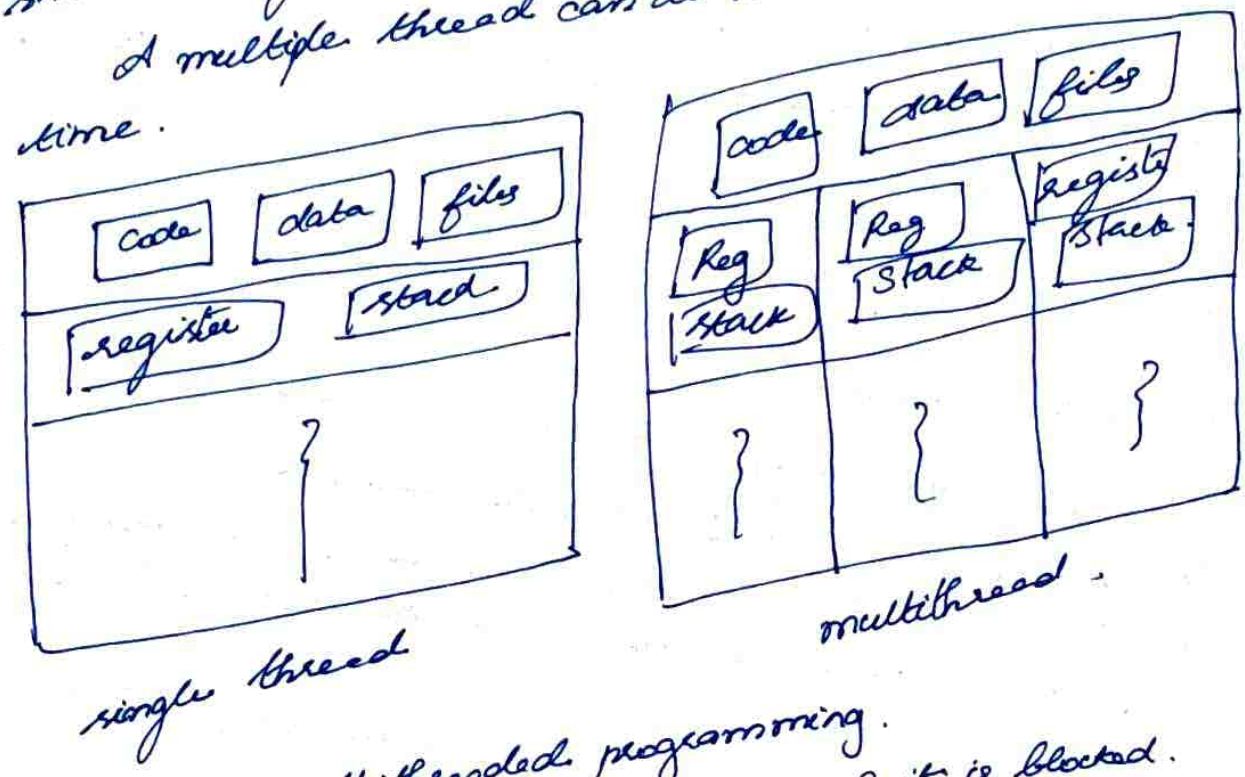
② Dispatch latency must be small.

THREADS

Overview:

A thread is a light-weight process (LWP), it comprises of a thread ID, a program counter, a register set, & a stack. The content of data section, code section, OS resources are shared among all the threads that belong to the same process.

A multiple thread can do more than one task at a time.



single thread

multithread.

Benefits of multithreaded programming.

1. Responsiveness:- A pgm run even if part of it is blocked.
2. Resource sharing – share mry & resources & code sharing.
3. Economy
4. Utilization of multiprocessor architecture.

User and Kernel Threads:

User threads:

User threads are supported above the kernel and are implemented by a thread library at the user level. The library spports for thread creation, scheduling and management with no support from the kernel.

If the kernel is signale-threaded, then any user-level thread performing a blocking system call will cause the entire process to block, even if other threads are available.

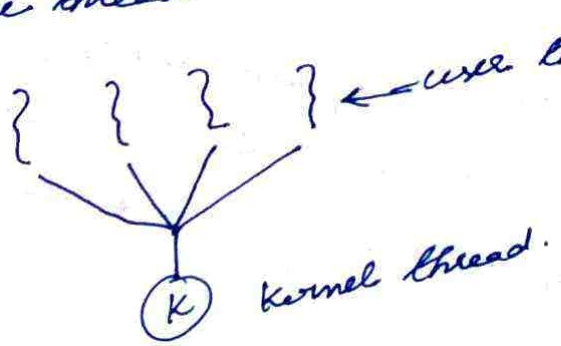
Kernel threads:

Kernel threads are supported by the OS. Kernel performs thread creation, scheduling & management in kernel space. If a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.

MULTITHREADING MODELS:

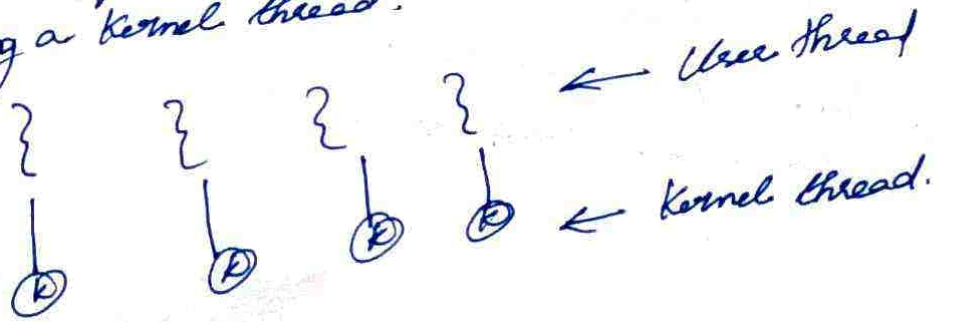
① Many to one model:

This maps many user-level threads to one kernel thread. Thread mgmt is done in user space. Entire process will block if a thread makes a blocking s/m call. Only one thread can access the kernel at a time.



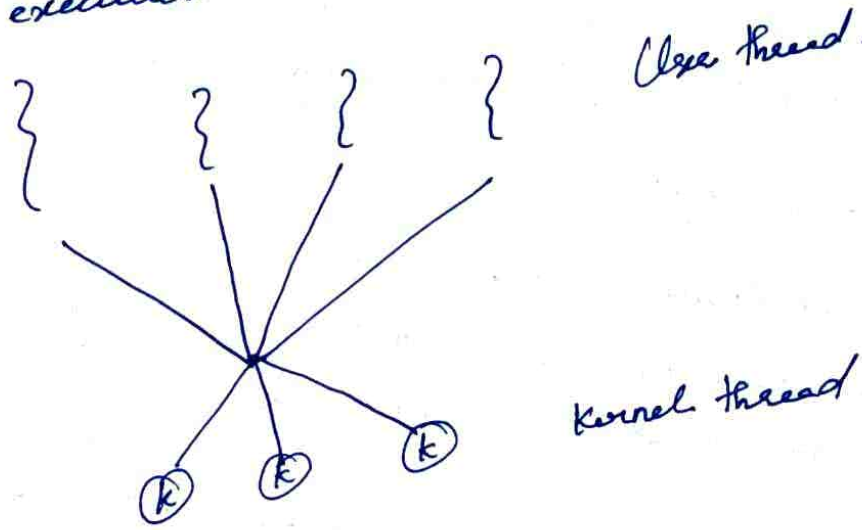
② One to One model:

Maps each user thread to a kernel thread. This provides concurrency. It allows multiple thread to run in parallel on multiprocessors. Creating a user thread requires creating a kernel thread.



③ Many to many model.

This multiplexes many user-level threads to a smaller or equal no. of kernel threads. Any number of user threads can be created. When a thread performs a blocking system call, the kernel can schedule another thread for execution.



THREADING ISSUES

The fork and exec System calls:

A fork system call duplicates either the only one thread that invokes the fork or all the threads.

A thread invokes the exec system call, the program specified in the parameter to exec will replace the entire process - including all threads and LWPs.

Cancellation:

Thread cancellation is the task of terminating a thread before it has completed.

e.g. Multiple threads searching through a database, and one thread returns the result, so the remaining threads might be cancelled.

2 methods for cancellation:

Asynchronous cancellation:
One thread immediately terminates the target thread.

Deferred cancellation:
The target thread can periodically check if it should terminate, allowing the target thread an opportunity to terminate itself in an orderly fashion.

Signal handling:
A signal is used in UNIX system to notify a process that a particular event has occured. A signal may be received either synchronously or asynchronously.

Steps:
1. A signal is generated by the occurence of a particular event.
2. A generated signal is delivered to a process.
3. Once delivered, the signal must be handled.

Synchronous signals are delivered to the same process that performed the operation causing the signal.
When a signal is generated by an event external to a running process, that process receives the signal asynchronously.
Any signal must be handled by either default signal handler or user-defined signal handler.

Every signal has a default signal handler that is run by the kernel when handling the signal. This default action can be overridden by a user-defined signal handler function.

Thread Pools.

Whenever the server receives a request, it creates a separate thread to service the request.

Number of threads are created at the process startup time and are placed into a pool, where they sit and wait for work. When a server receives a request, it awakens a thread from the pool and it is requested to service. Once the thread completes its service, it returns to the pool awaiting more work. If the pool contains no available thread, the server waits until one becomes free.

Benefits of thread pools:

- It is faster to service a request with an existing thread than awaiting to create a new thread.
- A thread pool limits the no. of threads that exist at any one point. This is important on systems that cannot support a large no. of concurrent threads.

Thread specific data:

Threads belonging to a process share the data of the process. Each thread might need its own copy of certain data in some circumstances. These data are called as, thread-specific data.

PROCESS SYNCHRONIZATION

THE CRITICAL SECTION PROBLEM

Consider a system consisting of n processes $\{P_0, P_1, P_2 \ldots P_n\}$. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file and so on.

When one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

The execution of critical section by the process is mutually exclusive in time.

The critical section problem is to design a protocol that the process can use to cooperate. Each process must request permission to enter its critical section. The code implementing this request is the entry section, that is followed by an exit section. The remaining code is the remainder section.

```
do
{
entry section
    critical section
exit section
    remainder section
} while (1);
```

A solution to the critical section problem must satisfy 3 requirements

1. Mutual Exclusion
   If process P_i is executing in its CS, then no other process can be executing in their CS. (only one process can execute the CS)
2. Progress
   If no process is executing in its CS and some processes wish to enter their CS, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its CS next. (If one process doesn't need to execute into CS then it should not stop other process to get into the CS)
3. Bounded waiting
   There exists a bound on the no. of times that other processes are allowed to enter their CS after a process has made a request to enter its critical section and before that request is granted. (The process must not be endlessly waiting for getting into the CS).

Two-Process solutions

The processes are P_0 & P_1. or P_i & P_j.

Algorithm 1

The processes share a common integer variable turn, initialized to 0.

If turn == i, then P_i can execute its CS.

This solution ensures that only one process can be at CS.

This does not satisfy the progress requirement.

eg) if turn == 0 & P_1 is ready to enter its CS, P_1 cannot do so even P_0 may be in its remainder section.

Algorithm 2

Here instead of turn variable an array is used and are initialized to false.

If flag[i] is true, then P_i is ready to enter the CS.

```
do
{
    flag[i] = true;
    while (flag[j]);
    critical section
    flag[i] = false
    remainder section
} while (1);
```

```
do
{
    while (turn != i);
    Critical section
    turn = j;
    remainder section
} while (1);
```

Algm 1.

fig:- Structure of P_i in Algm 2.

- P_i sets flag[i] to true, to enter its CS.
- Then P_i check P_j is not ready to enter its CS.
- If P_j is ready, then P_i wait until P_j is no longer needed to be in the CS.
- Then P_i enters the CS. On exiting the CS, P_i sets P[i] to false.

Mutual exclusion may not be satisfied.

Algorithm 3.

The processes share 2 variables.

```
boolean flag[2];
int turn;
```

Initially flag[0] = flag[1] = false

turn = 0 or 1.

To enter CS, Pi sets flag[i] to true and turn = j, if other process wishes to enter the CS, it can do so.

If both processes wish to enter CS then turn will be set to both i & j.

3 requirements are satisfied.

```
do
{
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = false;

        remainder section.
} while (1);
```

Multiple-Process Solutions

Bakery algorithm

Bakery algorithm solves the critical section problem for n processes. Before entering its CS, process receives a number. Holder of the smallest no. enters the CS. If P_i & P_j receive the same no., and if $i<j$, then P_i is served first.

2 datastructures are,

```
boolean choosing[n];
int number[n];
```

These are initialized to false & 0 respectively.

- $(a,b) < (c,d)$ if $a<c$ or if $a==c$ & $b<d$.
- $\max(a_0, \ldots a_{n-1})$ is a no., k such that $k \geq a_i$ for $i = 0, 1, \ldots n-1$.

```
do
{
choosing[i] = true;
number[i] = max(number[0], number[1], ... number[n-1])+1;
choosing[i] = false;
for (j=0; j<n; j++)
 {
 while (choosing[j]);
 while ((number[j] != 0) && (number[j,j] < number[i,i]));
 }
critical section
number[i]=0;
remainder section
} while (1);
```

SYNCHRONIZATION HARDWARE

The critical section problem can be solved in a uniprocessor environment if we could forbid interrupts to occur while a shared variable is being modified.

Disabling interrupts on a multiprocessor can be time-consuming, as the msg is passed to all the processors.

To resolve this, TestAndSet instruction is used. This instruction is executed atomically.

```
boolean TestAndSet (boolean &target)
{
    boolean rv = target;
    target = true;
    return rv;
}
```

TestAndSet - modifies content of a word

ap - modifies content of 2 words.

```
do
{
while (TestAndSet (lock));
    CS
lock = false;
    RS
} while (1);
```

fig:- Mutual exclusion implementation with TestAndSet.

Mutual exclusion can be implemented by using the TestAndSet instruction, by declaring a boolean variable lock.

If a machine supports the swap instruction, then mutual exclusion can be provided as, Global variable lock is declared are initialized to false

↳ Each process has a local boolean variable key.

```
void swap (boolean &a, boolean &b)
{
boolean temp = a;
 a = b;
 b = temp;
}
```

fig:- Definition of Swap.

```
do
{
key = true
while (key == true)
swap (lock, key)
CS
lock = false
RS
} while (1);
```

fig:- Mutual exclusion with Swap inst

These algorithms do not satisfy the bounded-waiting requirement.

To implement both mutual-exclusion & Bounded-waiting the following algorithm is used.

The common data structures used are,

```
boolean waiting [n];
boolean lock;
```

These are initialized to false.

Process P_i can enter its CS if waiting[i] == false or key == false.

The value of Key can become false only if the TestAndSet is executed.

```
do
{
waiting[i] = true;
key = true;
while (waiting[i] && key)
    key = TestAndSet (lock);
waiting[i] = false;
```

CS

```
j = (i+1) % n;
while ( (j != i) && !waiting[j])
  j = (j+1) % n;
if (j == i)
  lock = false
else
  waiting[j] = false;
```

RS

```
} while (1);
```

SEMAPHORES

The solutions to the critical section problem are not easy to more complex problems. To overcome this, a synchronization tool called semaphore is used.

A semaphore S is an integer variable, which is accessed only through 2 standard atomic operations wait & signal

wait() operation is termed as P and signal() is termed V.

```
wait(S)
{ while(S<=0)
  ;
  S--;
}
```

```
signal(S)
{
S++;
}
```

when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Two types of semaphores:

Counting semaphore :- The value can ranges over an unrestricted domain

Binary semaphore :- The value range b/n 0 & 1. This behave like a mutex lock.

Counting semaphore:

- control access to resources.
- value is initialized to no of available resources.
- Process wish to use resource performs a wait() operation on the semaphore (decrementing the count)
- when the process releases a resource, it performs signal() (incrementing the count).
- when count of semaphore = 0 i.e, all resources are used.

Semaphore implementation:

To avoid busy waiting, when a process has to wait, it will be put in a blocked queue of processes waiting for the same event.

```
typedef Struct
{ int value;
  Struct process *L;
} semaphore;
```

Two operations

block - suspends the process that invokes it.

wakeup (P) - resumes the execution of a blocked process P.

Semaphore operations:

Each semaphore has an integer value & a list of processes list. When a process waits on a semaphore, it is added to the list of processes. A signal() operation removes one process from the list of waiting process and awakens that process.

```
wait (semaphore *S)
{
    S->value--;
    if (S->value < 0)
    {
        add this process to S->list;
        block();
    }
}
```

```
signal (semaphore *
{
S->value++;
if (S->value <=0)
{
remove process P from
        S->list;
wakeup(P);
}
}
```

Deadlock & Starvation:

The implementation of semaphore with a waiting queue may result in a situation where 2 or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes. These processes are said to be deadlocked.

MUTEX LOCKS.

Mutex lock is the s/w solution for the critical section problem. A process must acquire the lock before entering a CS and it releases the lock when it exits the CS. The acquire() fn acquires the lock and the release() fn is used to release the lock.

The mutex lock has a boolean variable 'available'. This value indicates if the lock is available or not. If the lock is available, a call to acquire() fn succeeds, then the lock is considered as unavailable. A process that attempts to acquire an unavailable lock is blocked until the lock is released.

```
acquire()
{
while(!available)
   /* busy wait */
available = false;
}
```

```
release()
{
available = true;
}
```

Solution to CS problem using mutex lock

```
do
{
acquire lock
    CS
Release lock
    Remainder section
} while(true);
```

The main disadvantage here is busy waiting. While a process is in CS, any other process that tries to enter its CS must loop continuously in the call to acquire(). This type of lock is called spin lock.

Consider P_0 & P_2 processes accessing 2 semaphores S & Q set to value 1.

P_0	P_1
wait (S)	wait (Q)
wait (Q)	wait (S)
⋮	⋮
signal (S)	Signal (Q)
Signal (Q)	Signal (S)

This is a deadlock condition.

Starvation:— Process wait indefinitely within the semaphore.

Priority inversion:— scheduling problem when lower priority process hold a lock needed by higher priority process. This problem can be solved by priority inheritance protocol — All processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question.

Eg). L, M, H are process with low, medium & high priority. L uses a resource R and H waits for R. Now the priority is inherited to H by L. So after L releases the resource, it will be used by process H.

CLASSICAL PROBLEMS OF SYNCHRONIZATION

1. Bounded Buffer problem
2. Readers-writers problem
3. Dining philosophers problem

Bounded buffer problem:

The producer & consumer processes share the following data structures.

```
int n;
Semaphore mutex = 1
Semaphore empty = n
Semaphore full = 0
```

The pool consists of n buffers, each capable of holding one item. The producer produces full buffer for the consumer of the consumer produces empty buffer for the producer.

Reader-writers problem:

A data can be shared among several concurrent processes. Some may read the data and others may update the data. They are referred to as readers and writers respectively. If 2 process reads a data simultaneously no effect occurs, but if a writer and some other access the shared data simultaneously chaos occurs.

Solution:—

1. No reader should wait for other readers to finish simply because a writer is waiting.

② If a writer is waiting to access the object, no new readers may start reading.

Both solutions results in starvation. In case 1 writer may starve & in case 2 reader starve.

Best solution :—

Reader processes share the data structures.

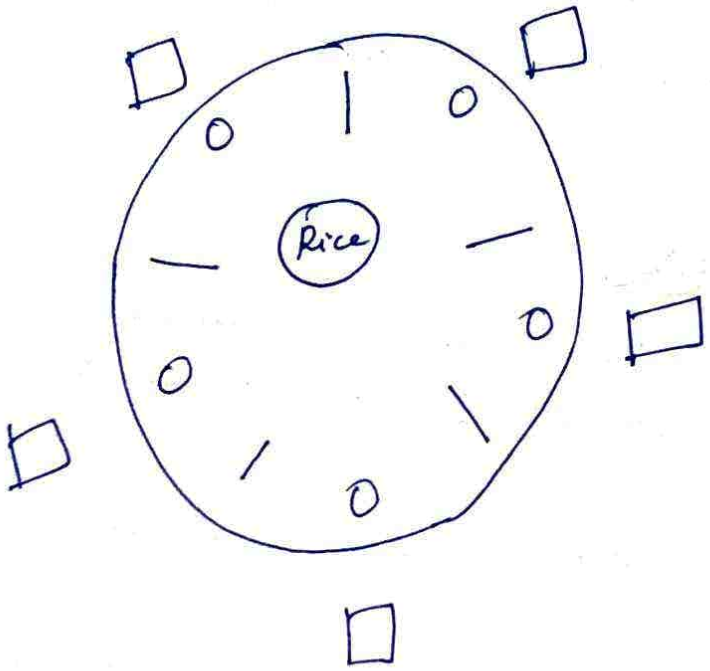
```
semaphore mutex, wrt;
int readcount;
```

Initially
mutex = 1
wrt = 1
readcount = 0

- wrt is common to both reader & writer.
- mutex is used to ensure mutual exclusion when readcount is updated.
- readcount keeps track of how many processes are currently reading the object.
- semaphore wrt fns as mutual-exclusion for the writers. It is also used by readers who enters or exit while other readers are in their CS.
- If a writer is in the CS and n readers are waiting, then one reader is queued on wrt, and n-1 readers are queued on mutex.
- when a writer executes signal(wrt), we may resume the execution of either the waiting reader or a single waiting writer.

Dinning philosopher problem:

Considers 5 philosophers share a common circular table. In the center of the table is a bowl of rice; and the table is laid with 5 single chopsticks. When she is hungry, she has to pick up 2 chopsticks nearby her. She can pick up only one chopstick at a time. After eating she puts down both the sticks.



Solution:—

- Each chopstick represents a semaphore.
- philosopher tries to grab a ck by executing a wait operation on that semaphore.
- releases the ck by signal operation on the semaphore

The data structure used here are

semaphore chopstick [5]

Each are initialized to 1.

— If 2 neighbors are eating simultaneously, it must be rejected because it has the possibility of creating a deadlock.

Solution 2 :-

- Allow atmost 4 philosophers to be sitting simultaneously at the table.
- Allow a philo. to pick up her ck only if both cks are available.

+ An odd philo picks up first her left ck & then her right ck, whereas an even philo picks up her right ck & then her left ck.

CRITICAL REGIONS

A process to enter a CS should first operate the wait and to exit, the signal operation is used in semaphore.

(Deadlocks) Errors may occur if they are not properly used

like, signal (mutex);
CS
wait (mutex);

or

wait (mutex)
CS
wait (mutex)

(or)

Both signal or wait are omitted.

To eliminate such deadlocks conditions, 2 constructs are used.

i) Critical Regions

ii) Monitors.

Consider a process consists of some local data, and a sequential program that operate on the data. The local data can be accessed by only the sequential program that is encapsulated within the same process. ie, one process cannot directly access the local data of another process. Processes can share global data.

The critical region requires a variable v of type T, T refers to data share among many processes.

v : shared T;

The variable v can be accessed only inside a region statement of the following form,

region v when B do S.

This means, while stmt S is being executed, no other process can access the variable v. B is a boolean expression that governs the access to the critical region. If B is true, S is executed.

region v when (true) S1;
region v when (true) S2;

are executed concurrently in distant sequential process.

The critical region construct guards against simple errors in semaphore solution.

MONITORS

A monitor is characterized by a set of programmer defined operators.

```
monitor monitor-name
{
    shared variable declarations
    Procedure body P1( )
    {
    }
    Procedure body P2( )
    {
    }
        .
        .
    Producer body Pn( )
    {
    }
    {
    initialization code
    }
}
```

fig:— Syntax of monitor.

Procedures defined within a monitor can access only those variables declared locally within the monitor & its formal parameters. Local variables of monitor can be accessed by only the local procedures.

This monitor ensures that only one process at a time can be active within the monitor.

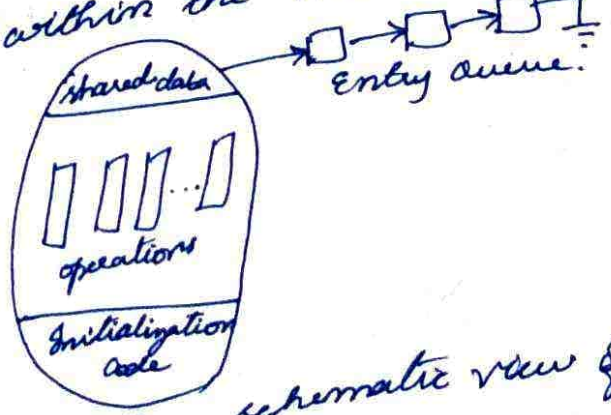


fig:— schematic view of a monitor

A programmer who needs to write her own tailor-made synchronization scheme can define one or more variables of type condition.

condition x, y;

These conditional variables can be accessed only through wait & signal operations.

x.wait() – To suspend a process

x.signal() – To resume (one) a process.

Disadvantages of monitors.

- A process might access the resource without first gaining access permission to that resource.
- A process might never release the resource once it has been granted access to that resource.
- A process might attempt to release a resource that it never requested.
- A process might request the same resource twice (without first releasing that resource).

DEADLOCKS.

A process requests resources, if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

System model:-

Resources may be memory space, CPU cycles, files and I/O devices.

A process must request a resource before using it, and must release the resource after using it. A process can request as many resources it needs.

Request:- If a request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.

Use:- The process can operate on the resource.

Release:- The process releases the resource.

Request and release of resources are system calls. These can be accomplished through the wait & signal operations on semaphores.

A system table records whether each resource is free or allocated to (any) which process. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

DEADLOCK CHARACTERIZATION

A deadlock situation can arise if the following four conditions hold simultaneously.

1. Mutual exclusion :— Atleast one resource must be in non-sharable mode. ie, only one process at a time can use the resource. If another process requests that resource the requesting process must be delayed until the resource has been released.

2. Hold & wait :— A process must be holding atleast one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No Preemption :— Resources cannot be preempted ie, resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait :— A set $\{P_0, P_1 \dots P_n\}$ of waiting process must exist such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by 2 and so on. P_n waits for P_0 to release a resource

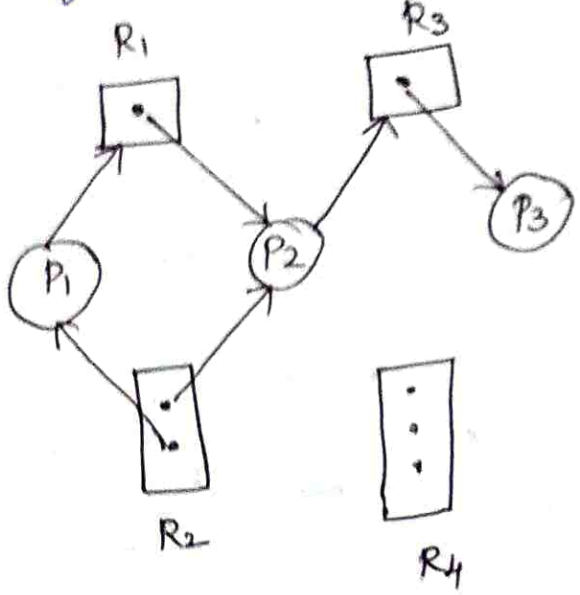
Resource - Allocation Graph:

This is a directed graph. Graph consists of set of vertices V and a set of edges E. There are two different types of nodes $P = \{P_1, P_2 \dots P_n\}$ refers the processes and $R = \{R_1, R_2 \dots R_m\}$ refers the resources.

A directed edge from P_i to R_j is denoted as, $P_i \rightarrow R_j$ Process P_i requests an instance of resource type R_j.

$R_j \rightarrow P_i$ denotes that, Resource R_j has been allocated to process P_i. The directed edge $P_i \rightarrow R_j$ is called a request edge and $R_j \rightarrow P_i$ is called an assignment edge.

Processes are represented in circle & resources in square. When P_i requests an instance of resource type R_j a request edge is inserted in the graph. When this request is fulfilled the request edge is instantaneously transformed to an assignment edge. When the resource is no longer needed the assignment edge is deleted.



sets

$P = \{P_1, P_2, P_3\}$

$R = \{R_1, R_2, R_3, R_4\}$

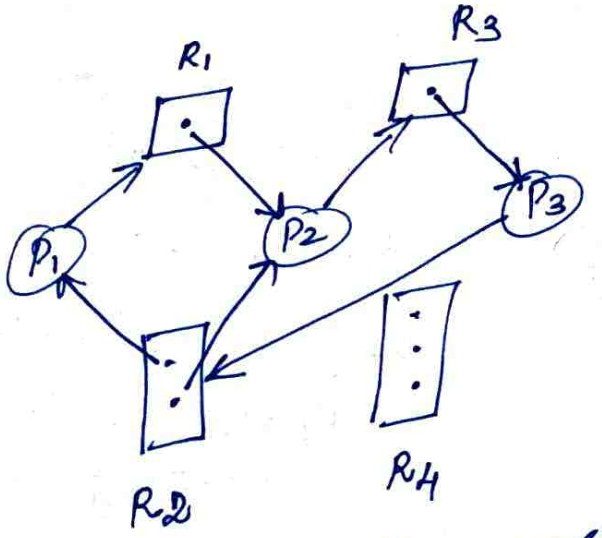
$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$

Resource instances
- one instance of R_1
- 2 instance of R_2
- 1 instance of R_3
- 3 instances of R_4.

Process states:
- P_1 is holding an instance of R_2 and is waiting for an instance of resource type R_1.
- P_2 holds R_2 & R_1. waiting for R_3.
- P_3 holds R_3.

If there is no cycle in the resource allocation graph then there is no deadlock. If there exists any cycle in the graph then there may be any deadlock

If each resource type has exactly one instance then a cycle implies that a deadlock has occured.
If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occured

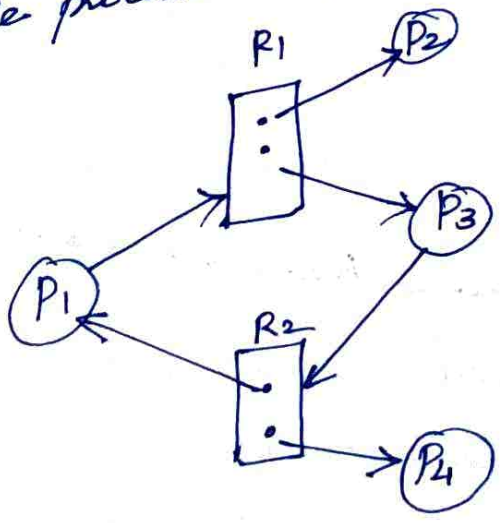


cycles occur.

① $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow$

② $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

This is a deadlock condition. Since P_2 waits for R_3 which is held by P_3. P_3 waits for R_2 which is held by P_2 & P_1. P_1 is waiting for R_1 which is held by P_2. So the process P_1, P_2 & P_3 are deadlock.



cycle exists here,

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

Even there exists a cycle, there is no deadlock because when an instance of R_2 is released by P_4 that can be allocated to process P_3.

METHODS OF HANDLING DEADLOCKS

Three ways to deal with deadlock.

① Protocol can be used to prevent or avoid deadlocks ensuring that the s/m will never enter a deadlock state.

② Allow the s/m to enter into deadlock state, detect it and recover.

③ Ignore the problem & pretend that no deadlock has occured.

Deadlock avoidance and prevention schemes can be used to ensure that deadlock never occurs.

DEADLOCK PREVENTION

To ensure deadlock, 4 conditions like mutual exclusion, hold & wait, No preemption & circular wait must hold. By ensuring that atleast one of these condition cannot hold, we can prevent the occurance of deadlock.

1. Mutual exclusion:—

Mutual exclusion must hold for non sharable resources like printer. Sharable resources do not require mutual exclusive access and thus cannot be involved in a deadlock. (eg) Accessing read-only file.

2. Hold and wait:—

To ensure that the hold-and-wait condition never occurs in the system, a process can request any resource only if it does not hold any other resources

2 protocols are present.

① Each process can request and allocate the resources before the execution of the process begins.

② A process may request some resources and use them. Before requesting any additional resources, it should release the other held resource.

Disadvantages of these protocol are low resource utilization and starvation.

No Preemption:—

There should be no preemption of resources that have already been allocated. If this condition does not hold then deadlock can be prevented. This condition can be avoided by the following protocols

① If a process is holding some resource and requests for another resource that cannot be allocated immediately to it, then all resources currently being held are preempted. Then the process can restart only when it regain all its old resources and the new ones.

② If a process requests some resources, first check if it is available. If available, allocate them. If not check if they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resources from the waiting process & allocate them to the requesting process. If the resources are not available or held by a waiting process, the requesting process must wait. The process restarts only when it is allocated the new resources & the old one.

Circular wait :—

One way to ensure that this condition never holds, is to impose a total ordering of all resource types, and to require that each process request resources in an increasing order of enumeration.

Let $R = \{R_1, R_2 \ldots R_m\}$ set of resource type.

Unique integer numbers are assigned to each resource type. Each process can request resources only in an increasing order of enumeration. If a process request R_i, then the process can request R_j if and only if $F(R_j) > F(R_i)$.

integer value of R_j > integer value assigned to R_i.

DEADLOCK AVOIDANCE

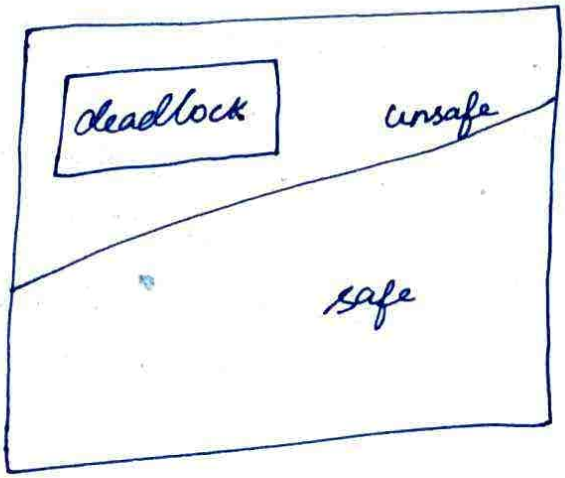
Deadlock prevention algorithms leads to poor device utilization and reduced system throughput.

Deadlocks can be avoided if additional information about the resources & process are known in advance. Maximum number of resources of each type must be requested for each process in advance to execute the avoidance algorithm. A deadlock avoidance algm examines the resource allocation state to ensure that a circular wait condition never exist. The resource allocation state is defined by the no. of available & allocated resources and the maximum demands of the process.

Safe state :

A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. A sequence of processes $\langle P_1, P_2 \ldots P_n \rangle$ is a safe sequence

for a current allocation state if, for each P_i, the resources that P_i can still request can be satisfied by the currently available resources plus the resources held by all the P_j with $j < i$.



Not all unsafe state are deadlock.

If P_i resource needs are not available, P_i can wait until all P_j have finished.
- when P_j is finished, P_i can obtain needed resources, execute, return allocated resources and terminate.
- when P_i terminates, P_{i+1} can obtain its needed resources

Unsafe state:

If no such safe sequence exists, then the system state is said to be unsafe.
- If a system is in a safe state then no deadlock.
- If a system is in unsafe state possibility of deadlock
- Avoidance ensure a system will never reach an unsafe state.

Eg: There exists a total of 12 tape drives.

Process	Max Needs	Allocated	current Needs
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Free system 3

The sequence <P1, P0, P2> is a workable sequence.

At time t0,

No. of free tape drives = Total no. of tape drives − No. of tape drives allocated.

$= 12 - (5+2+2)$

$= 3$

Process P1.

P1 is allocated with all its tape drives and returned it.

Process	Max Needs	Allocated	Current Needs
P0	10	5	5
P1	4	4(2+2)	-
P2	9	2	7
		Free system	3
		Free (Before Return)	1
		Free After Return	5

Process P0

P0 is allocated with all its tape drives & returned it.

Process	Max Needs	Allocated	Current Needs
P0	10	10(5+5)	-
P1	4	4	-
P2	9	2	7
		sym free	5
		Free Before Return	0
		Free after return	10

Process P_2

P_2 is allocated with all its tape drives & returned it

Process	Max needs	Allocated	Current Needs
P_0	10	10	-
P_1	4	4	-
P_2	9	9 (2+7)	-

Free available 10

Free Before Return 3

Free after return 12

Suppose P_2 requests and is given one more tape drive,

Process	Max Needs	Allocated	current needs
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

Sys free 2

Process P_1

P_1 is allocated with all its tape drives & returned it.

Process	Max needs	Allocated	Current needs
P_0	10	5	5
P_1	4	4 (2+2)	-
P_2	9	3	6

Free 2

Before Return 0

After Return 4

Process P_0

P_0 is put in wait state since P_0 requested resources (5 tapes) are unavailable.

Process	Max needs	Allocated	Current Needs
P_0	10	5(7 free)	5
P_1	4	4	-
P_2	9	3	6
		Free sys	4
		Free Before Return	0
		Free After Return	4

Process P_2

P_2 is also put in wait state since P_2 requested resources (6 tapes) are unavailable.

Process	Max needs	Allocated	Current Needs
P_0	10	5	-
P_1	4	4	-
P_2	9	3 (3+4) 7	-
		Free	4
		Before Return	0
		After return	4

Now P_0 & P_2 result in deadlock. So the sequence $\langle P_1, P_0, P_2 \rangle$ is a unsafe sequence.

Avoidance algorithms.

Single instance of a resource type
- Use a resource-allocation graph

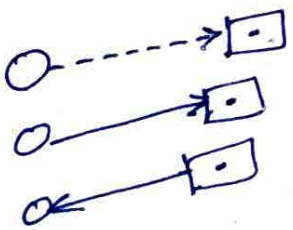
Multiple instances of a resource type.
- Use Banker's algorithm.

Resource-allocation Graph algorithm.

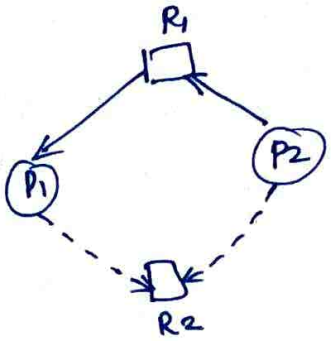
Claim edge :-

Claim edge is an edge in which $P_i \rightarrow R_j$ indicates that P_i may request resource R_j, represented by a dashed line. A claim edge is converted to request edge when a process requests a resource. When a resource is released by a process, assignment edge is reconverted to claim edge. Resources must be claimed a priori in the system.

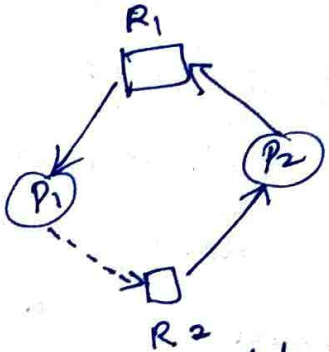
If request assigned does not results on a cycle - then it is in safe state else unsafe state.



Suppose that P_i requests a resource R_j, the request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph using cycle-detection algm which requires an order of n^2 operations.



Graph for deadlock avoidance



unsafe state.

P_2 requests R_2 which is currently free cannot be allocated to P_2, since a cycle will be created in the graph.
If P_1 requests R_2 & P_2 requests R_1, then deadlock will occur.

Banker's algorithm.

- This is a deadlock avoidance algm which is applicable to sym with multiple instances of resource types.
- Each process must a priori claim maximum use.

Data Structures of Banker's algm:

Let n = no. of processes & m = no. of resources types.

Available :— Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available.

Max :— $n \times m$ matrix. $Max[i,j] = k$ means that process P_i may request atmost k instances of R_j.

Allocation :— $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j.

Need :— $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Safety Algorithm

- Used to find whether a s/m is in safe state or not.

① Let work & finish be vectors of length m & n respectively.

Initialize work = Available.

Finish[i] = false for i = 1, 2 ... n,

② Find an i such that both:

Finish[i] = false

Need ≤ work

If no such i exists, go to step 4.

③ work = work + allocation

Finish[i] = true

go to step 2.

④ If Finish[i] == true for all i, then the s/m is in a safe state.

Time taken:

Let requires $m \times n^2$ operations.

Resource-Request algorithm for process P_i:

Let $Request_i$ = request vector for process P_i.

If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i \leq Need_i$ go to step 2. Otherwise raise error condition, since process has exceeded its maximum claim

2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.

3. Pretend to allocate requested resources to P_i by modifying the state as follows.

Available = Available − Request$_i$;
Allocation$_i$ = Allocation$_i$ + Request$_i$
Need$_i$ = Need$_i$ − Request$_i$

If safe, the resources are allocated to P_i.
If unsafe, P_i must wait & the old resource-allocation state is restored.

eg) Consider a system with 5 process P_0 through P_4 & 3 resource types.

A − 10 instances
B − 5 instances
C − 7 instances.

At time T_0.

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Need = Max − Allocation

Process	Need A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

① Check Need ≤ work.

P_0 7 4 3 ≤ 332 false.

✓ P_1 1 2 2 ≤ 332 true. Available = 200 + 332 = 532

P_2 600 ≤ 532 false.

✓ P_3 011 ≤ 532 true Available = 532 + 211 = 743

✓ P_4 431 ≤ 743 true Availab = 743 + 002 = 745

✓ P_2 600 ≤ 745 true Available = 745 + 302 = 1047

✓ P_0 7 4 3 ≤ 1047 true.

Available = 1047 + 010

Final Available ⇒ 1057.

The s/m is in a safe state since the sequence <P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

If P_1 requests (1 0 2)

Check that Request ≤ Available

1 0 2 ≤ 3 3 2 true

Remaining Available resources = 2 3 0

(3 3 2) − (1 0 2)

Allocation — Max.

Process	Allocation A B C	Need A B C	Available A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.

Disadvantages of banker's algorithm.

- Requires fixed no. of resources to allocate
- Users should state their maximum needs in advance
- Algm requires the process to return all resources within a finite time
- No. of users should remain fixed
- Algm requires that the bankers grant all request within a finite time.

DEADLOCK DETECTION

Let deadlock to occur, then detect and recover.

Find:- An algorithm that examines the state of the s/m to determine whether a deadlock has occured.

Recover:- An algm to recover from the deadlock.

Detection Algorithm:

- Single instance of a resource type. Use a wait for graph.
- Multiple instances of a resource type. Use algm similar to Banker's algm.

Single instance of each resource type

wait for graph :— A graph obtained from the resource allocation graph by removing the nodes of type resources and collapsing the appropriate edges. An edge from P_i to P_j implies that P_i is waiting for P_j to release any resource.

Deadlock detection

- Maintain a wait for graph.
- Nodes are processes.
- $P_i \rightarrow P_j$, if P_i is waiting for P_j.

Periodically invoke an algm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.

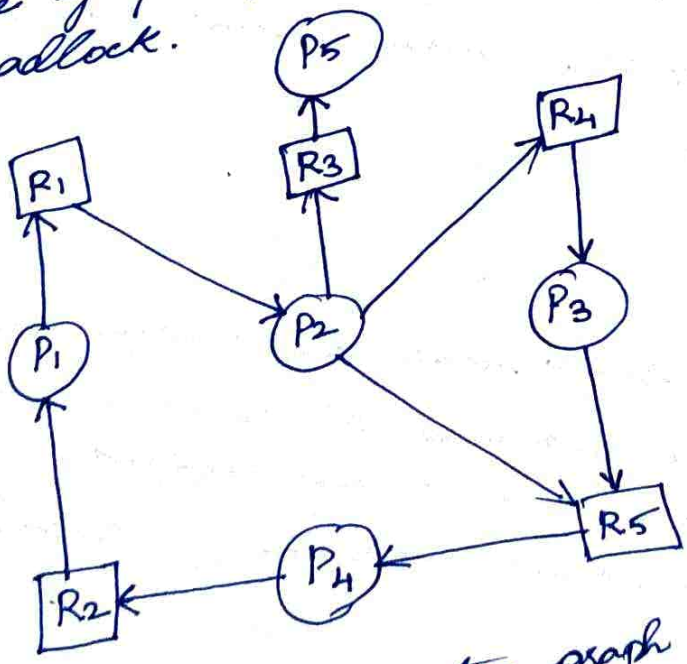


fig:— Resource allocation graph

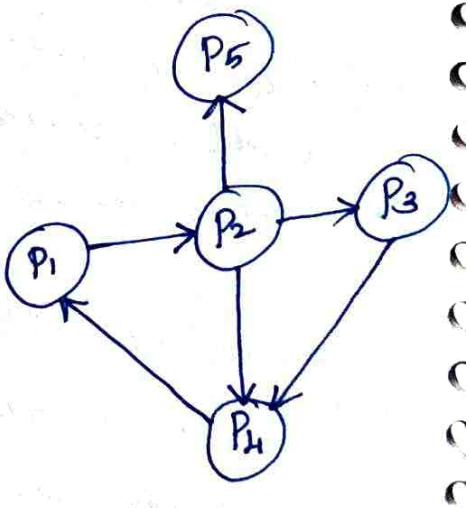


fig:— wait for graph.

7, 16, (22) 58 302, 303, 304

Several instances of a resource type.

Data structures used:

· Available :— A vector of length m indicates the no. of available resources of each type.

· Allocation :— A n x m matrix defines the no. of resources of each type currently allocated to each process.

· Request :— A n x m matrix indicats the current request of each process. If Request [i,j] = K then P_i is requesting K more instances of resource type R_j.

Detection-Algorithm Usage.

1. Let work & finish be vectos of length m & n respectively initialize

work = Available for i = 1, 2 .. n

if Allocation i ≠ 0, then

Finish [i] = false;

else, Finish [i] = true.

2. Find an index i such that both,

Finish [i] = false

Request i ≤ work

If no such i exists, go to step 4.

3. work = work + Allocation

Finish [i] = true

goto step 2

4. If Finish [i] = false, for some i, 1 ≤ i ≤ n then deadl

if Finish [i] = false then P_i is deadlocked.

(8) Consider 5 processes P_0 through P_4.

Three resource types.

A - 7 instances

B - 2 instances

C - 6 instances.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

If P_2 requests an additional instance of type C

- can reclaim resources held by process P_0, but insufficient resources to fulfill other processes requests.
- Deadlock exists, consisting of processes P_1, P_2, P_3 & P_4.

	Request A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

Detection algm invoked depends on
- How often a deadlock is likely to occur?
- How many processes will need to be rolled back?
- One for each disjoint cycle.

When a deadlock occurs :-

- Every time a request for allocation cannot be granted immediately.
  - Allows us to detect set of deadlocked processes & process that caused deadlock.
  - Extra overhead.
- Every hour or whenever CPU utilization drops.
  - less expensive.

with arbitrary invocation there may be many cycles in the resource graph & we would not be able to tell which of the many deadlocked processes caused the deadlock.

RECOVERY FROM DEADLOCK.

There are 2 approaches to eliminate the deadlock.

① Process termination :— Abort one or more process to break the circular wait.

② Resource preemption :— Preempt some resources from one or more of the deadlocked processes.

Process Termination :—

- Abort all deadlocked processes.
  - All resources returned for reuse.
- Abort one process at a time until the deadlock cycle is eliminated.
  - How to choose order of precedence?
  - Will the process need to be rerun?

Factors that determine which process is chosen to abort

Priority of the process

- How long process has computed, & how much longer to completion. Resources the process has used.

Resources process needs to complete.

- How many processes will need to be terminate
- Is process interactive or batch?

Resource Preemption :—

Resource preemption takes resources from a process and give to other processes until the deadlock cycle is broken

Issues to eliminate deadlocks.

Selecting a victim :— Determine the order of preemption to minimize cost, order of precedence for preempting. number of resources already held, how many more will it need to complete? Amount of CPU time already used.

Rollback :— Return to some safe state, restart process for that state.

Starvation :— same process may always be picked as victim, include no. of rollback in cost factor.

UNIT - III
STORAGE MANAGEMENT

1. MAIN MEMORY

Memory management:—
- Controlling & coordinating computer mry
- Assigning portions called blocks to various running programs to optimize overall system performance.
- Freeing it for reuse when no longer needed.
- Mry mgmt resides in h/w, in the OS & in pgms & apln.

Motivation:—
- Keeps several processes in mry to improve a system's performance
- Allows multiprogramming.

Limitation:—
Requires the entire process to be in mry for execution.

BACKGROUND

Memory:—
- central to the operation of a modern computer sym.
- A large array of words or bytes
- Each word or byte has its own address.

Instruction execution cycle:—
Consists of 2 cyle.
- Fetch cycle
- Execute cycle.

Steps:—
① Instruction is fetched from mry addressed by PC (program counter).

② PC value is incremented to address the next instruction.
③ Instruction decoder decodes the instruction
④ Instruction is then executed
⑤ Results are stored back to the memory.

Basic hardware:—
- Main mry & registers are only storage, CPU can access directly.
- Register access in one CPU clock, but main mry can take many clock cycles.
- Cache sits in btw main mry & CPU registers.
- Protection of mry is required to ensure correct operation.
- Protection is ensured using 2 registers that define logical address space.

Base register :— Holds the smallest legal physical mry address.

Limit register :— Specifies the size of the range.

eg) Base registers = 300040
Limit registers = 120900
The pgm can legally access all address from 300040 through 420939.

Memory space protection :—
Mry protection is a way to control mry access rights on a computer.
CPU hardware compares every address generated in user mode with the registers.

• A pgm executing in user mode attempt to access OS mry or other user's mry results in a trap to the OS, which treats the attempt as a fatal error.

• This prevents a user pgm from modifying the code or data structures of either the OS or other users.

Address Binding:—

• Address binding is the process of converting the (relative or symbolic) address used in a pgm to an actual physical (absolute) RAM address.

• Pgm must be brought into mry & placed within a process for it to be executed.

Multistep processing of a user pgm:—

User pgm go through several steps before being executed.

- Pgm components do not know where in RAM they will be loaded.
- RAM deals with absolute addresses.
- Logical addresses need to be bound to physical addresses at some point.

Binding steps:—

① Compile time:

At compile time, if we know where the process will reside in mry, then absolute code can be generated. If the starting address changes, then recompile this code.

② Load time:-

If it is not known at compile time, where the process will reside in mey, then the compiler must generate relocatable code.

③ Execution time:-

If process can be moved during its execution from one mey segment to another, then binding must be delayed until run time. Special h/w are available for this scheme.

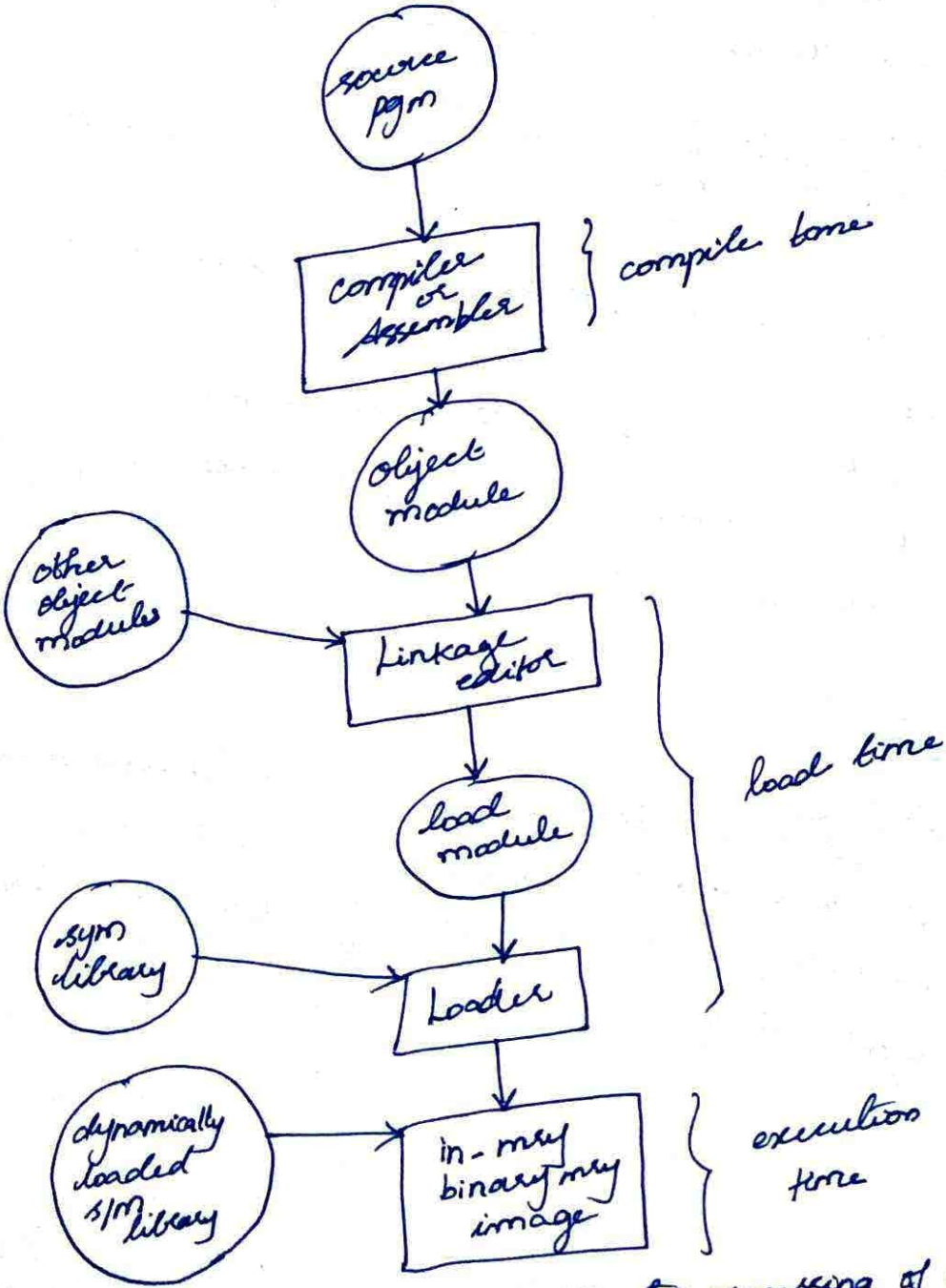


fig:- Multi step processing of a user pgm.

Logical Vs Physical Address space :—

Logical address :—
Address generated by the CPU. Also it is referred to as virtual address.

Physical address :—
Address seen by the mey unit.

Virtual address :—
Logical address results in the execution time.

Logical address space :—
Set of all logical addresses generated by the program is called logical address space.

Physical address space :—
Set of all physical address corresponding to the logical addresses is called physical address space.

Note :
- Logical & physical address are same in compile-time and load-time, address binding schemes.
- Logical & physical address differ in execution-time address binding scheme.

Memory-management Unit (MMU) :—
- MMU is a h/w device that maps virtual to physical address at run time.
- In a simple MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to mey.
- The user pgm deals with logical addresses, it never see the real physical address.

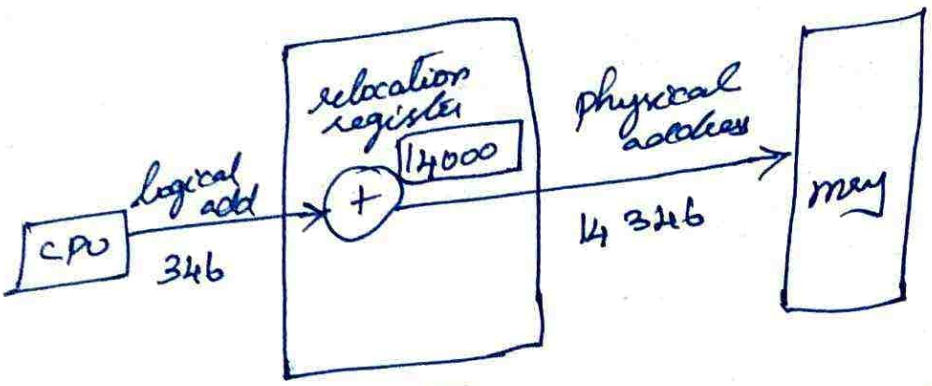


fig:- Dynamic relocation using a relocation register.

Dynamic loading:-

Whole pgm or data for the pgm must be on physical mem for a process to execute. For better mem space utilization, dynamic loading is used. With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main pgm is loaded into mem & is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If not the relocatable linking loader is called to load the desired routine into mem & to update the pgm's address table to reflect this change.

Advantage:-

- Unused routine is never loaded.

Dynamic Linking & Shared libraries:-

Static linking libraries:-

Static linking resolve all addresses before running pgm. Compiler compiles each file, but leaves external references unresolved. But static linker resolves all previously unresolved reference

Advantages:
- changing libraries does not affect previously compiled pgms.
- No address resolution necessary at run time.

Disadvantage:
- The executable size will be larger.
- Multiple copies of code might be in mry.
- New version of library ⇒ must recompile.

Dynamic linking libraries:-
Dynamic linking is the process that links the external shared libraries at run time. It is also called as late linking.

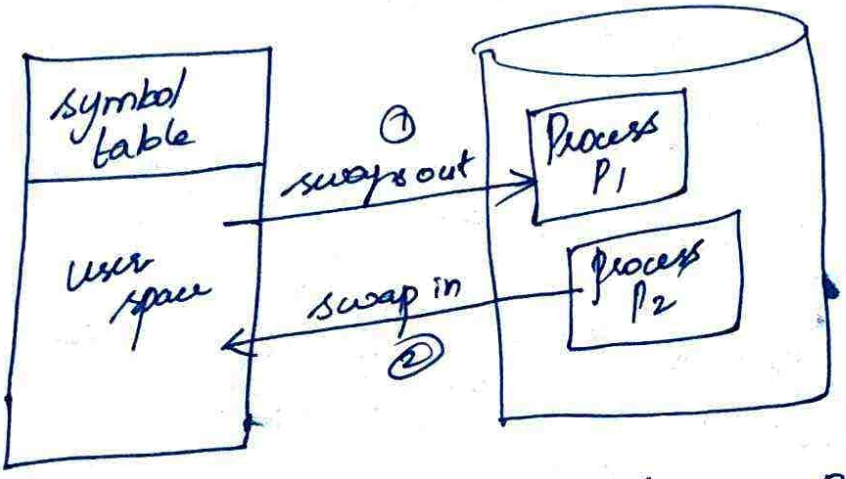
Stub:-
Stub is a piece of code that indicates how to locate the appropriate mry-resident library routine, or how to load the library if the routine is not already present.

It works as,
- A stub is used to replace the whole library routine
- Enough information to locate the library routine.
- When the stub is executed, it replaces itself with the address of the routine and executes the routine. only one copy of the library code is needed for all pgms.
- Such library is also known as a shared library.

One problem with dynamically linked executable is that it is less portable.
statically linked executables are more self-contained.

2. SWAPPING.

A process to be executed should be available in the main mey. The purpose of swapping is to move the process b/n main mey & backing store. All processes are stored in the backing store temporarily.



Swapping are done commonly in Round robin scheduling algorithm and priority-based scheduling algm

Address binding:—

If address binding is done at,

Assembly or load time — swapping can not be moved to different mey location.

Execution time — swapping of process moved to different mey locations.

Backing store :—

Swapping needs a backing store which is a fast disk large enough to accomodate copies of all my images for all users. It must provide direct access to these my images.

S/m maintains a ready queue of ready-to-run processes which have my images on disk.

3. CONTIGUOUS MEMORY ALLOCATION

The main my must accomodate both the OS & the various user processes. The my is divided into 2 parts. 1 for OS & the other for user processes. We want several user processes to reside in the my at the same time. So we need to consider how to allocate available my to the processes that are in the I/P queue. In contiguous my allocation, each process is contained in a single section of my that is contiguous to the section containing the next process.

OS resides in low my and the user processes are present in the high memory.

Memory protection :—

Protection can be achieved by combining relocation register with the limit register. Relocation-register scheme provides an effective way to allow the OS size to change dynamically.

Relocation-register scheme

- Allows protection to the my.
- Allows programs to be relocated to different my starting address as needed.
- Allows my space of OS to shrink or grow dynamically.

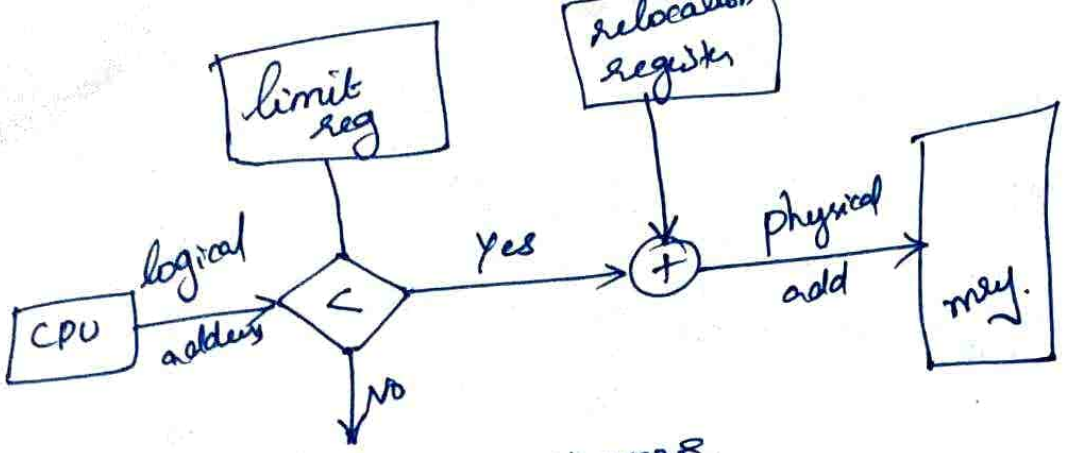


fig:- H/w support for relocation & limit register.

- Relocation register contains value of physical address.
- Limit reg - contains range of the logical add.
- Dispatcher - loads the relocation & limit registers with the correct values as part of the context switch.
- Each logical address must be less than the limit reg
- The MMU maps the logical address dynamically by adding the value in the relocation register which is sent to mey.

Disadvantage:
- poor utilization of process & mey.

Memory allocation:—
① Fixed partition scheme
② Variable partition scheme.

Fixed partition scheme:—
Main mey is partitioned into a no. of fixed sized partitions, where each one should contain only one process. when a partition is free, a process is selected from the i/p queue & is loaded into the free partition (hole). When it terminate, the partition becomes available for next process.

Advantage:
- Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- Supports multiprogramming.

Disadvantage:
- If a prgm is too big to fit into a partition use overlay technique.
- Mry use is inefficient.

Variable partition scheme:—
- Partitions are of variable length & no. Process is allocated exactly as much mry as required.
- Initially all mry is available & is considered as a single block (one big hole).
- OS maintains a table for the info abt.
  - Allocated partitions
  - Free partitions.

Methods to assign process to partitions:— (2 methods)
① Use multiple queues.
- For each process one queue is present.
- Assign each process to the smallest partition within which it will fit, by using the scheduling queues.
- when a new process is to arrive it will put on the queue, it is able to fit without waiting the mry space, irrespective of other blocks queue.

Advantage:- minimize wastage of mry.

Disadvantage:-
Larger partitions are unused.

② Use single queue:—
- Only one ready queue is present.
- If any block is free, even though it is larger than the process, it will simply join instead of waiting for the suitable block size.

Advantage:-
- Simple & minimum processing overhead.

Disadvantage:-
- Small jobs do not use partition space efficiently.

Dynamic allocation placement Algorithm:—

First fit:
- Allocates the first hole that is big enough.
- Process is placed in the first hole it can fit in.

Best fit:
- Allocate the smallest hole is, enough; must search entire list, unless ordered by size.
- Produces the smallest leftover hole.

Worst fit:
- Allocates the largest hole; must also search entire list.
- Produces the largest leftover hole.

Performance issues:
- Worst fit performs worst, since it leaves behind the smallest possible hole.
- First fit is simplest to implement & fastest & best.
- Best fit is little worse than first fit.

Fragmentation:—
This occurs when a sys contains total free mry to satisfy any request, but that cannot be utilized.

Internal fragmentation:
Internal fragmentation is that the allocated mry may be slightly larger than requested mry. this size difference is mry internal to a partition, but not being used.

sln:
- Blocks are split to prevent internal fragmentation

External fragmentation:
Is done when the total mry space exists to satisfy a request, but it is not contiguous. i.e. many small holes.

sln:
- Compaction.
- Non contiguous logical address space.
  - Paging
  - Segmentation.

Compaction:
The goal is to shuffle my contents to place all free my together in one large block.
- This is possible only if relocation is dynamic and is done at execution time.
- Impossible only if relocation is static & is done at assembly or load time.

Compaction algorithm:-
- Move all processes towards one end of my.
- All holes move in the other direction which produce large hole of my.

This is very expensive.

4. PAGING
Paging is a my mgmt scheme that permits the physical address space of a process to be noncontiguous. Paging is handled by hardware.

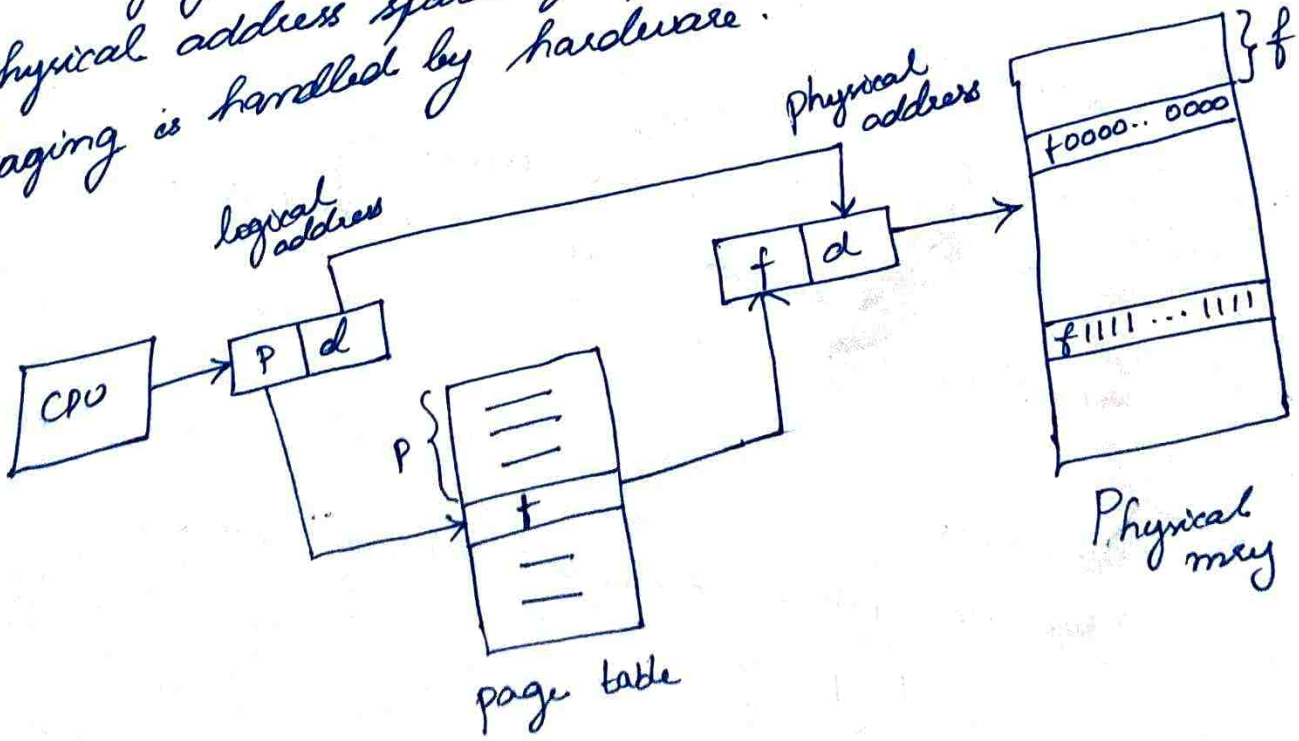


fig:- Paging hardware.

Basic method:-

- Physical mry is divided into fixed size blocks called frames.
- Logical mry is broken into blocks of the same size called pages.
- To execute a process, its pages are loaded into any available mry frames from the backing store.
- The backing store is divided into fixed sized blocks that are of the same size as the mry frames.
- Address generated by CPU is divided into page no. (p) and a page offset (d).
- The page no. is used as an index into a page table.
- The page table contains the base address of each page in physical mry.
- Base address is combined with the offset to define the physical mry.

Paging model of memory:

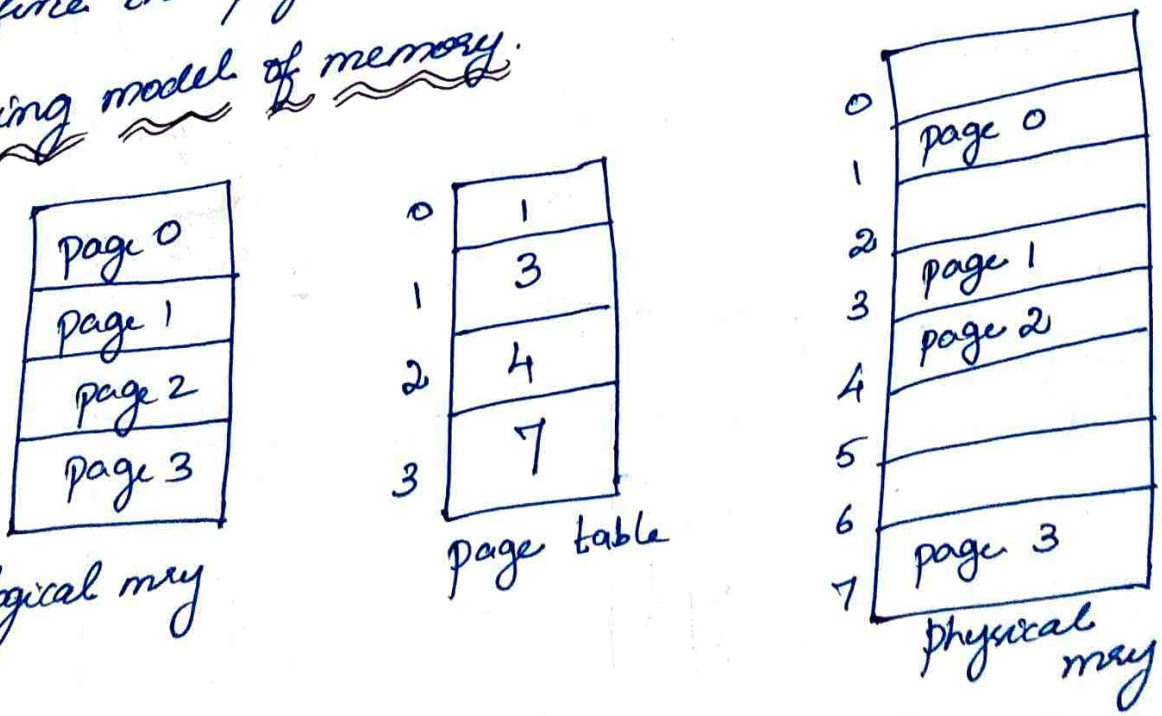
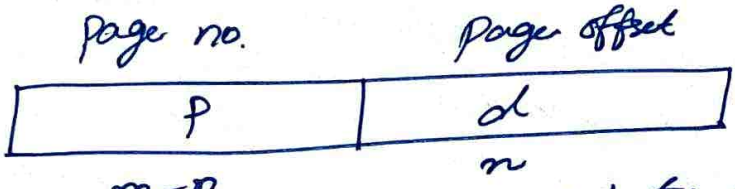


Fig:- Paging model of logical & physical mry.

If size of logical address space is 2^m & page size is 2^n addressing units.

- High order m-n bits of a logical address - page no.
- Low-order n bits of a logical address - page offset

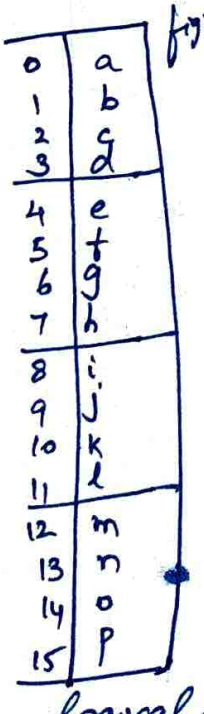


Consider a mry of page size 4 bytes and physical mry 32 bytes (32/4 = 8 pages).

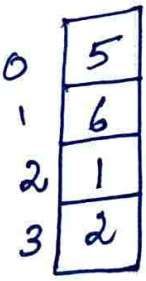
Mapping of logical and physical address.

Logical address (page, offset)	Page Table	Physical Address
0 (0,0)	5	((5×4)+0) = 20
3 (0,3)	5	(5×4)+3) = 23
4 (1,0)	6	((6×4)+0) = 24
13 (3,1)	2	((2×4)+1) = 9

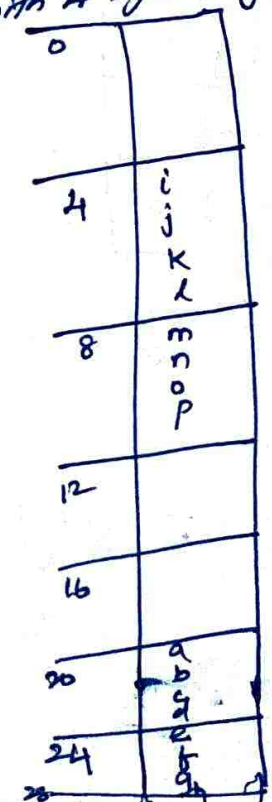
Ex:- paging eg for a 32-byte mry with 4 byte pages.



logical mry



page table



physical

Memory allocation in paging:-

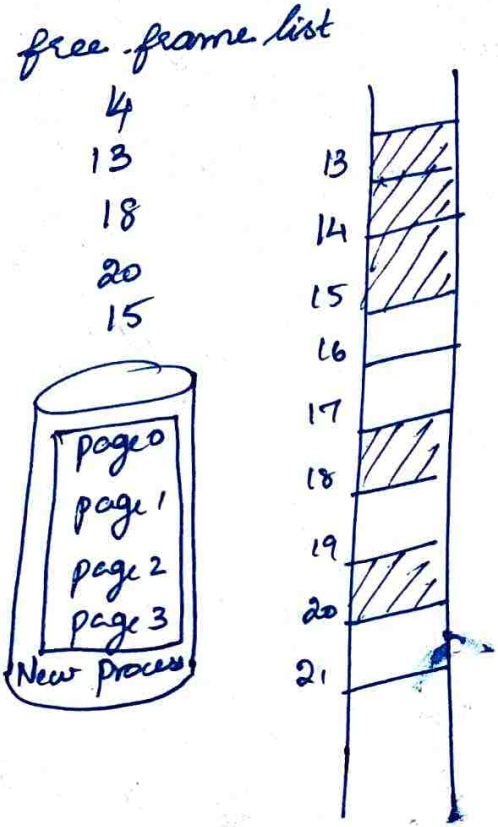


fig:- Before allocation

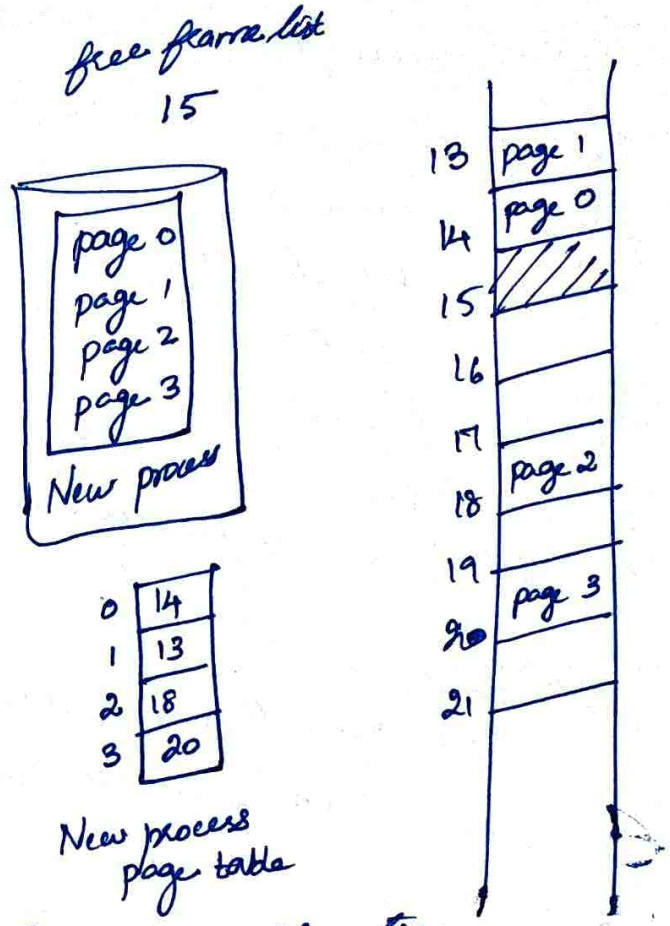


fig:- after allocation

Hardware support:-

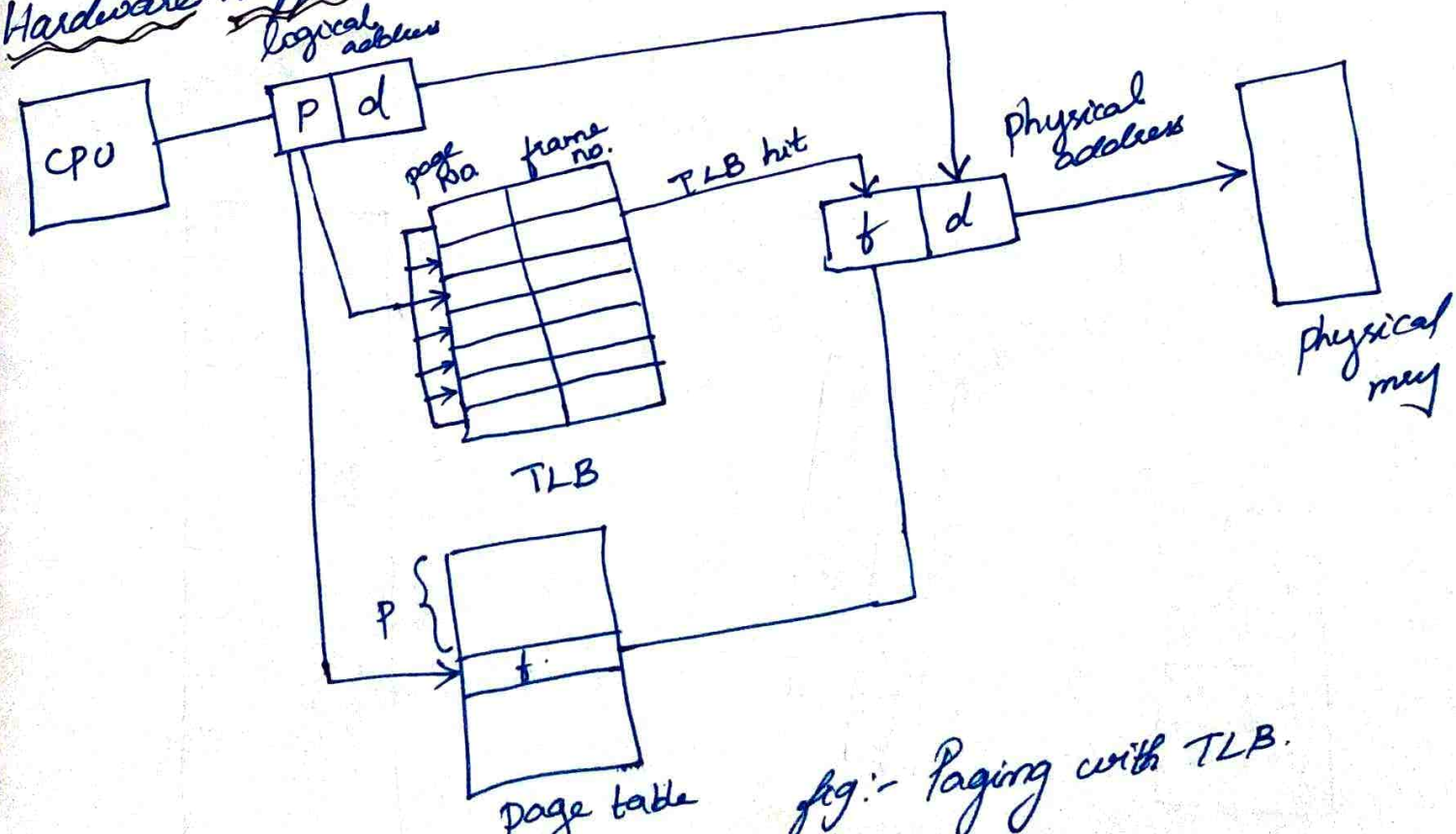
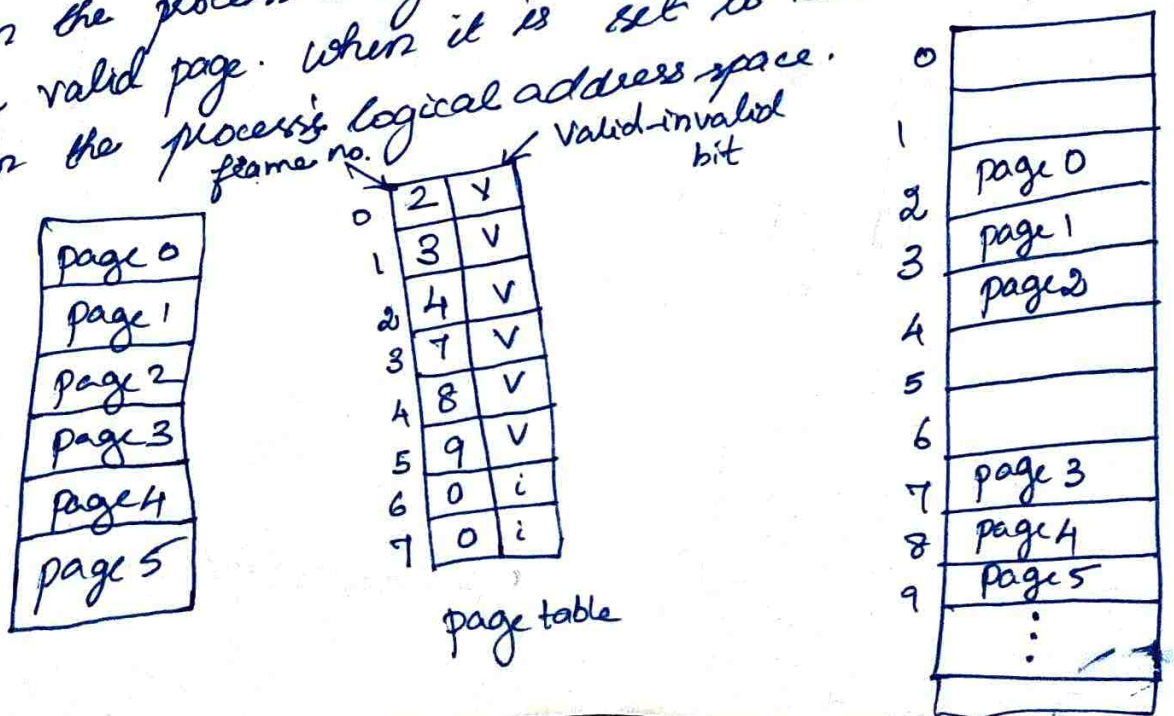


fig:- Paging with TLB.

A page table is implemented as a set of registers.
TLB - Translation look-aside Buffer. It consists of 2 parts.
Key and Value. The TLB contains only a few of the
page table entries. When logical address is generated
by the CPU, its page no. is presented to the TLB.
If the page no. is found, its frame no. is used to
access memory. If page no. is not found in the TLB
it is refered to as TLB miss, then a mey reference to
a page table must be made.

Protection :-

Mey protection is implemented by associating protection
bit with each frame. Valid, invalid bit attached to
each entry in the page table to ensure protection.
One bit can define a page ~~table to f~~ to be read-
write or read-only. The protection bits can be checked to
verify that no writes are being made to a read-only page
A separate protection bit can be provided for each kind
of access.
When the bit is set to valid, the associated page is
in the process's logical address space & is thus legal
or valid page. When it is set to invalid the page is not
in the process's logical address space.



Shared pages:

Reentrant code - it never changes during execution. Thus 2 or more processes can execute the same code at the same time.

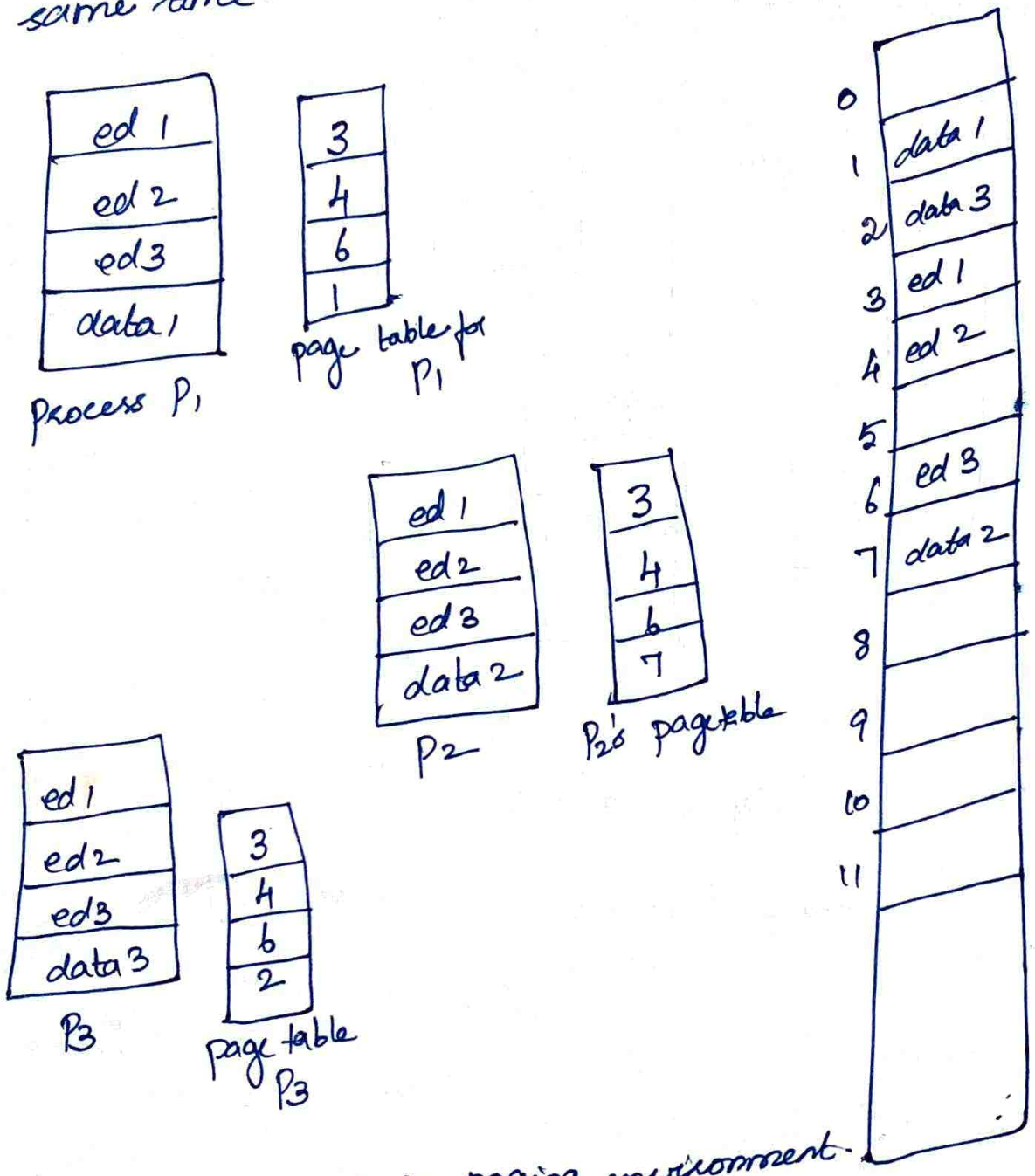


fig:- Sharing of code in paging environment.

5. SEGMENTATION

Segmentation is a mem management scheme that supports user view the main memory. A logical address space is a collection of segments.

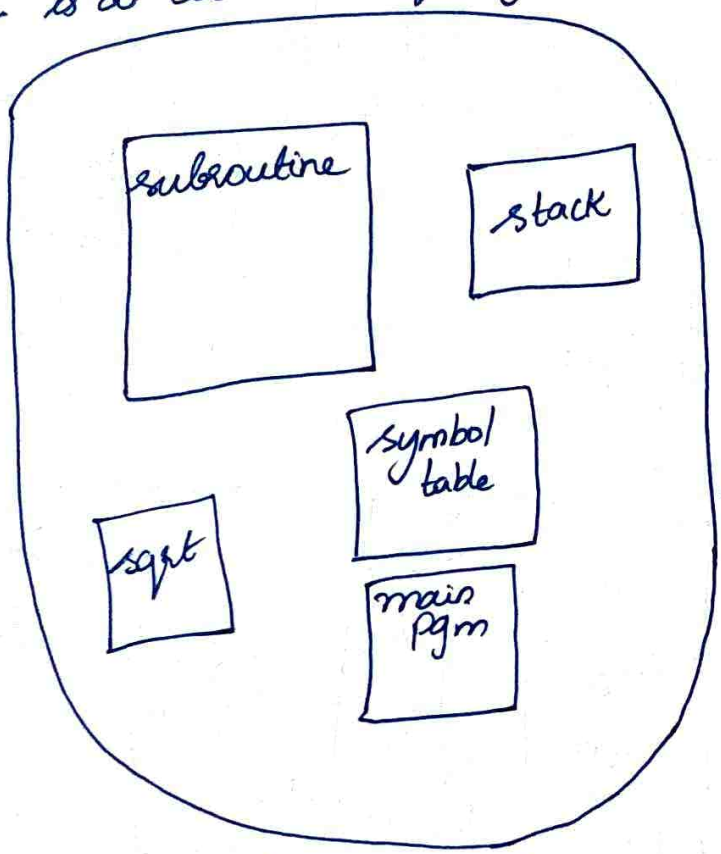


fig:- Programmer's view of a pgm. (user).

Each segment has a name & a length. Address specify both the segment name & the offset within the segment. So address is specified by segment name (or number) and an offset.

A logical address is, <segment no., offset>

When a program is compiled, it creates separate segments for

- code
- Global variables
- heap, from which mem is allocated
- Stack used by each thread
- Standard C library.

The loader ~~table etc~~ assigns a segment no. for all these segments.

Segmentation hardware :-

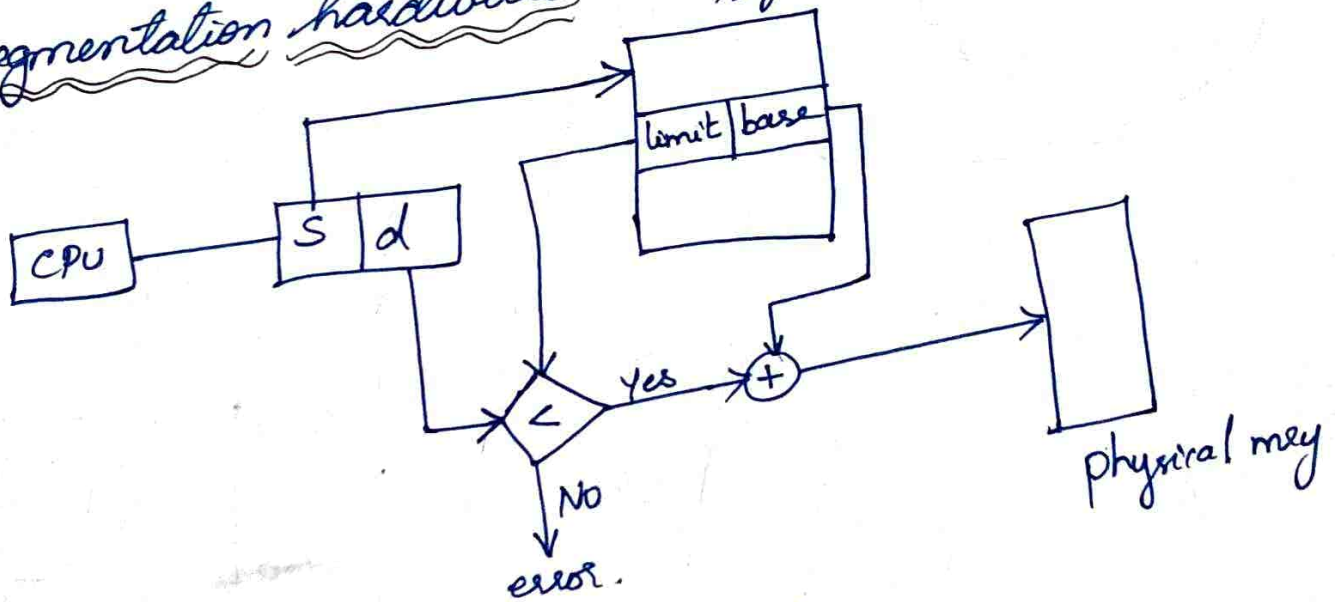


fig :- Segmentation hardware

- Each entry in a segment table has
segment base :- Contains starting physical address where the segment resides in the mem.
segment limit :- length of the segment.

The logical address consists of 2 parts segment no, s & offset d. s is used as an index to the segment table. d value must be b/n 0 & segment limit. If not then error

When an offset is legal, it is added to the segment base to produce the address in physical mey.

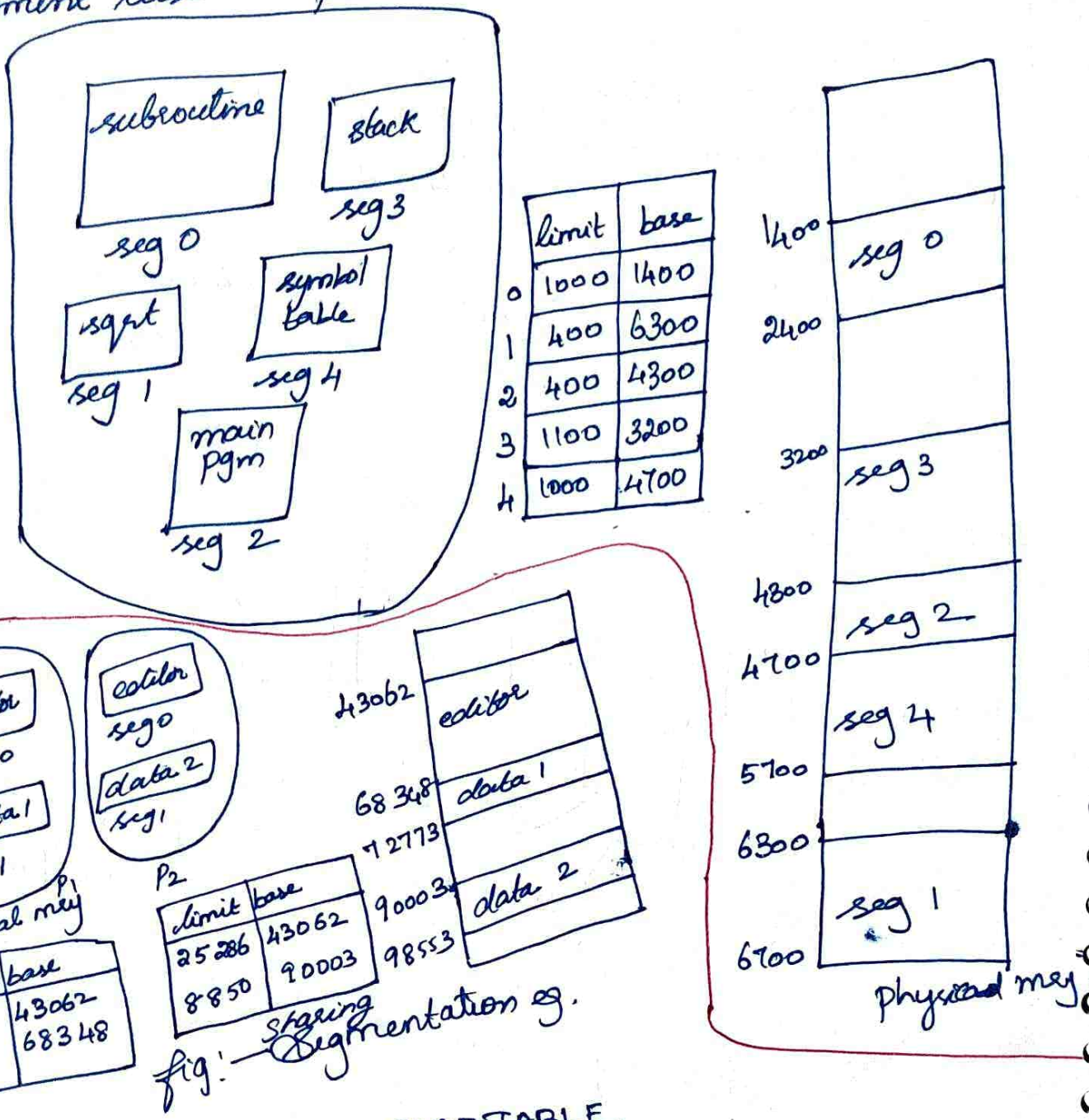


fig:- staging segmentation eg.

6. STRUCTURE OF THE PAGE TABLE

1) Hierarchical paging
2) Hashed page tables
3) Inverted page tables.

1) Hierarchical paging.

Computer systems support a large logical address space (2^{32} to 2^{64}). So page table becomes large. So page table can be divided into pieces.

Two level paging algm:

Page table is also paged.

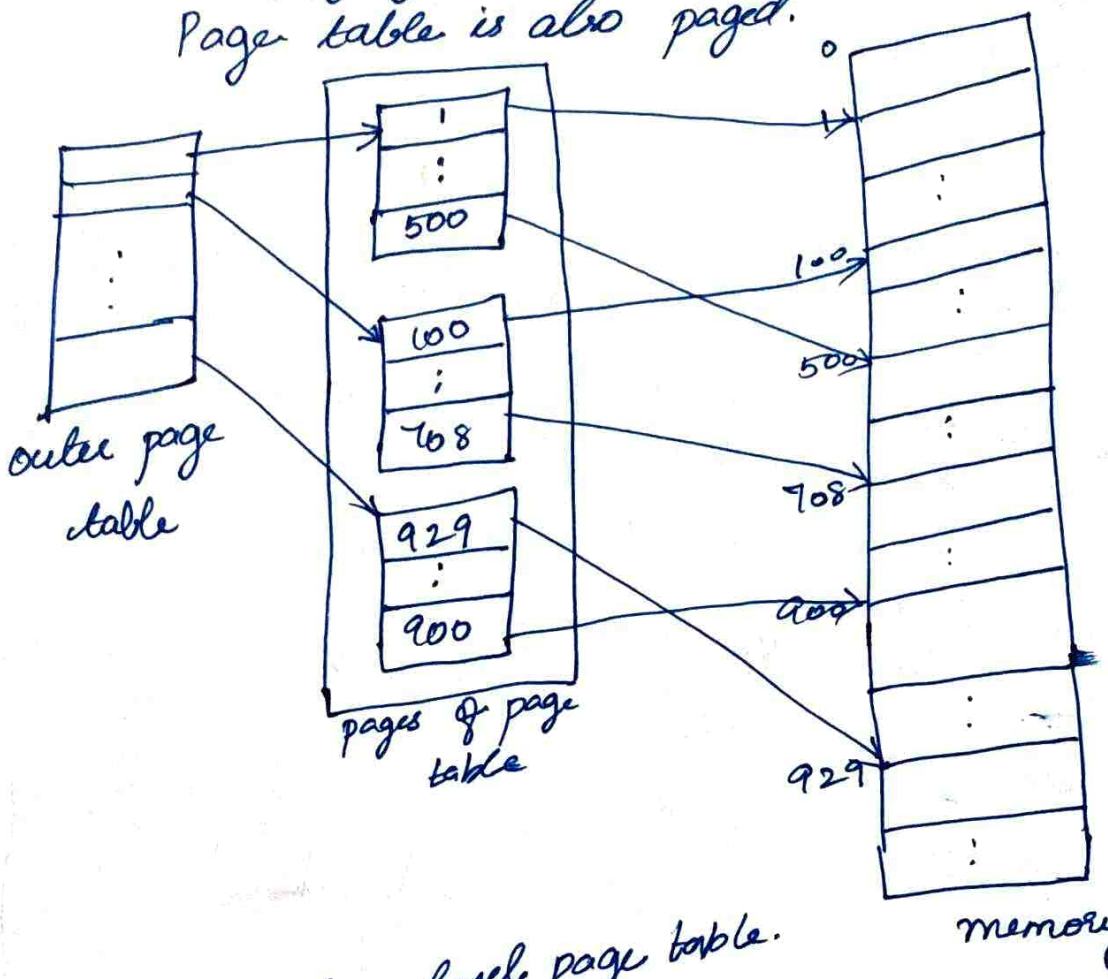


fig:- Two level page table.

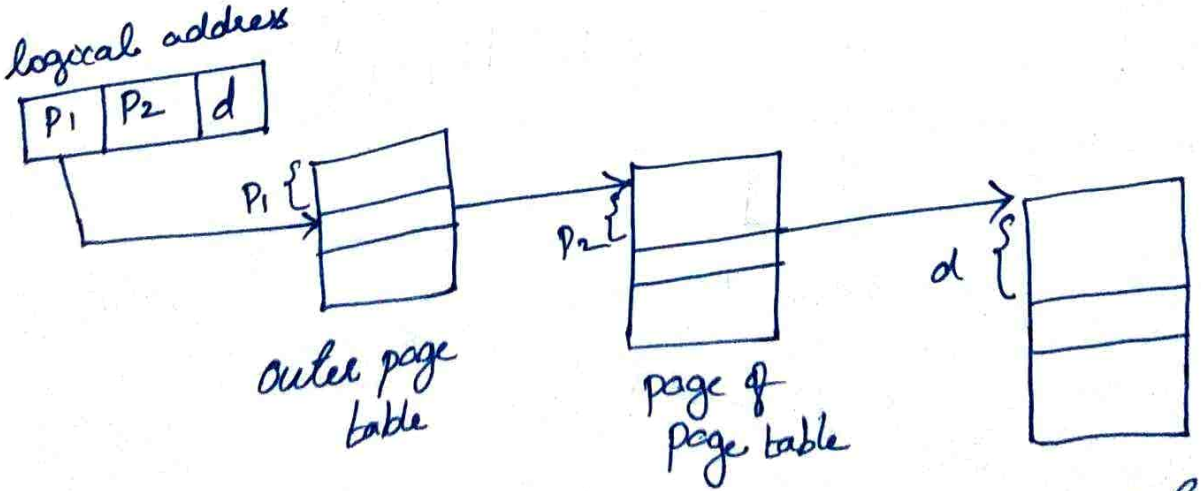


fig:- Address translation for a 2 level 32-bit paging architecture.

Page No.		page offset
p_1	p_2	d

p_1 – index into outer page table

p_2 – displacement within the page of the inner page table.

Address translation works from the outer page table inward, so this scheme is known as a forward-mapped page table.

2) Hashed page table:

A hash value is used as the virtual page no. Each entry in the hash table contains a linked list of elements that hash to the same location.

Each element consists of 3 fields

① Virtual page no.

② value of the mapped page frame

③ pointer to the next element in the linked list.

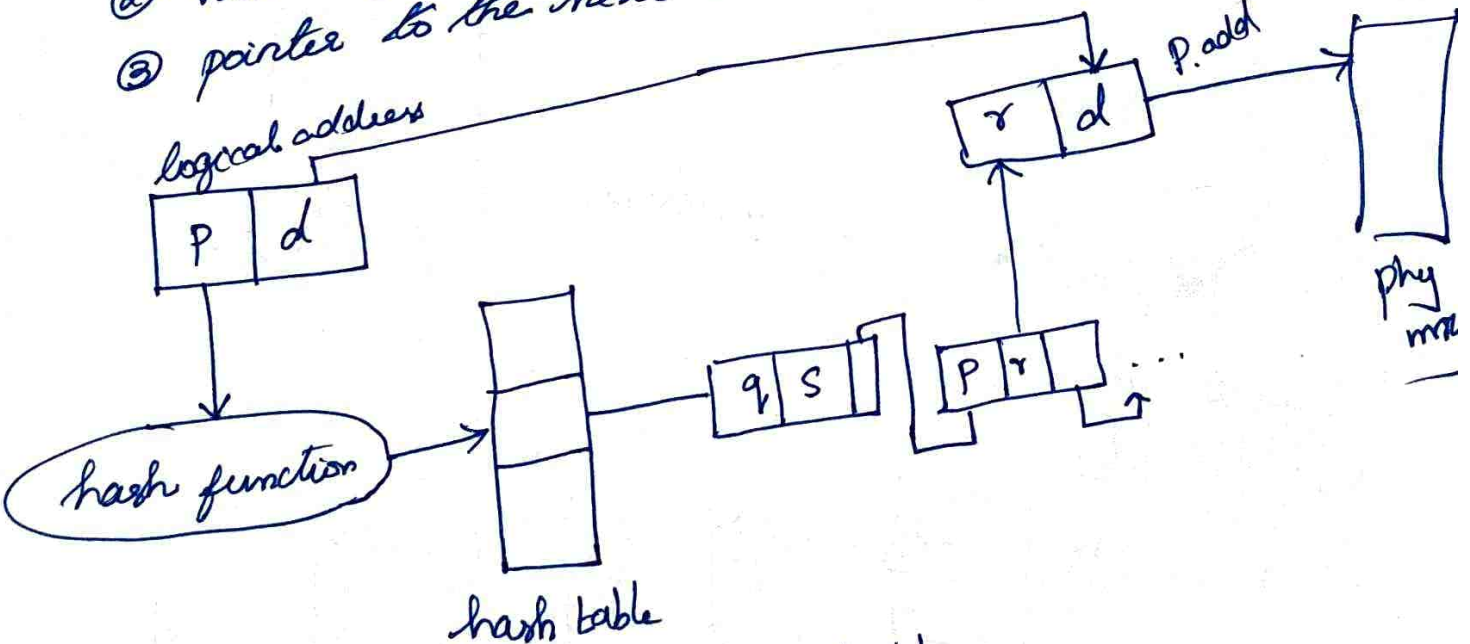


fig:- Hashed page table

① The virtual page no. in the virtual address is hashed into the hash table.
② The virtual page no. is compared with field 1 in the first element in the linked list.
- if matched, then the corresponding page frame is used.
- if not the element in the next linked list is searched.

3. Inverted page table :-

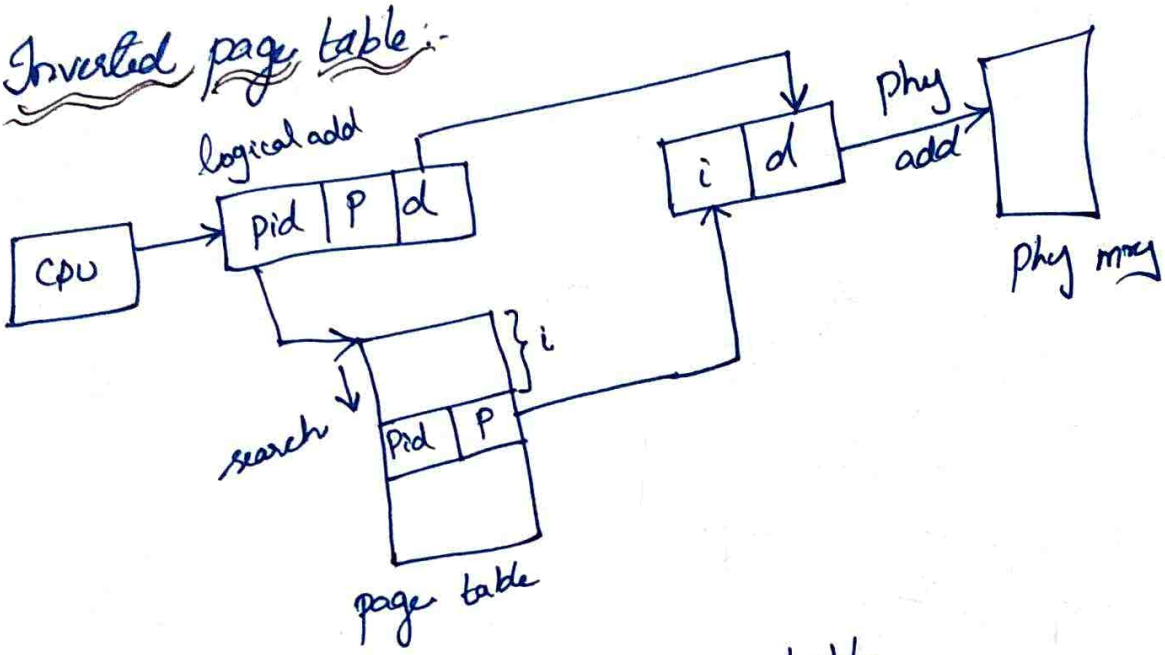


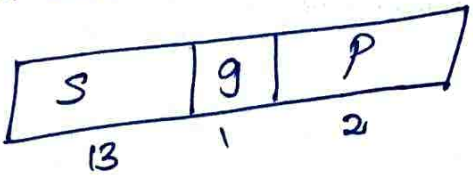
fig:- Inverted page table.

Each virtual address consists of
< process id, page-no., offset >

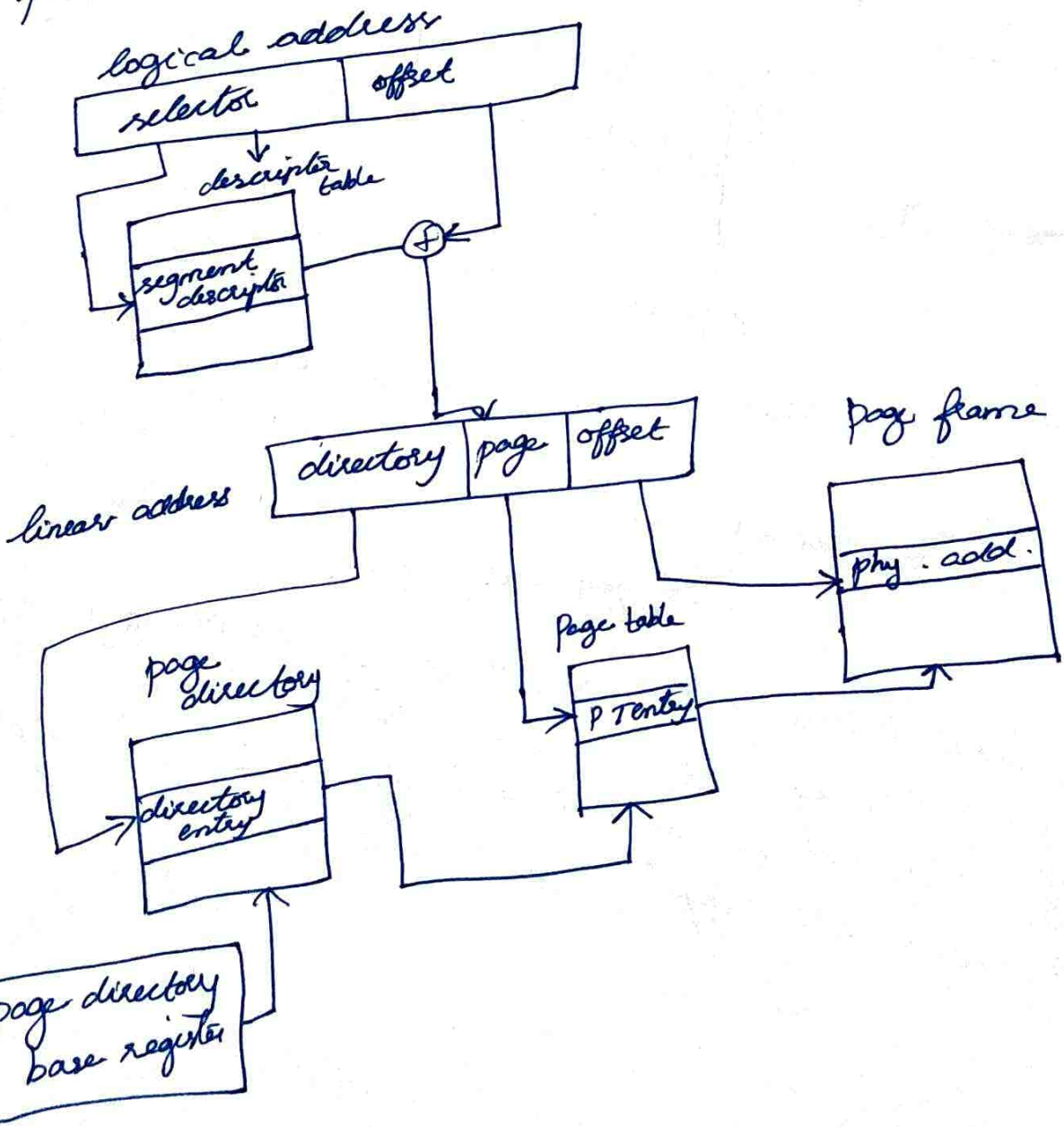
Inverted table entry is a pair of < process id, page no. >
< Pid, P > values are searched in the inverted page table.
If a match is found then < i, offset > is generated.
No match then illegal access.

7. SEGMENTATION WITH PAGING

The logical address space of a process is divided into 2 parts each of 8 KB. First part is private to that process & the second part is shared to all processes. Information of first part are stored in the local descriptor table (LDT) & of second part are stored in the global descriptor table (GDT).
The logical address is a pair of (selector, offset)



S - segment no.
g - whether the segment is in LDT or GDT.
P - protection.



9. VIRTUAL MEMORY

Virtual mry is a technique that allows the execution of process that may not be completely in mry. The program can be larger than physical mry is an advantage of virtual mry.

It maps mry addresses used by a program, called virtual addresses into physical addresses in computer mry. The process of translating virtual addresses into real address is called mapping.

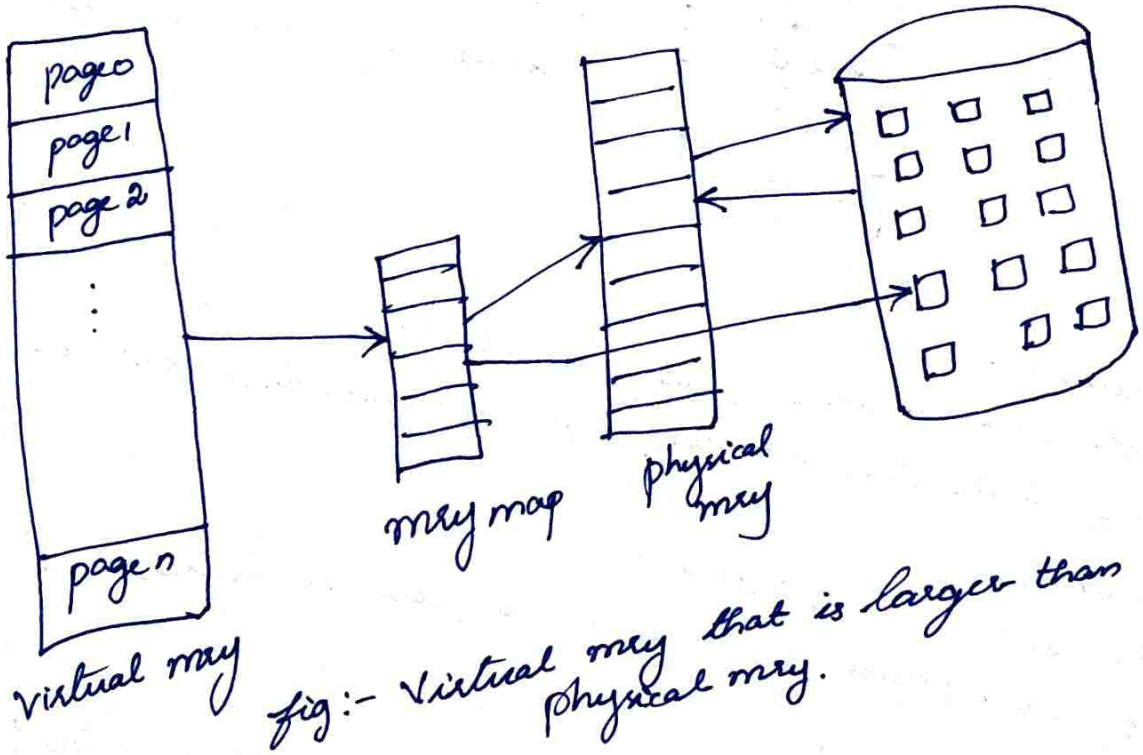


fig:- Virtual mry that is larger than physical mry.

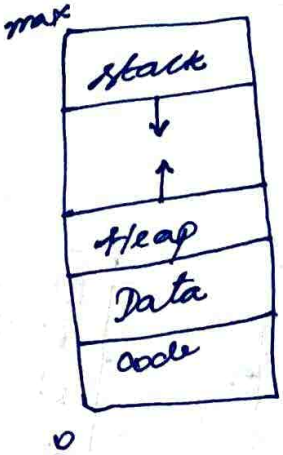
Implementation:-

1) Demand paging
2) Segmentation system.

Virtual address space (VAS) & mky space.

VAS is the set of virtual addresses. Mky space is the set of physical addresses in main mky directly addressable for processing.

- Heap grows upward in mky as it is used for dynamic mky al
- Stack grows downward in mky through successive fn ca
- Virtual address space is the blank space b/n the heap & the stack.



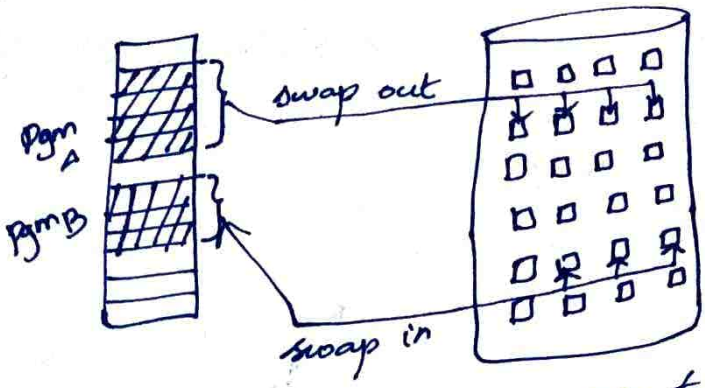
Holes can be
- filled as the stack or heap grows
- shared by 2 or more processes through page sharing.

Advantage
- Pgms can be larges than physical mky
- Useful in implementing multiprogramming environment.

Disadvantage.
- Not easy to implement
- decrease performance if it is used carelessly.

10. DEMAND PAGING.
- Is a paging sys with swapping. It initially loads pages only into mky as they are needed.

Lazy swapper :-
process of swapping a page into mky only when it is needed



Transfer of a paged mry to contiguous disk space.

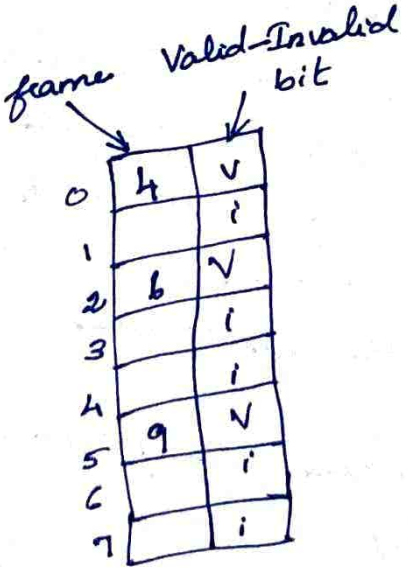
Benefits:
- Less I/O needed less swap time.
- Less mry needed
- Faster response
- More users.

Basic concepts :—
Valid-Invalid Bit scheme:—
. Is used to distinguish b/n pages that are in mry and the pages on the disk.

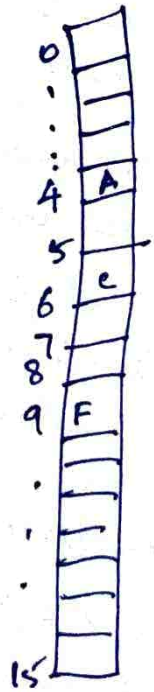
v - in mry
i - not in mry. or in the disk.

~~Valid bit~~ :—
Page fault:

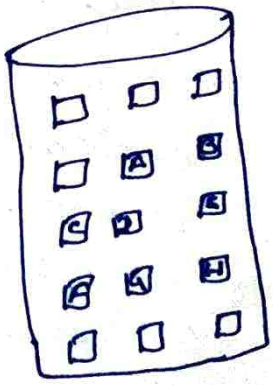
logical mry



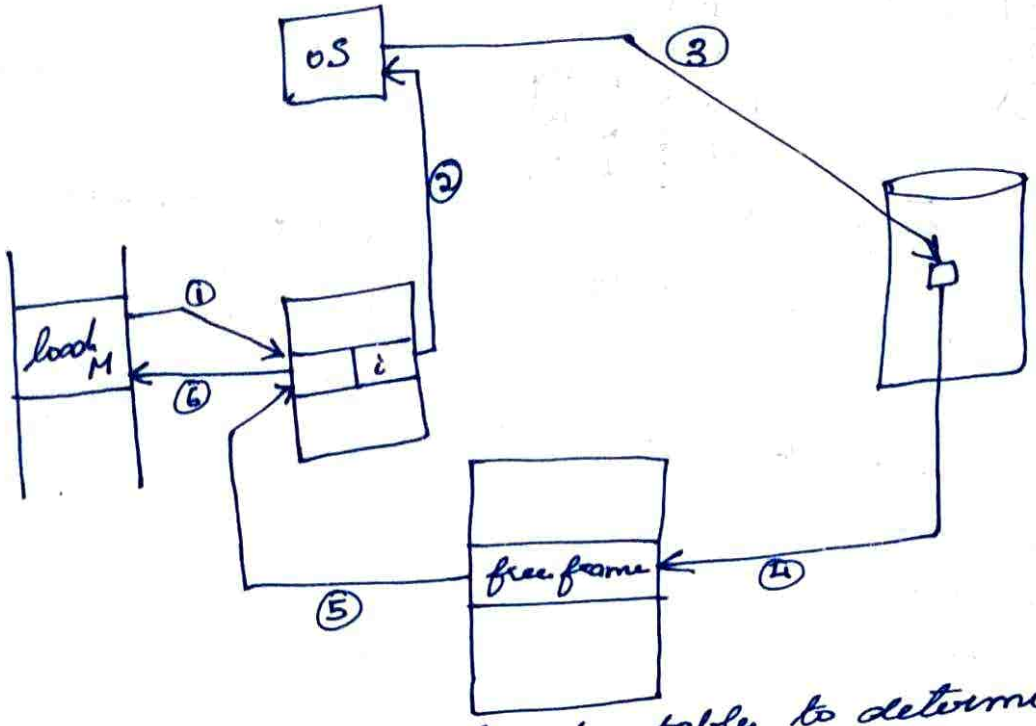
Page table.



Physical mry



Handling a page fault.



1. OS checks an internal table to determine whether th reference was a valid or an invalid my address.
2. Terminate the process if the reference was invalid
   . If it was valid, but we have not yet brought in that page, we now page it in.
3. Swap page into frame
   · Find a free frame from the free-frame list.
4. Schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process & the page table to indicate that the page is now in my.
   • Set validation bit = V.
6. Restart the instruction that was interrupted by the trap.
   • Process can now access the page as though it has always been in my.

Pure demand paging:-
- is a scheme that never brings a page into mry until it is required.

Requirements for demand paging:-
① Page table
② Secondary mry

Performance of demand paging:
- measured by computing the effective access time for a demand paged mry.
Let P be probability of a page fault rate $0 \le P \le 1$.
If P=0 no page faults
If P=1 every reference is a fault.

Effective Access Time (EAT)

EAT = (1-P) × mry access + P × page fault time.

Page fault time = page fault overhead +
[swap page out] +
swap page in +
restart overhead).

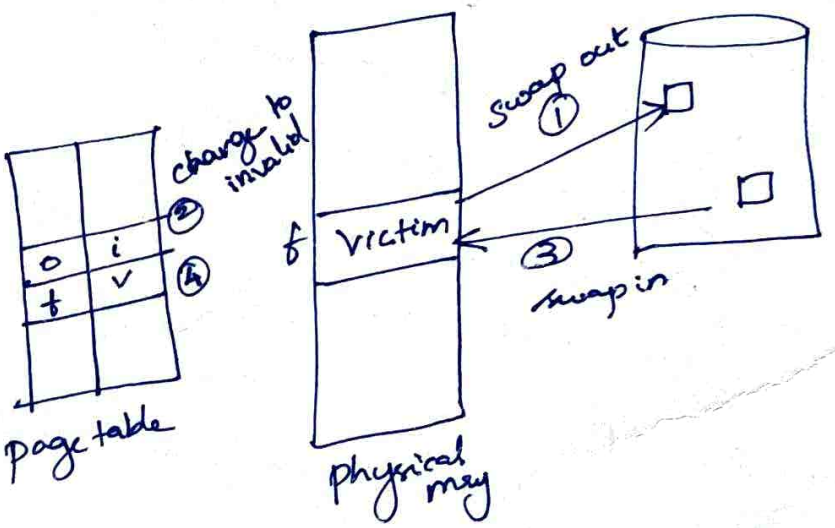
11. PAGE REPLACEMENT
- decides which mry pages to be paged out to disk to allocate mry for another page.

Types of page replacement algorithm.
1) FIFO Page Replacement
2) Optimal Page Replacement
3) LRU Page Replacement.

Basic steps.

When there is a page fault, the referenced page must be loaded. The steps are

1. Find the location of the desired page on disk.
2. Find the free frame in main mem.
   - If there is a free frame in mem use it
   - If no free frame, using page replacement algm, select an existing page for replacing with the new referenced page in mem.
   - The page being replaced is referred as the victim page.
3. Read the desired page into the newly free frame. Update the page & frame tables.
4. Restart the process.



Page Replacement.

FIFO page Replacement:—

- Create a FIFO queue
- The references are brought into the frames in FIFO manner.

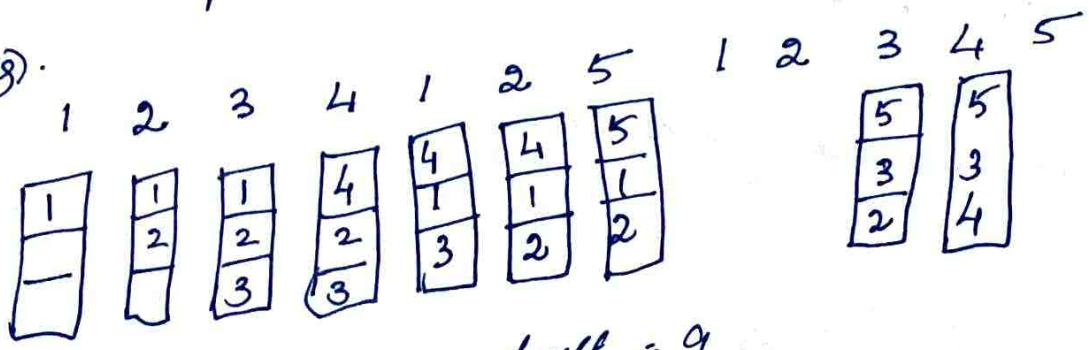
. First n references are inserted at the end of the queue.

. Next $n+1^{th}$ frames are checked with the existing frames in queue.

- If available, then no page fault
- If not available, then page fault occur. Replace the head of the queue.

. Repeat the above process until all references is allocated.

eg).

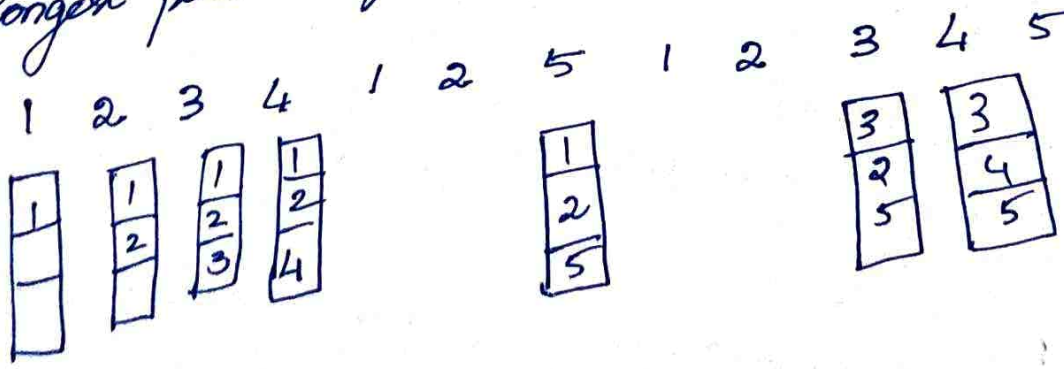


No. of page fault = 9

Optimal Page replacement:

. Algm that yields the lowest possible page fault.

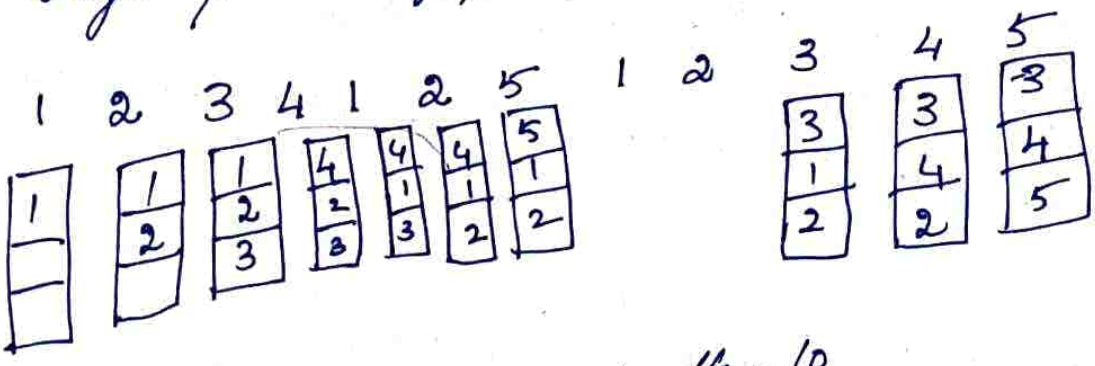
. Replace the page that will not be used for the longest period of time in the future.



Page fault = 7

LRU Page Replacement:

- Use recent past as an approximation of near future.
- Replace the page that has not been used for the longest period of time in past.



No. of page fault = 10

ALLOCATION OF FRAMES.

Allocation Algorithms.

1. Fixed allocation
2. Priority allocation

① Fixed allocation :—

The ratio of frames depends on the size of the process.

2 types.

1) Equal allocation
2) Proportional allocation.

Equal allocation :—

n process & m frames, give each process m/n frames.

Proportional allocation :—
Allocates frame according to the size of process.

Let S_i = Size of process P_i

$$S = \Sigma S_i$$

m = total no. of available frames.

a_i = allocation for P_i

$$= \frac{S_i}{S} \times m.$$

② Priority allocation:
Is the proportional allocation scheme that uses priorities of processes rather than the size of processes.
It gives high priority process more may than low priority process.

If P_i generates a page fault
. Select for replacement one of its frames from a process with lower priority no.

Global Vs Local Allocation.
Page replacement can be implemented broadly in 2 ways.
① Global replacement
② Local replacement.

Global replacement :—

Allows a process to select a replacement frame from the set of all frames, one process can take a frame from another process.

Local replacement :—

Each process selects from only its own set of allocated frames. The no. of frames allocated to a process does not change.

THRASHING

If a process does not have enough pages, the page fault rate is very high. This leads to high paging than executing activity called thrashing.

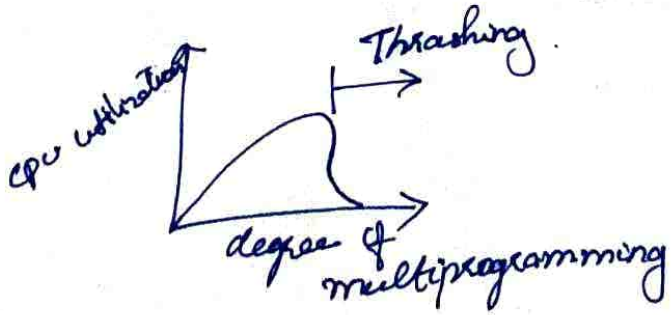
a. A process spends more time in paging than execution.

Cause of Thrashing:

. OS observes low CPU utilization & increases the degree of multiprogramming.

. Global page-replacement algm is used, it takes away frames belonging to other processes.

. Many processes join the waiting queue for the paging device, CPU utilization further decreases.



As degree of multiprogram is increased further, thrashing sets in & CPU utilization drops.

Thrashing effects can be limited using a local replacement algm. With local replacement, if one process starts thrashing, it cannot steal frames from another process & cause the latter to thrash also.

To prevent thrashing, processes must be provided with as many frames as it needed. Working set strategy is used to find how many frames are needed by it.

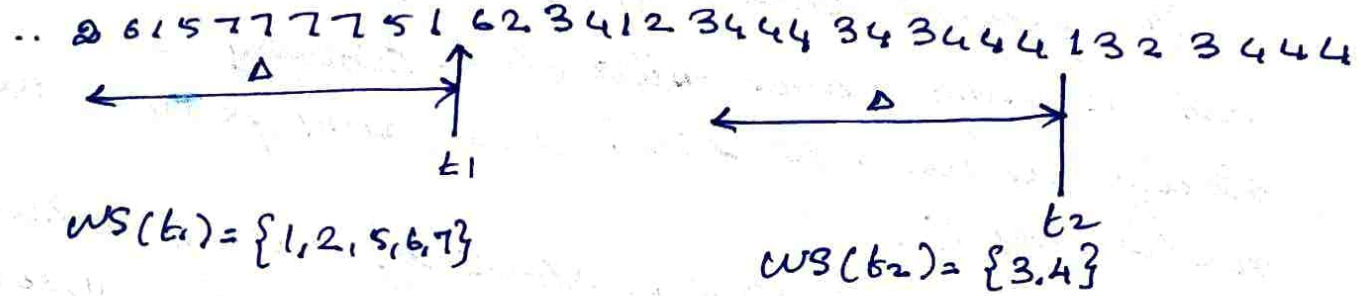
The working set strategy starts by looking at how many frames a process is actually using. This approach defines the locality model of process execution.
A locality is a set of pages that are actively used together. A pgm have several different localities, which may overlap.
@ when a subroutine is called, it defines a new locality. When it exits, the process leaves this locality.

working set model :—

Is based on the assumption of locality.
Parameter Δ, define the working-set window. The set of pages in the most recent Δ page references of process P_i constitutes the working set.
. If a page is in active use, it will be in working set.
. If a page is no longer used, it will drop from the working set Δ time units after its last reference.

Eg:
Given the following sequence of mry reference.

- If $\Delta = 10$ mry references, then the working set at time t_1 is $\{1,2,5,6,7\}$
- At time t_2 the working set has changed to $\{3,4\}$



Property of the working set :—

The important property of the working set is is siz

WSS_i (working set of process P_i) = total no of pages referenced in the most recent Δ (varies in ter

- If Δ is too small will not encompass entire localit
- If Δ too large will encompass several localities
- If $\Delta = \infty$ will encompass entire pgm.

$D = \Sigma WSS_i$ total demand frames

if $D > m \Rightarrow$ Thrashing.

Policy if $D > m$, then suspend one of the processes.

Keep track of the working set.

Page fault frequency :—

It establish acceptable page fault rate

- If actual rate too low, process loses frame
- If actual rate too high, process gains frame.

Page fault frequency (PFF) strategy:—

- Define an upper bound U & lower bound L for page fault rates.
- Allocate more frames to a process if fault rate is higher than U.
- Allocate less frames if fault rate is less than L.
- The resident set size should be close to the working set size W.
- We suspend the process if the PFF > U & no more free frames are available.

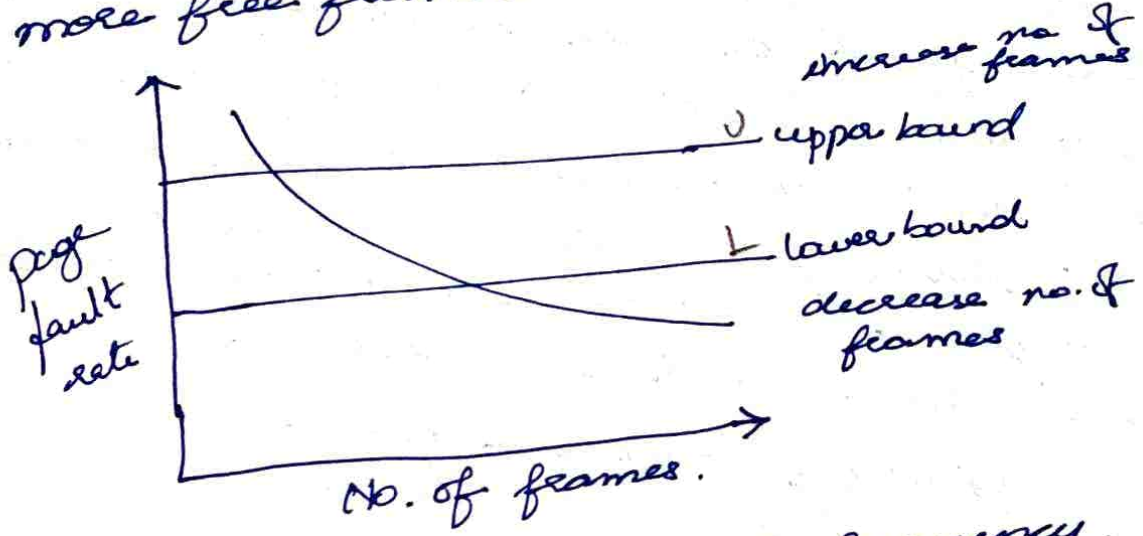


fig:— Page-fault frequency.

ALLOCATING KERNEL MEMORY

This is not simple as allocating mry in user space. It is allocated from a free mry pool.

Factors of complication are,

- Kernel is limited to about 1GB of virtual & physical mry.
- The Kernel's mry is not pagable.
- Kernel wants only physically contiguous mry.
- Kernel must allocate the mry without sleeping.

Strategies for allocating the kernel mry to the processes are,

① Buddy system

② Slab allocation.

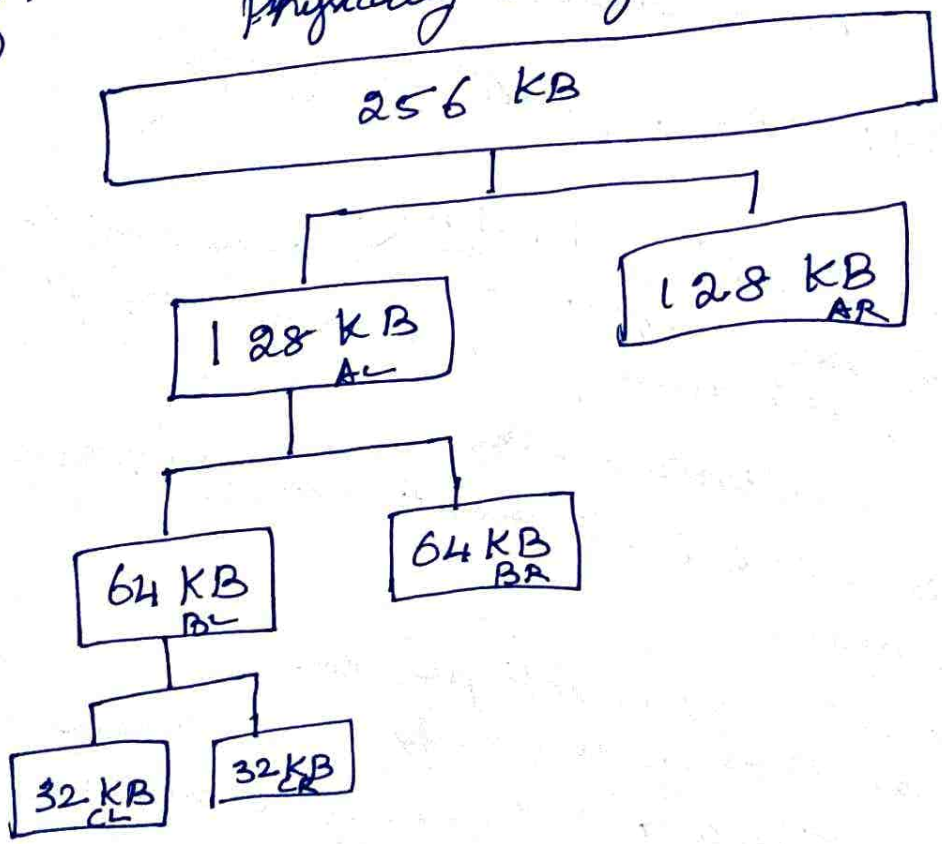
① Buddy system :—

Mry is broken down into power-of-two sized naturally aligned blocks. These blocks are known as buddies.

Mry allocated using power-of-2 allocator

- satisfies requests in units sized as power of 2.
- Request rounded up to next highest power of 2.
- When smaller allocation is need than is available current chunk split into two buddies of next-lower power of 2. & Continues until appropriate sized chunk is available.

② Physically contiguous space.



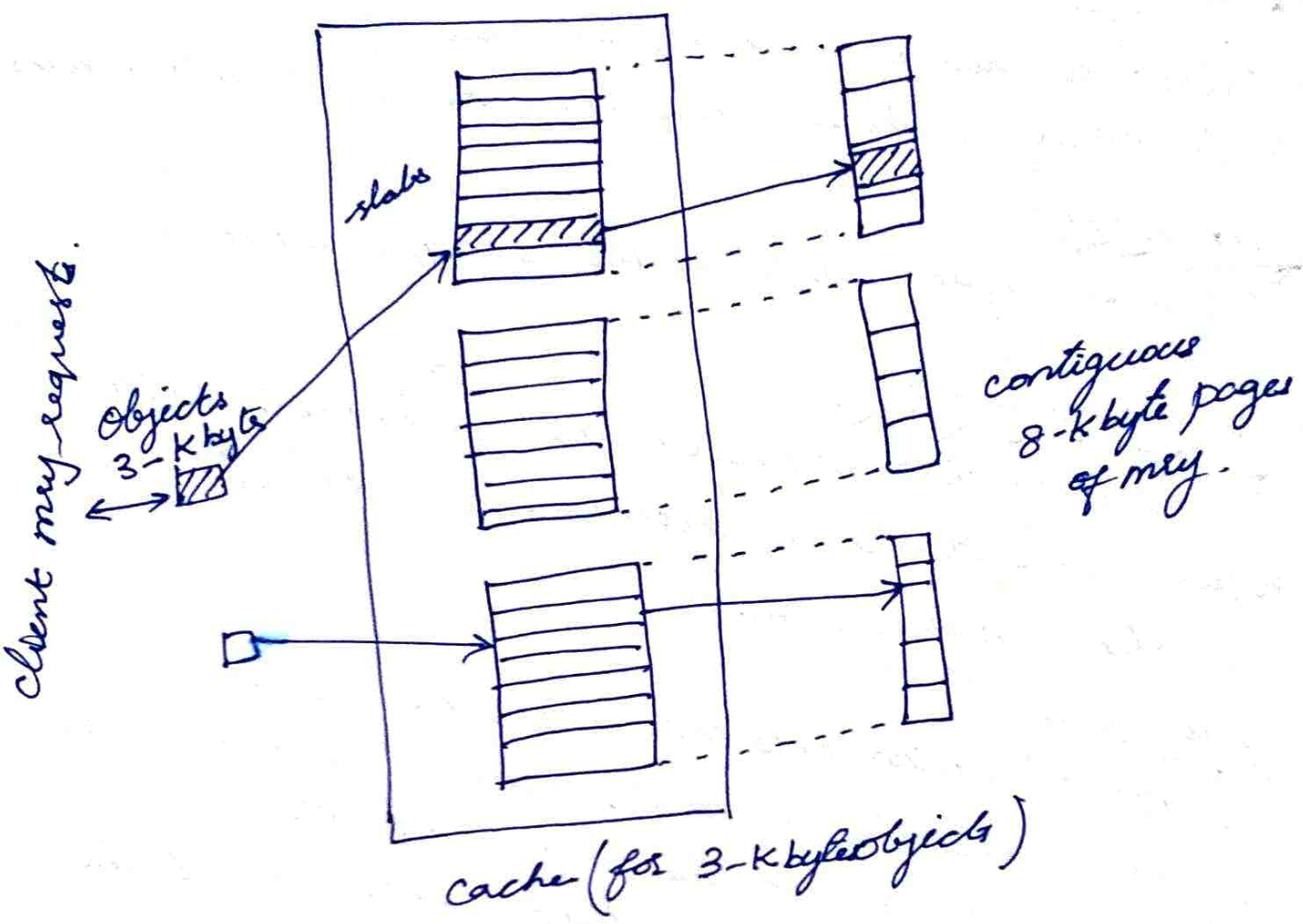
Advantage:-
- Fast to allocate & deallocate mry.
- Quickly combine unused chunks into larger chunk.

Disadvantage:-
- wastes space in internal fragmentation.

② Slab allocation:
Initialization & destruction of Kernel data objects can actually outweigh the cost of allocating mry for them.

Layout:-
Slab allocation has the structure as below,
- Object - describe a single mry allocation unit
- Cache:- manager for the pool of objects.
- Slab - group of objects that reside within the cache.
  - slab - one or more physically contiguous pages
  - slab the amount that a cache can grow or shrink by.
  - slabs may exist in one of the following states.
    - empty - all objects on a slab marked as free
    - partial - slab consists of both used & free objects.
    - full - all objects on a slab marked as used.

Benefits:-
- No fragmentation
- Fast mry request satisfaction.



OS EXAMPLES.

i) Windows XP

ii) Solaris.

UNIT-IV
FILE SYSTEMS & I/O SYSTEMS

1. MASS STORAGE SYSTEM OVERVIEW

Mass storage refers to the storage device used in computer to store large amount of data in a persisting & machine-readable fashion. Types & storage varies based on the type of data & the rate at which it is created & used.

eg) hard disk, CD-ROM, Magnetic tape & removable disk.

Mass storage devices are characterized by.

- **Seek time**
  Time to move heads from one track to another
- **Rotation delay**
  Half the time required for the disk to make a complete rotation.
- **Access time**
  Seek time + rotation delay.
- **Transfer rate**
  Rat at which data can be transferred to or from the disk.

Magnetic disks:-

Refer unit 1.

Bus:- A disk drive is attached to a computer by a set of wires called an I/O bus.

eg) Advanced technology attachment (ATA)
Serial ATA (SATA)
Universal serial bus (USB)
Fibre channel (FC).

Controllers :-
The data transfers on a bus are carried out by special electronic processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is the controller which is built into each disk drive.

Magnetic Tapes :-
Refer Unit 1.

Solid state disks (SSD):
SSD is nonvolatile memory that is used like a hard drive. SSDs have the same characteristics as traditional hard disks but can be more reliable bcz, they have no moving parts & faster bcz they have no seek time or latency. Consumes low power & have less capacity than the larger hard disks, & may have shorter life spans than hard disks. One use for SSDs is in storage arrays, where they hold file-system metadata that require high performance.

2. DISK STRUCTURE

Disk provide the bulk of secondary storage for modern computer systems. Magnetic tape was used earlier, but the access time was too large. So tapes are used for backup.

Modern disk drives are addressed as large one-dimensional arrays of logical blocks i.e, smallest unit of transfer. The size of a logical block is 512 bytes.

The 1-D array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, & then through the rest of the cylinder from outermost to innermost.

Most disks pack many more sectors into outer cylinders than inner ones, using one of the 2 approaches.

- Constant linear velocity (CLV), the density of bits is uniform from cylinders to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This approach is used by CDs & DVDs.
- Constant angular velocity (CAV) the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. These disks could have a constant no. of sectors per track on all cylinders.

3. DISK SCHEDULING

Disk scheduling algm is used to schedule the servicing of disk I/O requests.

Types. (5) ① FCFS ② SSTF ③ SCAN ④ C-SCAN ⑤ Look scheduling.

Consider,

Queue = 98, 183, 37, 122, 14, 124, 65, 67

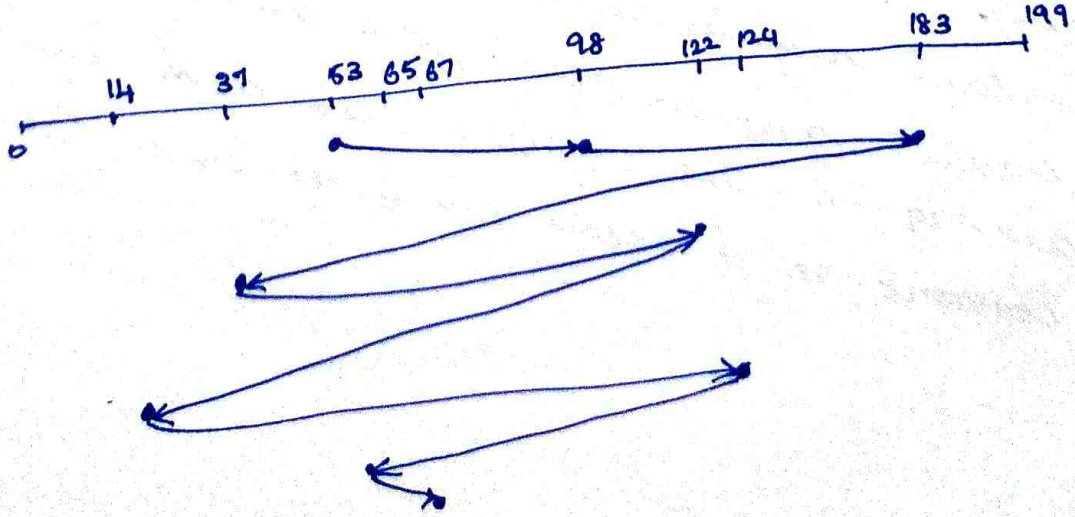
Head pointer = 53.

FCFS

- simplest form of disk scheduling. It serves the first request in the queue.

Current	Next	Total
53	98	45
98	183	85
183	37	146
37	122	85
122	14	108
14	124	110
124	65	59
65	67	2
	Total	640

Total head movement = 640 cylinders.

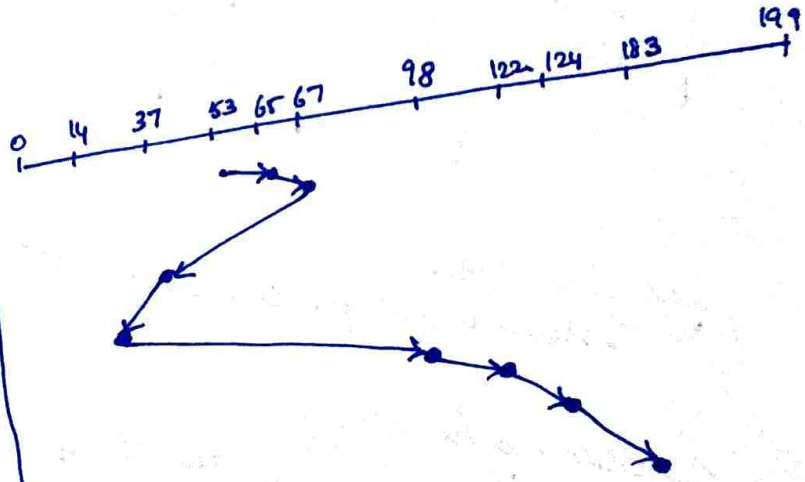


Advantage:—
- Simple to write & understand
- Perform operations in order requested
- No reordering of work queue
- No starvation: every request is serviced.

Disadvantage:—
- Large seek time as the disk head may have to jump from one extreme to anotha.
- Does not provide the fastest service.

SSTF
Shortest seek time first algorithm selects the request with the minimum seek time from the current head position.

current	Next closest	Total
53	65	12
65	67	2
67	37	30
37	14	23
14	98	84
98	122	24
122	124	2
124	183	59
	Total	236



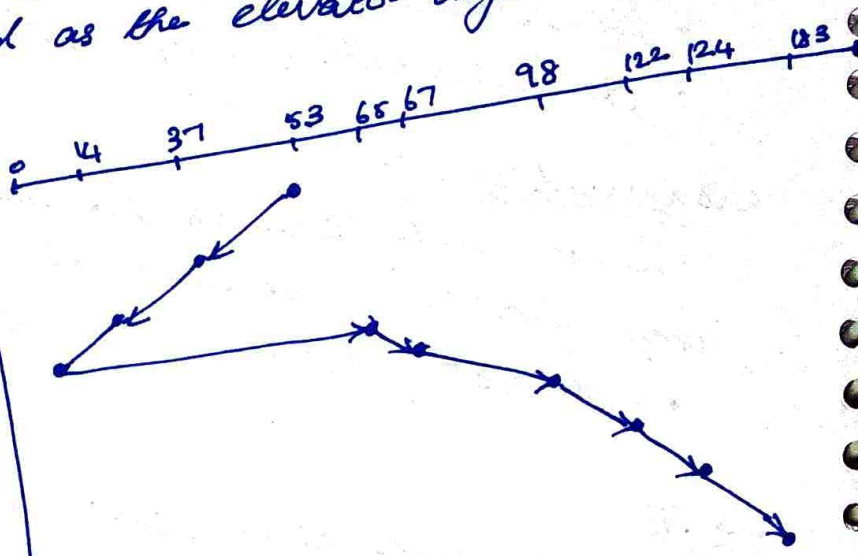
Advantage
- Minimize seek time
- Maximizes throughput

Disadvantage
- Starvation is possible
- More CPU time required to find out the shortest seek time.
- Not always optimal.

SCAN scheduling:

In SCAN algm, the disk arm starts at one end of the disk and moves toward the other end. (end of the disk) The head continuously scans back & forth across the disk. So it is called as the elevator algm.

Current	Next	Total
53	37	16
37	14	23
14	65	51
65	67	2
67	98	31
98	122	24
122	124	2
124	183	59
	Total	208



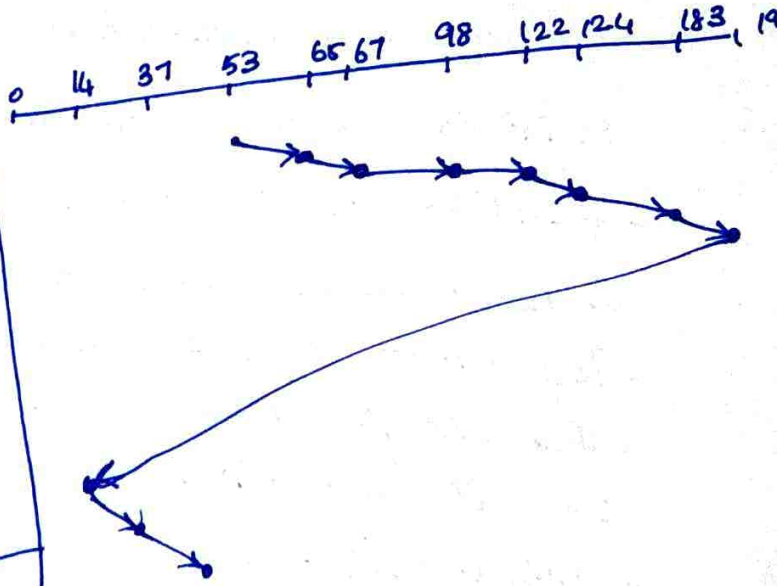
Disadvantage:
- More wait for the requests at the end.

C-SCAN scheduling:

Circular SCAN scheduling is a SCAN variant which provides a more uniform wait time than SCAN.

The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Current	Next	Total
53	65	12
65	67	2
67	98	31
98	122	24
122	124	2
124	183	59
183	199	16
199	0	199
0	14	14
14	37	23
	Total	382



LOOK/C-LOOK scheduling:

LOOK/C-LOOK is a version of SCAN/C-SCAN in which the arm goes only as far as the final request in each direction. Then it reverses the direction immediately, without going all the way to the end of the disk because they look for a request before continuing to move in a given direction.

current	Next	Total
53	65	12
65	67	2
67	98	31
98	122	24
122	124	2
124	183	59
183	14	169
14	37	23
	Total	322

Selection of a disk-scheduling algm.

- SSTF is common & has a natural appeal.
- SCAN & C-SCAN perform better for sys that place a heavy load on the disk.
- Performance depends on the no. & types of requests.
- Requests for disk service can be influenced by the file allocation method.
- Either SSTF or LOOK is a reasonable choice for the default algm. Priority scheduling if some requests are more important.

4. DISK MANAGEMENT

The OS is responsible for the following disk mgnt.

- disk initialization
- booting from disk
- Bad block recovery.

Disk formatting:-

Low level formatting
. Also called as physical formatting
. Divide the disk into sectors that the controller can read & write.
. Fills the disk with a special DS for each sector
. Header & Trailer contains information used by disk controller, such as a sector no. & an error-correcting code (ECC)
. When the controller writes a sector of data, ECC is updated with a value calculated from all the bytes in the data area.
. When the sector is read, ECC is recalculated & is compared with the stored value → verify the data is correct.

Disk partition:

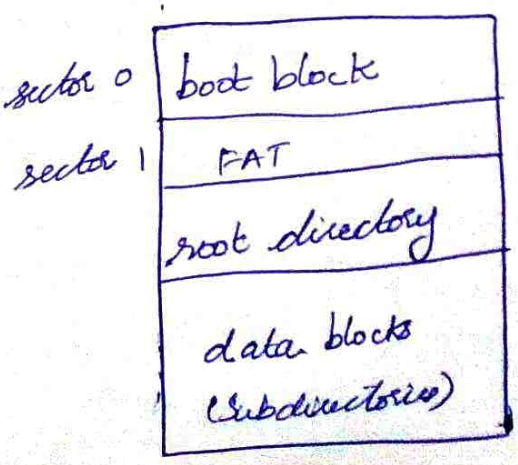
- To use a disk to hold files, OS still needs to record its own DS on the disk.
- Partition the disk into one or more groups of cylinders.
  - Each partition can be treated as a separate disk
- Logical formatting or making a file s/m.
  - Store the initial file-system DS onto the disk
- Maps of free & allocated space (FAT or inode)
- Initial empty directory
  - To increase efficiency most file s/m group blocks into clusters.
- Disk I/O done in blocks
- File I/O done in clusters.

Boot Block:—

- Boot block is an initial program to run when the system is powered up or rebooted.
- Boot strap is stored in Read only memory (ROM).
- The full bootstrap program is stored in boot blocks at a fixed location on the disk.

Boot Disk

A disk that has a boot partition is called boot disk or system disk. The boot ROM instructs the disk controller to read the boot blocks into memory, and then start executing the code to load the entire OS.



MS DOS disk layout

Bad blocks

Disks are moving parts and small tolerances they are prone to failures. Most disks come from the factory with bad blocks.

IDE controllers

The bad blocks are handled manually in some disks with IDE controllers.

- MS-DOS format : Write a special value into the corresponding FAT entry for bad blocks.
- MS-DOS chkdsk : search and lock bad blocks.

SCSI disk

The controller maintains a list of bad blocks on the disk. The list is initialized during the low-level format at the factory and is updated over the life of the disk.

- Low-level formatting → spare sectors (OS don't know)
- Sector sparing
  - Controller replaces each bad sector logically with one of the spare sectors.
    - Invalidate optimization by OS's disk scheduling.
    - Each cylinder has a few spare sectors.
- Bad-sector transaction
  - OS tries to read logical block 87
  - Controller calculates Ecc and finds that is bad. Report to OS

. Reboot next time, a special command is run to tell the controller to replace the bad sector with a spare.
. Whenever the system requests block 87, it is translated into the replacement sector's address by the controller.

• Secta slipping.
. As an alternative approach to sector sparing, some controllers can be instructed to replace a bad block by sector slipping.

5. SWAP-SPACE MANAGEMENT:

Modern systems typically swap out pages as needed, rather than swapping out entire processes. Hence the swapping system is part of the virtual mgmt sym.

Swap space Use:-

. The amount of swap space needed by an OS varies greatly according to how it is used.
. Some systems support multiple swap spaces on separate disks in order to speed up the virtual mgy system.

Swap space location:

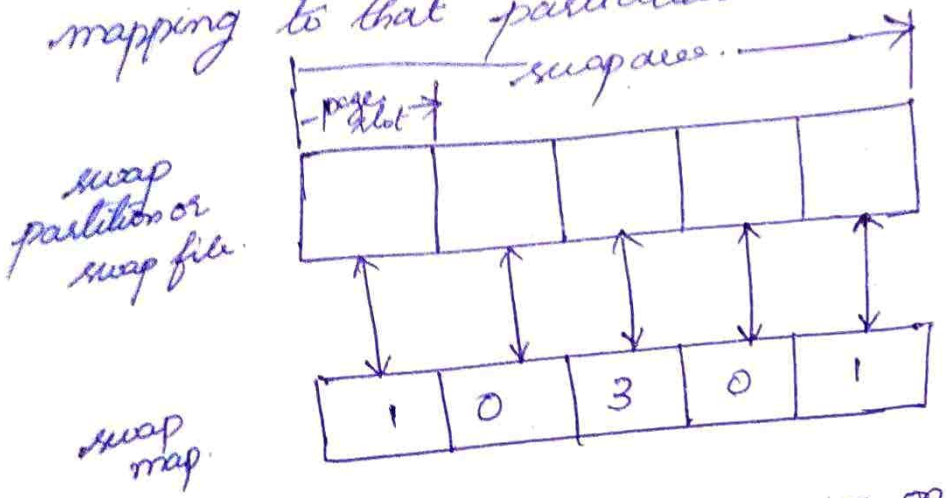
Swap space can be physically located in one of the two locations:

• As a large file which is part of the regular filesystem. This is easy to implement but inefficient.

- As a raw partition, on a separate or little-used disk

Example:-

In Linux system, a map of swap space is kept in memory, where each entry corresponds to a 4K block in the swap space. Zeros indicate free slots and non-zero refers to how many processes have a mapping to that particular block.



Data Structures for swapping on Linux systems.

5. FILE SYSTEM

- File is a named collection of related information recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

- File system is the mechanism used to control how data and program is stored and retrieved.

Parts of file system:

- A collection of files, each storing related data
- A directory structure - organizes and provides information about all the files in the system.
- Partitions - separate physically or logically large collections of directories.

File types:

1) Data files
   - numeric, alphabetic or binary
   - text files
2) Text files
   - Sequence of characters organized into lines
3) Source file
   - Subroutines & functions
4) Object files
   - Understandable by system's linker.
5) Executable file
   - No execution

Implementation:

Name of the file is splited into two parts

- Name
- an extension.

eg) resume - doc.

Common file types:

Different types of file are executable, object, source code, batch, text, word processor, library, print or view, archive, multimedia.

File attributes

Name, Identifier, Type, location, size, production, Time, data, user identification.

File operations

- Create
- Write
- Read
- Reposition
- Delete
- Truncate

Types of Lock:

1. Shared lock

Read - several process can acquire the lock concurrently

② Exclusive lock.

Write lock - only one process can acquire such a lock at a time.

File access mechanism:

① Sequential access :- Records are accessed in some sequence.

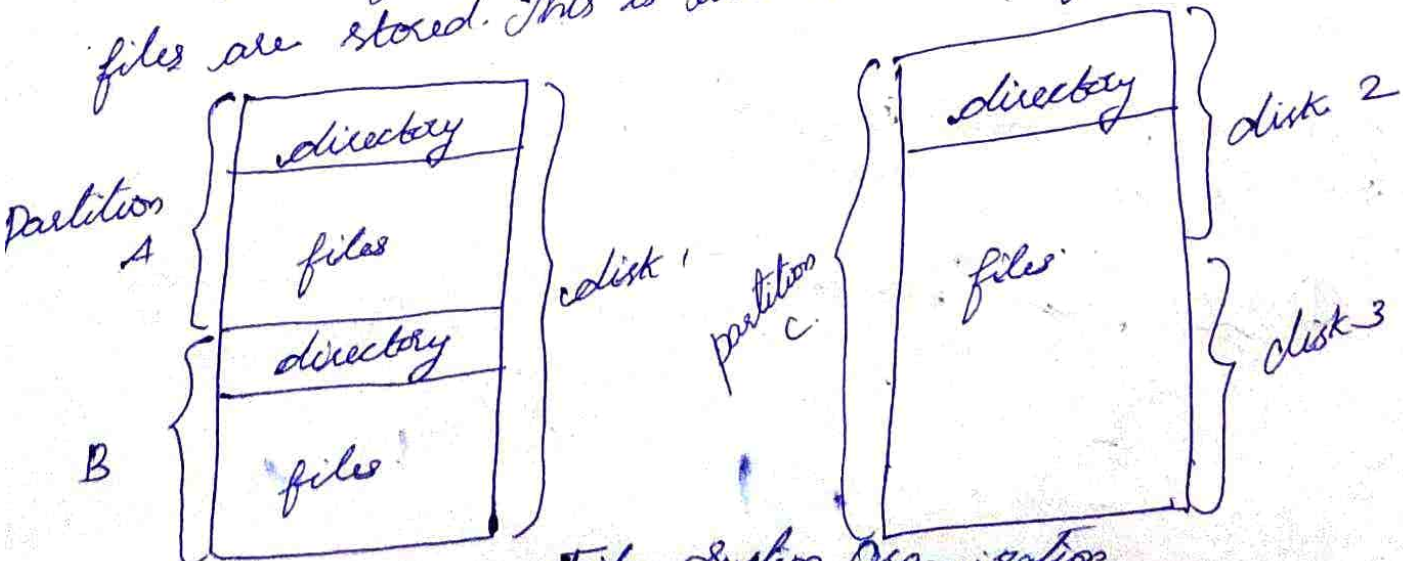
② Direct / Random access :- Records are accessed directly. Records are accessed by their own address. They need not be in any sequence.

③ Indexed sequential access :-

An index is created for each file which contains pointers to various blocks. Index is searched and its pointer is used to access the file directly.

6. DIRECTORY & DISK STRUCTURE:

Directory is a file system structure in which files are stored. This is also known as foldees.



File System Organization

Directory structure:

- A directory structure is the way the file s/m & its files are organized into a hierarchy of folders
- File s/m is organized in 2 parts.
  - disks are split into one or more partition
  - Each partition contain information like name, location size & type.

Operations performed on directory:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file s/m

Characteristics:-

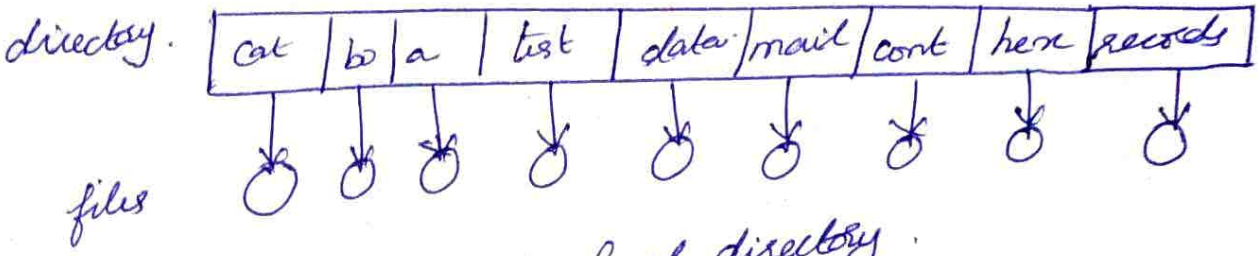
- Efficiency - locating a file quickly
- Naming:- convenient to users.
  - 2 users can have same name for different files.
  - Same file can have several different names.
- Grouping - logical grouping of files by properties.

Logical structure schemes.

Different schemes are (5 types).

① single level directory:-

All files are contained in the same directory.
- A single root directory for all users
- There are no sub directories.

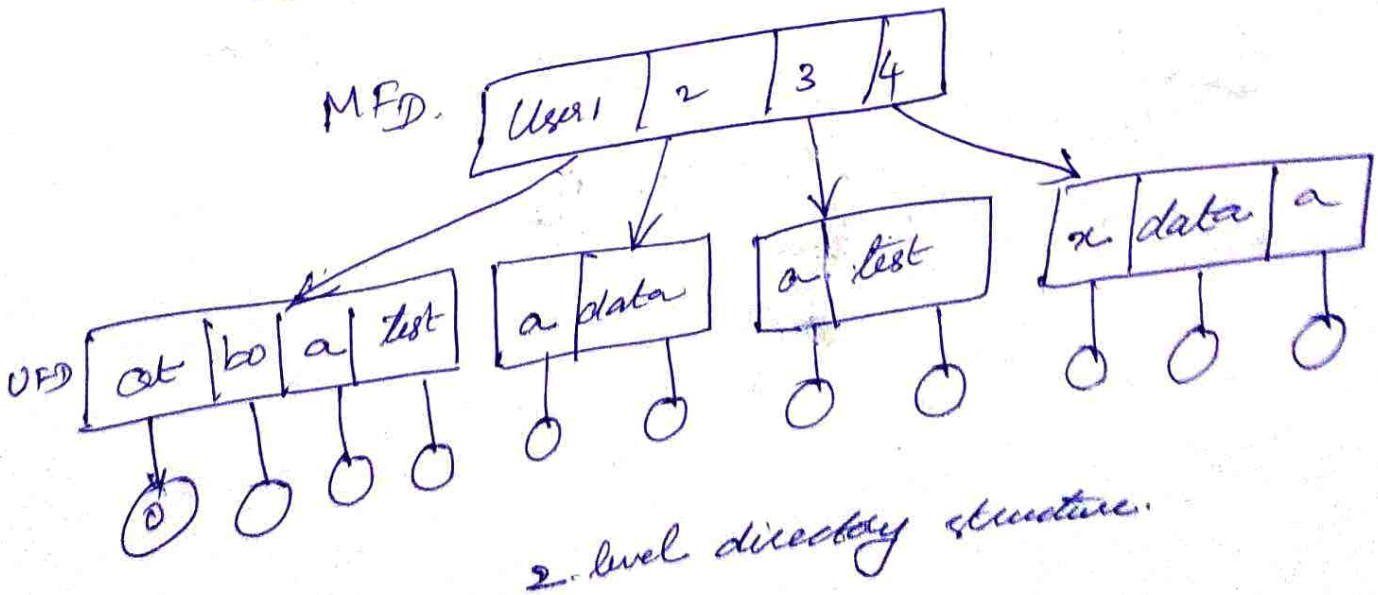


single-level directory.

Advantages:
- Easy to find a file in a directory.

Disadvantage:
- Naming and grouping problem.

② Two-level directory:
- One master file directory (MFD)
  - Each entry in MFD points to a user file Directory (UFD)
  - MFD is indexed by user name & account no.
- Each user has their own UFD.
  - UFD has similar structure
  - UFD lists files of a single user.



2-level directory structure.

Tree ~~structured directories~~ organization :-

A 2 level directory can be thought of as a tree or an inverted tree of height 2.

- Root is MFD
- Its direct descendants are the UFDs.
- The descendants of the UFDs are the file itself.
- Leaves are files.

File Operation :-

- File creation.
- File deletion

Issues :-

- Sharing - accessing other users files
- System files
- Grouping pbm.

Adv :-

Isolation when the users are completely independent.

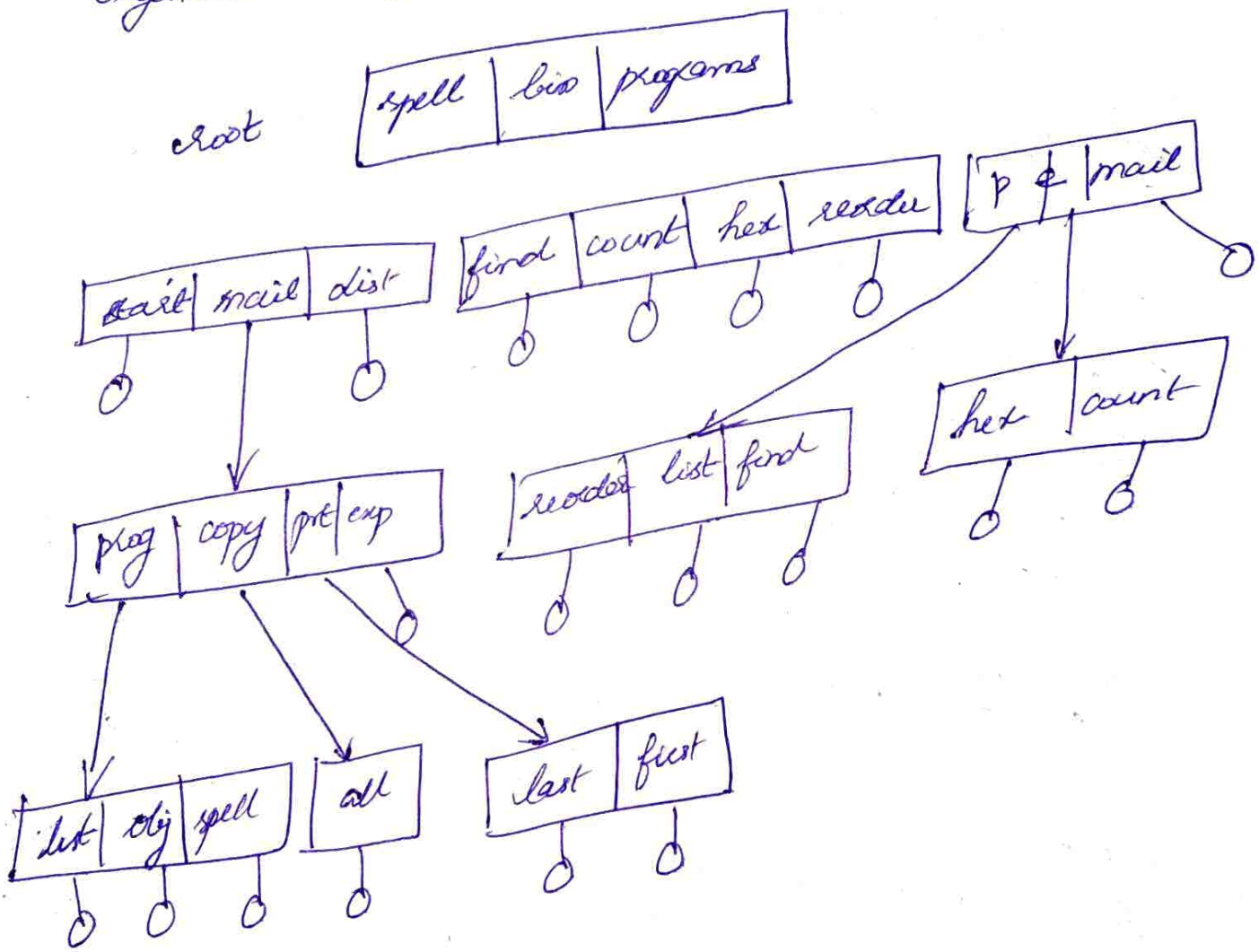
Disadv :-

Users want to cooperate on some task & to access one another's files

③ Tree - Structured directories:

An extension of 2-tiered directory.

- Allows users to create their own subdirectories & to organize their files.



Tree traversing

Tree can be traversed in 2 ways.

① Absolute path name
- Begins search from the root ⇒ D: /os/unit4/chap9.

② Relative path name
- Begins search from the current directory.
⇒ Unit4/chap9

Deletion of directory:

If it is empty, its entry in its containing directo is deleted. If not empty, MS-DOS will not delete a directory unless it is empty. UNIX rm cmd is used to delete directory

Advantage:

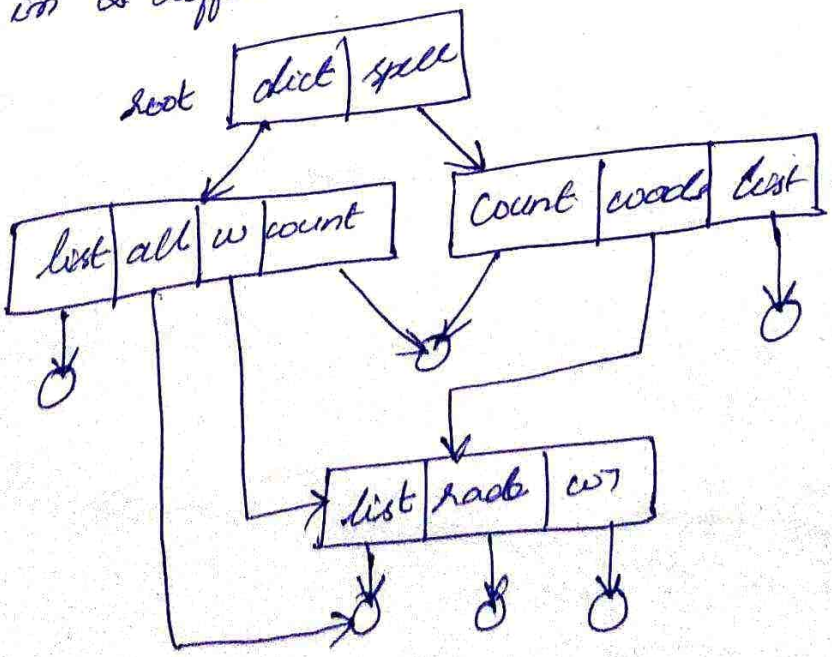
- User can create his own sub-directory
- Separate directories for files associated with different topics can be created.

Limitation:

- Sub-directories cannot be shared b/n users
- path to a file will be longer.

(4) Acyclic-graph directories:-

- Graph with no cycles that allows directories to share subdirectories & files. The same file or subdirectory may be in 2 different directories.



Implementation of shared files & subdirectories

2 approaches:

① Create a new directory entry called a link. A link is effectively a pointer to another file or subdirectory.

② Duplicate all info. about shared files in both sharing directories.

Adv:

More flexible than simple tree structure

Disadv:

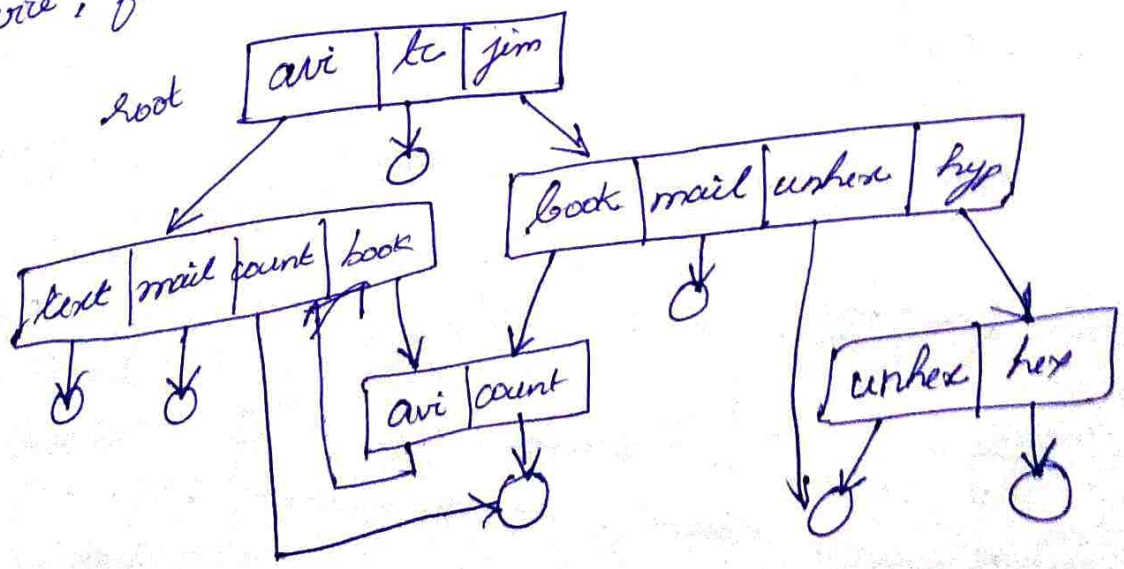
More complex

Ensures that no cycles are formed.

⑤ General graph directory:

When the links to an existing tree-structures directory are added, the tree structure is destroyed resulting in a simple graph structure.

If cycles are allowed to exist in the directory, we likewise want to avoid searching any component twice, for reasons of correctness as well as performance.



Garbage collection:

- scheme determines when the last reference has been deleted and the disk space can be reallocated.
- involves traversing the entire file system, marking everything that can be accessed.
- Cycles are avoided, and no extra overhead is incurred.

Advan

Relative simplicity of the algm to traverse the graph & to determine when there is no more reference to a file

7. FILE SHARING:

File sharing is the sharing of data among multiple users with various levels of access privilege.

- For multi-user os
  - Sharing of files on multi-user sys is desirable
  - Sharing may be done through a protection scheme
- File sharing over the n/w
  - FTP
  - WWW
  - NFS

Remote file sys:

The client-server model

- Server

To declare a resource is available to clients

To specify exactly which resource and exactly which clients.

- Client
  To access the remote files
- The protocol
  - Authentication
  - Operations

Distributed information systems
- Also called as distributed naming services
- DNS - domain name to IP address.

Failure Modes:
- RAID is used to prevent data loss
- For n/w machines
  - Stateful mode - Too many pbm can cause operation to fail
  - Stateless.

8. FILE PROTECTION
The information stored in a computer system must be kept safe from physical damage and improper access.

Types of access:
Access is permitted or denied depending on several different types of operations.
- Read
- Write
- Execute
- Append
- Delete
- List

Access Control

The most common approach to the protection pbm is to make access dependent on the identity of the user.

access-control list (ACL)

- This is a list that specifies the user names and the types of access allowed for each user

Advan: Enabling complex access methodologies

pbm: access list length.

Condensed version of the access list.

1) Owner
2) Group
3) Universe.

9. FILE SYSTEM STRUCTURE.

Characteristics:

- A disk can be rewritten in place.
- A disk can access directly any given block of information it contains.

Design problems:

① Defining how the file s/m should look to the user.

② Creating algm & DS to map the logical file system onto the physical secondary storage device.

Layered file s/m:

Has many levels. Each level produces the o/p for the next higher level.

I/O control:-

- This is the lowest level
- Consists of device drivers & interrupt handlers to transfer information b/n the main mry & the disk system.

Basic file sym:

- needs only to issue generic commands to the appropriate device drivers to read and write physical blocks on the disk.

File organization module

By knowing the type of file allocation used & the location of the file, the file organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.

Logical file system:

- manages metadata information.
- manages the directory structure to provide the file organization module with the information the latter needs, given a symbolic file name.

10. FILE SYSTEM IMPLEMENTATION

Done by,

- On-disk structure
- In-memory structure

On-disk :-

On-disk is the file sym may contains info. about

- how to boot an OS
- The total no. of blocks
- The no. & location of free blocks
- The directory structure
- Individual files.

Boot control block

- Contains information needed by the sym. to boot an OS from the volume.

Volume control block.

- Contains volume details, such as the no. of blocks in the partition, size of the blocks, free block count & free-block pointers, and free FCB count and FCB pointers.

FCB
- Contains the details about the file, including file permissions, ownership, size, and location of the data blocks.

In-memory structure
The structures may include
- In-memory mount table
- In-memory directory-structure
- System-wide open-file table
- Per-process open file table.

File creation
- An appln program calls the logical file system
- Reads the appropriate directory into the mem
- Updates it with the new file name & FCB &
- Writes it back to the disk.

File I/O operation (open & Read)
- The open() call passes a file name to the file s/m.
- Search for the given file name in the directory structure
- Once file is found, the FCB is copied into a s/m wide open file table in mem.
- Entry is made in the per-process open-file table pointing to the entry in the existing s/m wide open-file table.

Partitions & Mounting.

A disk can be sliced into multiple partitions. Each partition can be either raw or cooked.

Raw disk:
- used where no file s/m is appropriate
- RAID system.

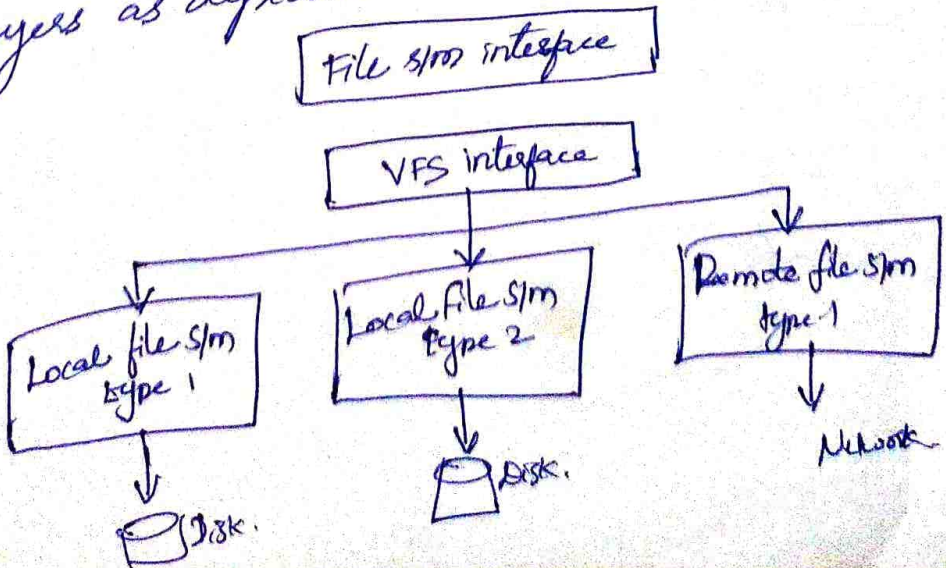
Boot information:
- stored in separate partition

Boot loader:
- contains multiple file s/m & multiple OS occupy the boot space.

Boot partition:
- contains the OS kernel and other system files.
- OS verifies the device contains a valid file s/m in the successful mount operation.

Virtual file systems:

The file s/m implementation consists of 3 major layers as depicted in the below fig.



11. DIRECTORY IMPLEMENTATION

Directory is implemented in 2 ways.

① Linear list
② Hash table

① Linear list:

Linear list of file names with pointer to the data blocks is the simplest method of implementing a directory which requires linear search to find a particular entry.

- Simple to program
- Time consuming to execute
- File creation
- File deletion
- File reuse.

Advan:
Decrease the time required to delete a file.

Disad:
- Searching a file is slow.

② Hash Table.

- Hash table is a linear list with hash data structure.
- Hash table takes a value computed from the file name & returns a pointer to the file name in the linear list.

Collisions is a situation where 2 file names hash to the same location. Alternatively a chained overflow hash table can be used to avoid collisions.

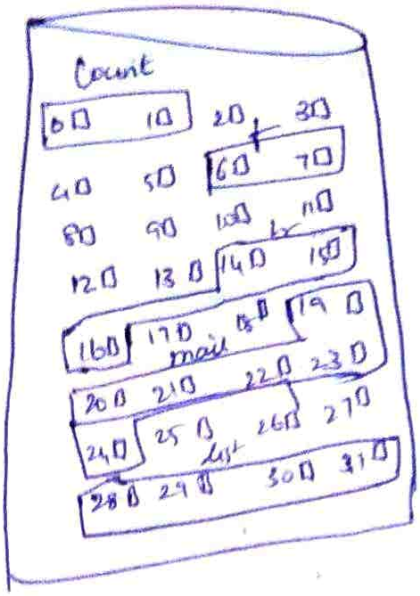
Disadv:
Hash tables are fixed size
Dependence of the hash fn on that size.

12. ALLOCATION METHODS:

Allocation method is a method used to allocate space to the files for
- Utilizing the disk space effectively
- Accessing the files quickly.

The 3 methods for allocating disk space are,
① Contiguous allocation
② Linked allocation
③ Indexed allocation

① Contiguous allocation:
- Each file occupies a set of contiguous blocks on the disk
- Both sequential & direct access are supported
- Directory entry of each file contains
  - Address of the starting block
  - Length of the area allocated for this file.



directory

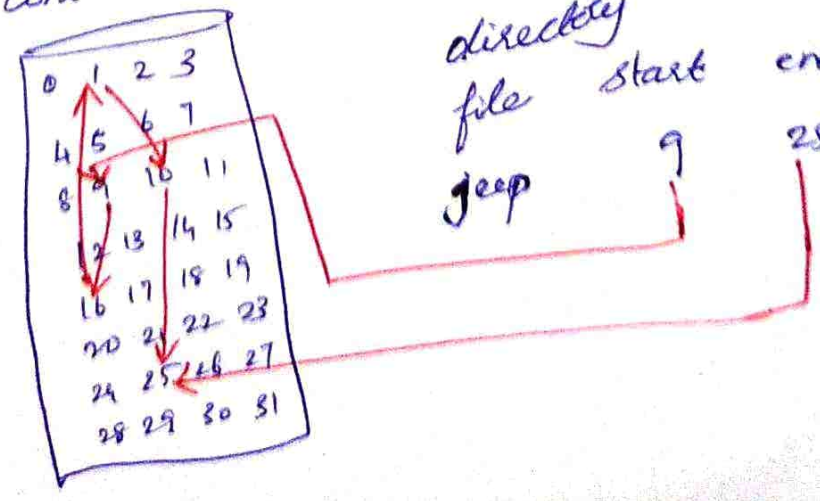
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Disadv.

- Difficult to find free space for a new file
- Suffers from external fragmentation.

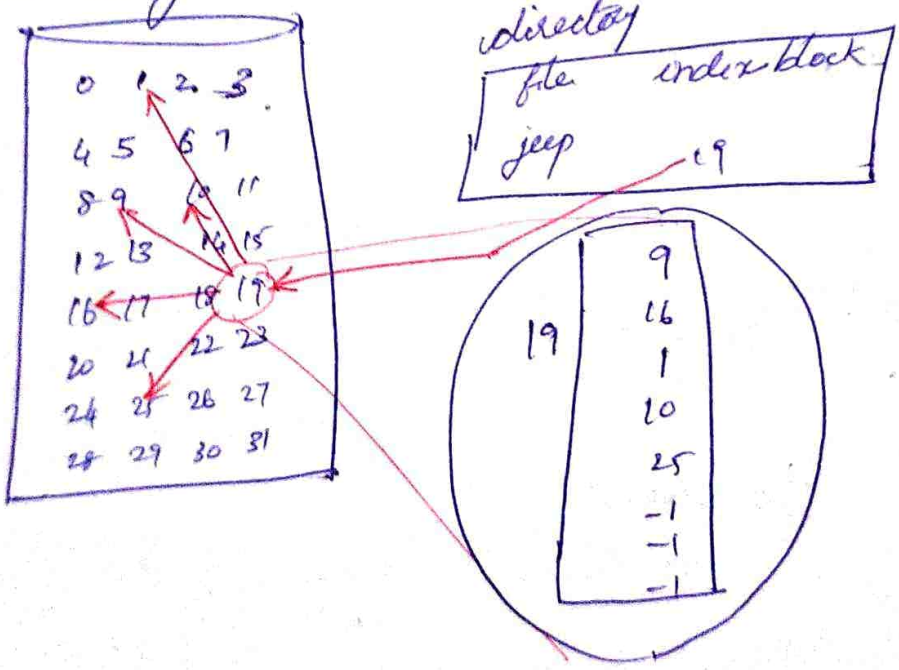
② Linked Allocation

- Each file is a linked list of disk blocks, the disk blocks may be scattered on the disk.
- Directory contains
  - pointer to the first & block of the file
  - pointer is initialized to null to signify an empty file.
- Each block contains a pointer to the next block & the last block contains NUL pointer.



② Indexed Allocation

- This is the solution to the problem of both contiguous & linked allocation.
- Each file has its own index block which is an array of disk block addresses. The i^{th} entry in the index block points to the i^{th} block of the file.
- The directory contains the address of the index block
- When a file is created all pointers in the index block are set to NULL
- When i^{th} block is written a block from the free space manager & its address is put in the i^{th} index block entry.



13. FREE SPACE MANAGEMENT

- Mechanism of maintaining the free space list to keep track of free disk space

- File creation:
  - search the free space
  - space is removed from the free space list & allocate it to the new file.
- File Deletion:
  Disk space is added to the free space list.

Types of mechanism:
Free space list can be maintained in the ways.

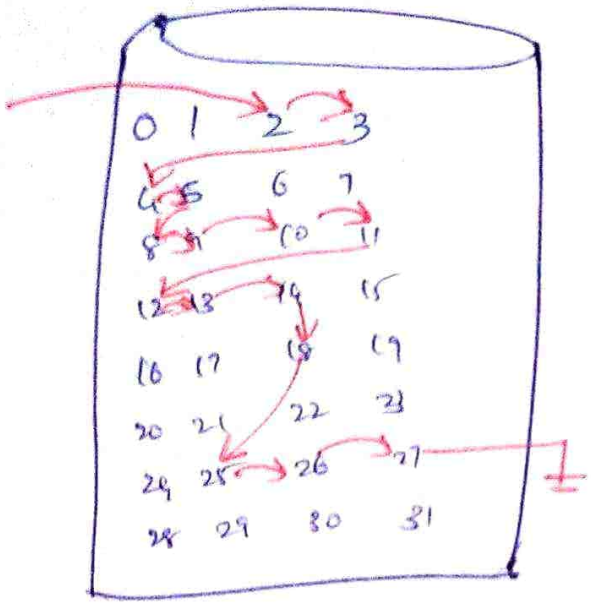
① Bit Vector
② Linked List
③ Grouping
④ Counting

① Bit Vector:
- Free space list is maintained as bit map vector.
- Block is represented as 1 if it is free or 0 if it is allocated.
- Finding the first free block is difficult
- Calculation of the block no.

(no. of bits per word) × (no. of 0-value words) + offset of first 1 bit

② Linked list:

The first block contains a pointer to the next free disk block and so on.



2 is the first free block.

③ Grouping.

- A modification of the free list approach is to store the address of n free blocks in the first free block.
- The first n-1 of these blocks is free
- The last block contains the address of another n free blocks & so on.

④ Counting.

Rather than keeping a list of n free disk addresses, we can keep the address of the first free block & the no. n of free contiguous blocks that follow the first block.

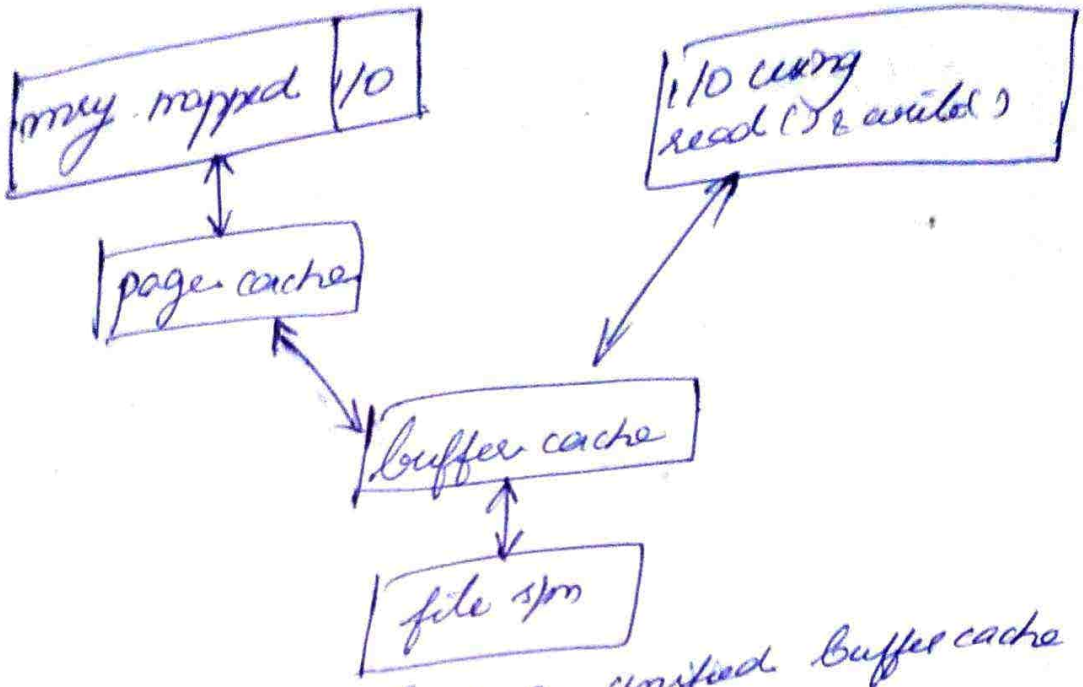
14. EFFICIENCY & PERFORMANCE.

Efficiency:

- UNIX pre-allocates inodes,
- UNIX distributes inodes across the disk and tries to store data files near their inode to reduce the distance
- Syms. use variable size clusters depending on the file size
- mode data in a directory leads to have more re-written method.
- Address scheme has developed.
- Kernel table sizes are fixed.

Performance

- Some OSes cache disk blocks they expect to need again in a buffer cache.
- A page cache ~~disk blocks~~ is connected to the virtual mem s/m is actually more efficient as mem address do not need to be converted to disk block address & back again.
- Some s/m use page caching for both process pages and file data in a unified virtual mem.



I/O without a unified buffer cache.

- Solaris has many variations resulting in priority paging.
- Another issue affecting performance is the question of whether to implement synchronous writes or asynchronous writes.

2 policies:

- Free behind frees up a page ~~ASAP~~ as soon as the next page in the file is requested.
- Read-ahead reads the requested page & several subsequent pages at the same time with the assumption that those pages will be needed in the near future.

- The caching s/m & asynchronous writes speed up disk writes

15. RECOVERY

A consistency checker is run at boot time or mount time, particularly if a file sys was not closed down properly

Problems
- Disk blocks allocated to file & also in free list
- Disk block neither allocated nor in the free list
- Disk block allocated to more than one file.
- 2 or more identical file name
- Illegally linked directories

Log structured file system:
Log based transaction-oriented file sys
say for any given transaction either completes successfully or can be rolled back to a safe state before the transaction commenced.
- All metadata changes are written to a log.
- Changes are known as transactions.
- As changes are written to a log they are said to be committed.
- When all changes are completed the transaction can be removed from the log
- The log will contain information pertaining to uncompleted transactions only.

Backup & Restore:

- Files should be copied to some removable medium.
- A full backup copies every file on a file s/m
- Backup tapes are reused for daily backups
- Full backup should be kept forever.
- Backup should be tested if it is readable
- Keep backup tapes secure

16. MASS STORAGE STRUCTURE

17. I/O SYSTEM

An I/O system is the s/m which is used to manage & control I/O operations and I/O devices

I/O subsystem exhibits two conflicting trends

1) Increasing standardization of d/w & s/w interface
2) Increasing broad variety of I/O devices.

Device drivers:

The device drivers are to provide a uniform device-access interface to the I/O subsystem.

18. I/O hardware.

I/O devices are categorized as

- Human readable
  - used to communicate with the user
  - printers
  - Video display terminals
    - Screen
    - Keyboard
    - Mouse
- Storage devices
  - Used to store data
    - Disks
    - Tapes
- Machine readable
  - Used to communicate with electronic equipment
  - Disk & tap drives
  - sensors
  - Controllers
- Transmission Devices
  - Used to communicate with remote devices
  - N/w cards
  - Modems

Bus structure:

Port - communicate with the machine via communication point.

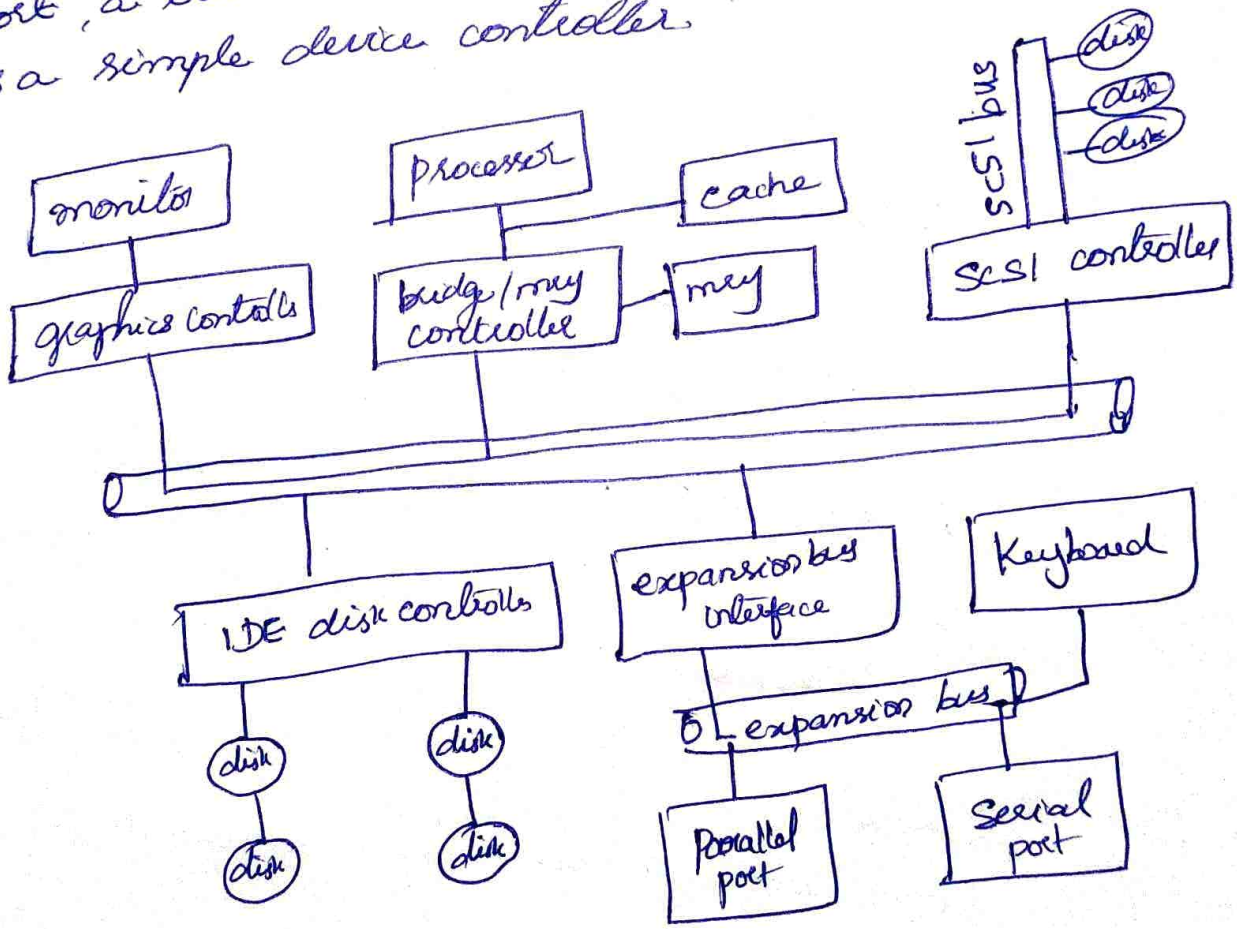
Bus - set of wires and protocol that specifies the msg sent over the wires.

Daisy chain - Device A is plugs into B and B has a cable plug to C. This is called daisy chain.

PCI bus
- Connects the processor-memory subsystem to the fast devices.

Expansion Bus
- connects relatively slow devices such as keyboard and serial & parallel ports.

Controller -
Collection of electronics that can operate a port, a bus or a device. A serial-port controller is a simple device controller.



A typical PC Bus structure.

I/O Registers

I/O ports consists of 4 registers.

1) Status register
2) Control register
3) Data-in
4) Data-out

Polling

The host is said to be busy-waiting or polling which is a loop reading the status register over & over until the busy bit becomes clear.

Handshaking

Handshaking is the complete protocol for interaction b/n the host and a controller.

Interrupts

The H/W mechanism that enables a device to notify the CPU is called an interrupt

Features of interrupt-handling

1. The ability to defer interrupt handling during critical processing
2. An efficient way to dispatch to the proper interrupt handler for a device without first polling all the devices to see which one raised the interrupt

Interrupt request lines.

The CPUs have two interrupt request lines

1) Non maskable interrupt
- reserved for events such as unrecoverable memory errors

2) Maskable interrupt
- is used by device controllers to request service

Direct memory access:-
DMA is used to avoid programmed I/O for large data movement. It requires DMA controller and bypasses CPU to transfer data directly b/n I/O device and memory.

19. APPLICATION I/O INTERFACE
Device-driver layer hides differences among I/O controllers from kernel.

Characteristics of I/O devices
Each general kind is accessed through a standardized set of functions - an interface. A given device may ship with multiple device drivers - for instance, drivers for MS-DOS, windows 95/98, windows NT/2000 and Solaris

Devices vary in many dimensions.
- character-stream or block
- sequential or random access
- synchronous or asynchronous
- shareable or dedicated
- speed of operation
- Read-write, read only, or write only.

Block and character devices.

① **Block device:**

The block device interface captures all the aspects necessary for accessing disk drives and other block oriented devices.

- Block devices include disk drives
- Commands include
  - Read
  - Write
  - Seek.
- Raw I/O or file-system access
  - Mode of access a block device as a simple linear array of blocks.
- Memory-mapped file access
  - Provides access to disk storage via an array of bytes in main mem.

② **Character devices:**

- Access through a character stream interface

Character devices include keyboards, mice & serial ports.

Network devices:

Most OS provide a n/w I/O interface that is different from the read()-write()-seek() interface used for disks. One interface available in many OS, including UNIX and windows NT, is the n/w socket interface

Clocks and Timers

Functions of clocks and Timers

- Give the current time
- Give the elapsed time
- Set a timer to trigger operation X at time T.

Programmable interval timer

The h/w to measure elapsed time & trigger operations is called a programmable interval timer. It is used for timing interrupt, periodic interrupt

Construction of h/w clock

The h/w clock is constructed from a high-frequency counter whose value can be read from a device register. Although this clock does not generate interrupts it offers accurate measurements of time intervals

Blocking & Non blocking I/O

Blocking I/O

- Process execution is suspended until I/O completed
- Easy to use and understand
- Insufficient for some needs

Nonblocking I/O

- I/O call returns as much as available
- User interface, data copy
- Implemented via multithreading.

Asynchronous I/O

- process runs while I/O executes difficult to use
- I/O subsystem signals process when I/O completed

20. KERNEL I/O SUBSYSTEM

The services provided by the kernels I/O subsystem are

- scheduling
- buffering
- caching
- spooling
- device reservation
- error handling

I/O scheduling:

I/O scheduling is used to schedule a set of I/O requests to determine a good order in which to execute.

Buffering:

A buffer is a mry area that stores data while they are transferred b/n two devices or b/n a device and an application.

Caching

A cache is a regoon of fastest mry that holds copy of data.

Cached copy is more efficient than access to the original

caching - fast mry holding copy of data.

Spooling:
A spool is buffer that holds o/p for a device, such as printer, that cannot accept interleaved data streams.

Error handling:
. OS can recover from disk read, device unavailable, transient write failure
. Most return an error no. or code when I/O request fails.
. System error logs hold problem reports.

21. STREAMS
A stream is a full-duplex communication channel b/n a user-level process & a device in UNIX system V.

Structure:
A stream consists of Stream head interfaces with the user process. Driver end interfaces with the device.
Each module contains a read queue and a write queue
Message passing is used to communicate b/n queues
Accept msg & immediately sends to the queue in the adjacent module without buffering them

Accessing STREAMS:

Standard functions are,

- open()
- close()
- read()
- write()
- ioctl()
- pipe()
- putmsg()
- putpmsg()
- getmsg()
- getpmsg()
- poll()

Messages are accessible using.

- putmsg()
- putpmsg()
- getmsg()
- getpmsg()
- read()
- write()

Benefits:

- permits layered and de-layered multiplexing
- Message-based interfaces enable off-board protocol migration

22. PERFORMANCE

I/O a major factor in sys performance.

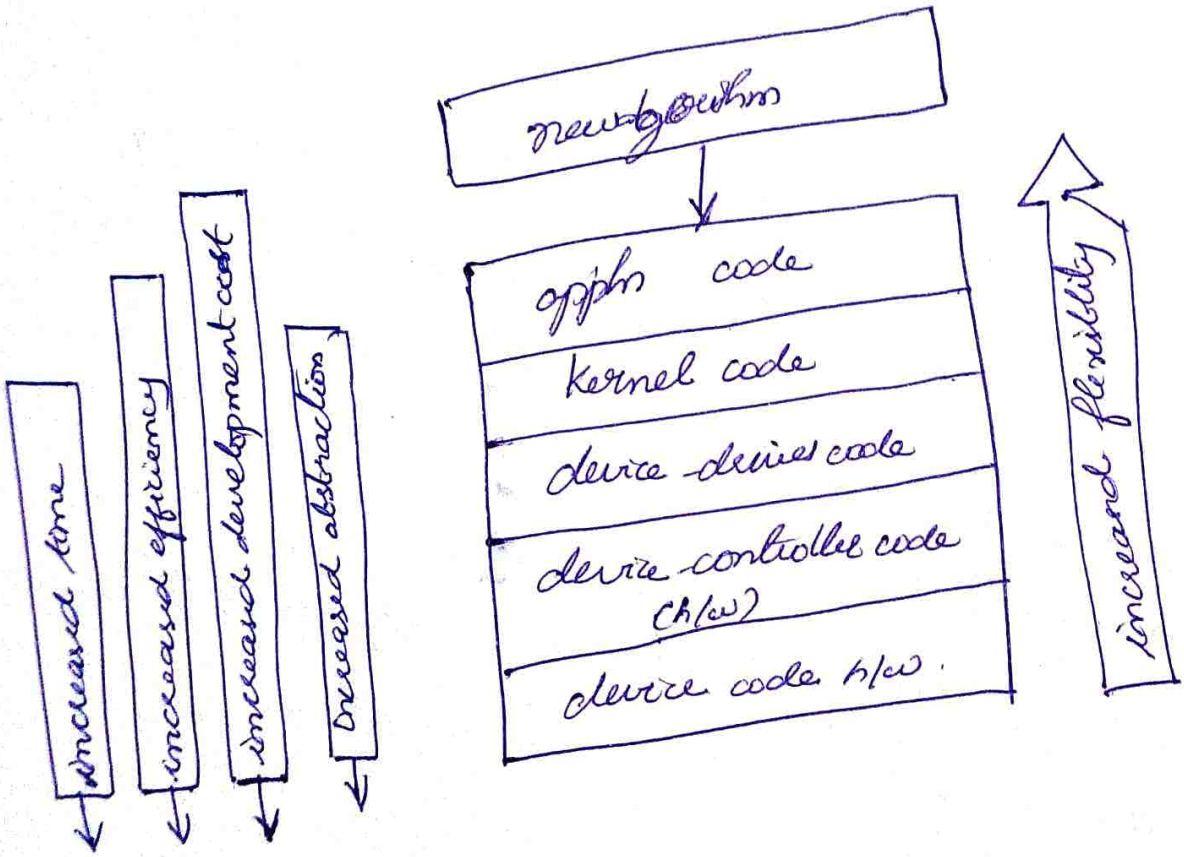
- Demands CPU to execute device driver, Kernel I/O code.
- Context switches due to interrupts
- Data copying
- N/w traffic especially stressful.

Improving performance:

- Reduce no. of context switches
- Reduce data copying
- Increase concurrency by using DMA
- Move processing primitives into H/w

Device - functionality progression:

1) Implement experimental I/O algorithms at the application level.
2) Re implement the appln - level algm in the kernel
3) Highest performance may be obtained by a specialized implementation in h/w either in the device or in the controller.



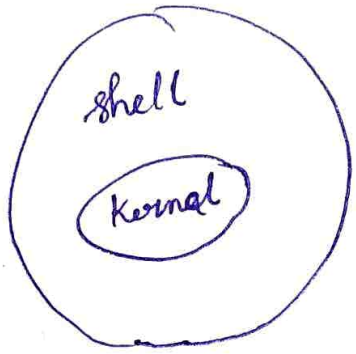
Device functionality progression.

UNIT-V

LINUX CASE STUDY

1. Introduction to LINUX:

- Developed in 1991 by Linus Torvalds.
- Used in computers ranging from super computer to embedded system.
- This is a free OS based on UNIX standards.
- This is multiuser, multitasking, time sharing and monolithic kernel.



- Kernel is the main pgm of UNIX. It controls h/w, CPU, mey, hard disk, n/w card etc.
- Shell is an interface b/n user and Kernel. This interprets i/p as commands and passes them to Kernel

2. Design Principles:

- LINUX is a multi user, multitasking s/m with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX n/w model.

- Its main design goals are speed, efficiency and standardization.
- Linux is designed to be compliant with the relevant POSIX documents, at least two Linux distributions have achieved official POSIX certification.
- Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behaviour.

Components of Linux System:

1) Kernel
2) System Library
3) System Utility

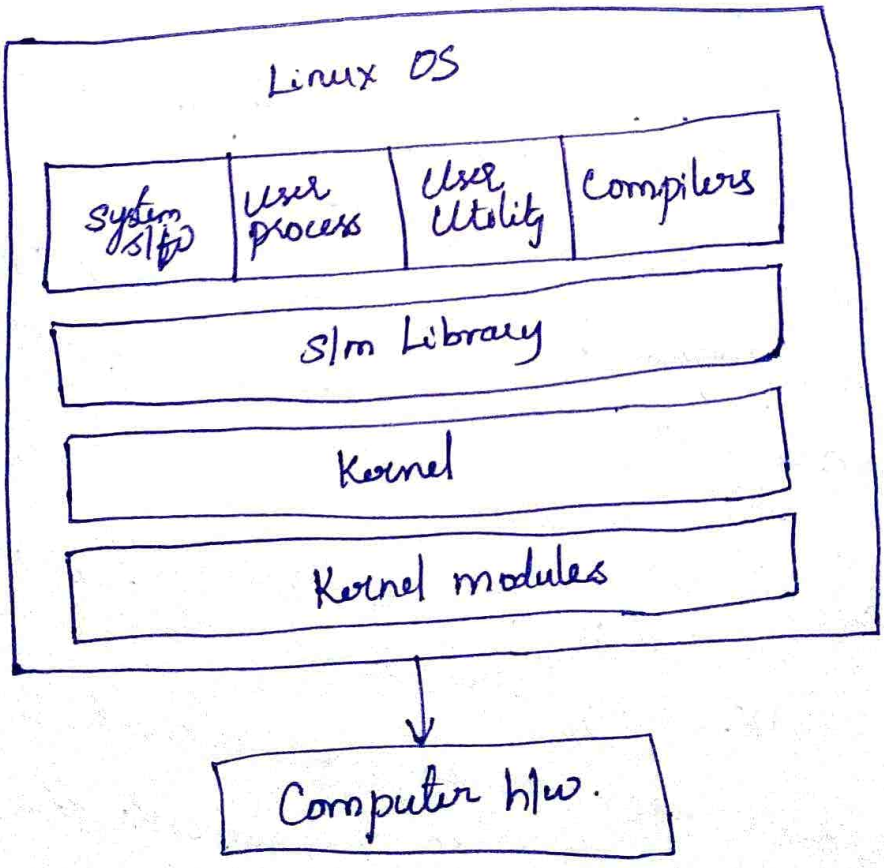


fig:- Linux OS components.

1) Kernel:
- Heart of the heart of the Linux OS.
- responsible for all activities of OS.
- hide low level h/w information to user or apln pgm.
- parts of kernel are
  - Process management
  - mem mgmt
  - h/w device drivers
  - file sym drivers
  - n/w mgmt etc.

2) System library:
- implements most of the functionalities of the OS and do not require kernel module's code access rights.

3) System Utility
- programs do specalized individual level tasks.

Linux Utilities :-
- has 2 types open source and commercial.
Open source are free to access package.
Commercial care paid and some are free (Real Player)
(BRU tape backup).

Commands in Linux:

1) External commands:-
   ex) cat, ls
   These are executed by the kernel.

2) Internal commands:-
   These are executed by the shell itself.
   ex) echo, cd.

3. Kernel Module:
- Modules are pieces of code that can be loaded & unloaded into the kernel upon demand. Kernel module runs in privileged kernel mode.
- Linux kernel is modular, which means it can extend its capabilities through the use of dynamically loaded kernel modules.
- Kernel modules are loaded only at the time of their need.
- Allows a Linux s/m to be set up with a standard minimal kernel without any extra device drivers built in.

The module support following Linux components.
1) Module mgmt s/m
2) Module loader & unloader
3) Driver registration s/m
4) Conflict-resolution mechanism.

Module mgmt sym:-
- modules are loaded into the main my to communicate with the rest of the kernel.
- 2 steps to load a module
① Kernel reserves a continuous area of virtual my for the module. The address of allocated my is returned by kernel. Loader uses this address for loading machine code.
② Module passes the sym call & symbol table to kernel.

Driver Registration:-
Kernel maintains a dynamic table of all known drivers.
Registration tables include following:
① Device drivers - character devices, block devices & network interface devices.
② File systems:- contains formats for storing files
③ Network protocol:- required the packets
④ Binary format:- recognizing, loading & executing a new type of executable file

Conflict resolution:-
Aims to
① prevent modules from clashing over access to hardware
② prevent auto probes from interfering with existing device driver
③ Resolve conflicts with multiple drivers trying to access the same hw

3. PROCESS MANAGEMENT:

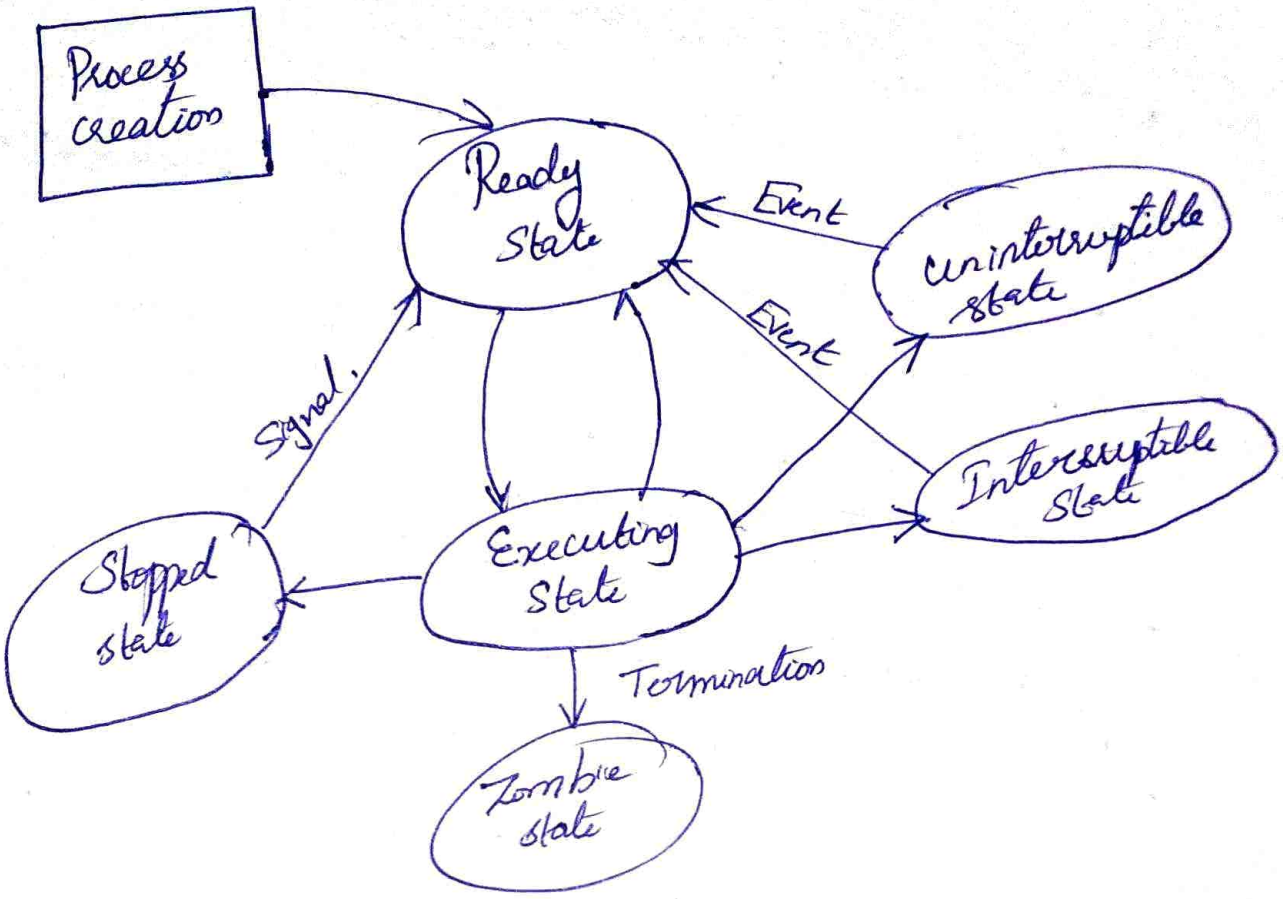
In Linux task represent both processes & threads. To manage multi-tasking, OS needs to use a DS to keep track of every task's progress. Every task needs its own private stack.

1) State :- are executing, ready, suspended, stopped, zombie
2) Scheduling information :- data related for scheduling proce
3) Identifiers :- Each process has its own identifier
4) IPC :- mechanisms are pipes, shared mey, socket etc.
5) Links :- Each process has a link with a previous process like a parent or sibling.
6) Times & Timers :- maintains proces creation time, execution period etc
7) File system : Process can open or close file as it needed.
8) Virtual memory :- Virtual mey should be mapped to the sym's physical mey.
9) Process specific context :- For a suspended process all information should be ~~safe~~ saved.

Linux Process :-

Process in linux is represented by task-struct DS

It contains various information like, State, IPC, scheduling information, Links, Identifier Times, File system.



State model.

Linux thread:-

- Older version of Linux Kernel does not support multithreading.
- Linux does not recognize a distinction b/w thread & process
- New thread is created in Linux by copying the attributes of the current process.
- If 2 processes share the same virtual mem, they act like a thread within a single process.
- clone() command is used for creating a process.

4. MEMORY MANAGEMENT

Management of physical memory

It has 2 components

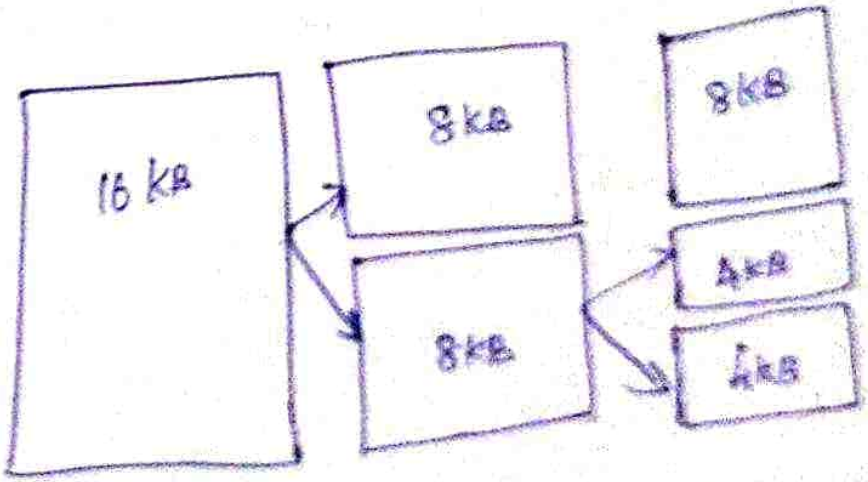
- Allocating & freeing physical mem pages, groups of pages and small blocks of RAM.
- Handling virtual mem.

Linux separates physical mem into 4 zones

- DMA – first 16Mb of physical mem
- ZONE DMA32 – first 4GB
- ZONE NORMAL – first 896 MB of address space
- ZONE HIGHMEM

In Linux, a slab may be in one of three possible states

- Full – All objects in the slab are marked as used
- Empty – all objects in the slab are marked as free
- Partial – The slab consists of both used and free objects



Splitting of mem in the buddy sys

Completely fair scheduler (CFS):

- CFS allocates CPU resources fairly.
- Each process would receive 1/n of the processor's time, n - is the no. of runnable processes.
- CFS choses Round-Robin scheduling process.
- CFS keep track of the fair share of CPU and make it available to each process in the s/m.
- CPU time is calculated by dividing the wall time by the total no. of processors waiting.
- Wait time is represented by per-task wait_runtime
- Scheduler selects the process with longest wait time & assign to the CPU.

Linux process scheduling policy:

- Linux uses a time sharing technique
- Linux schedule process according to a priority ranking.
- Process that have not received the CPU for a long time get their priorities increased
- Process preemption is done when time quantum has expired.
- (e) text editor's priority is higher than a Compiler

Virtual memory:

The VM sym maintains the address space visible to each process. It creates pages of virtual mey on demand, and manages the loading of those pages from disk or their swapping back out to disk as required.

VM Regions:

- A logical view describing instructions concerning the layout of the address space
- A physical view of each address space which is stored in the H/w page table for the process

VM regions are characterized by:

- The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing
- The regions reaction to writes

Kernel creates a new virtual address space

- When a process runs a new pgm with the exec() sys call
- Upon creation of a new process by the fork() system call

On executing a new pgm, the process is given a new, completely empty virtual address space. the pgm. loading routines populate the address space with virtual-mey regions.

Swapping & Paging.

The VM paging sys relocate pages of mey from physical mey out to disk when the mey is needed for something else.

The VM paging sys can be divided into 2 sections

① The pageout-policy algm decides which pages to write out to disk and when the paging mechanism actually carries out the transfer, and pages data back into physical mey as needed.

Kernel Virtual memory

Has 2 regions.

A static area that contains page table references to every available physical page of mey in the sys, so that there is a simple translation from physical to virtual addresses when running kernel code

The remainder of the reserved section is not reserved for any specific purpose its page table entries can be modified to point to any other areas of mey

Execution and loading of user program

- Linux maintains a table of functions for loading pgm.
- The registration of multiple loader routines allows Linux to support both the ELF and a.out binary formats.
- Initially, binary-files pages are mapped into virtual mem.
- An ELF-format binary file consists of a header followed by several page-aligned sections.

Static & Dynamic Linking

- A pgm whose necessary library fns are embedded directly in the pgm's executable binary file is statically linked to its libraries
- The main disadvantage of static linkage is that every program generated must contain copies of exactly the same common sys library fns.
- Dynamic linking is more efficient in terms of both physical mem and disk-space usage because it loads the system libraries into mem only once
- Linux implements dynamic linking in user mode through special linker library.
- Maps the link library into mem.
- Shared libraries compiled to be position-independent code (PIC) so can be loaded anywhere

5. INPUT OUTPUT MANAGEMENT

Linux splits all devices into 3 classes

- block devices
- Character devices
- Network devices.

Block device

- Block devices provide the main interface to all disk devices in a system. Performance is particularly important for disks, and the block device s/m must provide functionality to ensure that disk access is as fast as possible. This fundionality is achieved through the scheduling of I/O operations.
- The request manager is the layer to s/f that manages the reading & writing of buffer contents to and from a block-device driver.

Characters device

- A device driver which does not offer random access to fixed blocks of data.
- A character device driver must register a set of fns which implement the drivers various fits I/O operations.
- Line discipline is an interpreter for the information from the terminal device.

Network device:
N/w device are dealt with differently from block and character devices. Users cannot directly transfer data to n/w device. Instead, they must communicate indirectly by opening a connection to the kernel's n/w subsystem.

6. FILE SYSTEM:
A set of definitions that define what a file object is allowed to look like
- The inode object structure represent an individual file
- The file object represents an open file
- The superblock object represents an entire file system
- A dentry object represents an individual directory entry

File Object operations include:
- int open ( )
- ssize_t read ( )
- ssize_t write ( )
- int mmap ( )

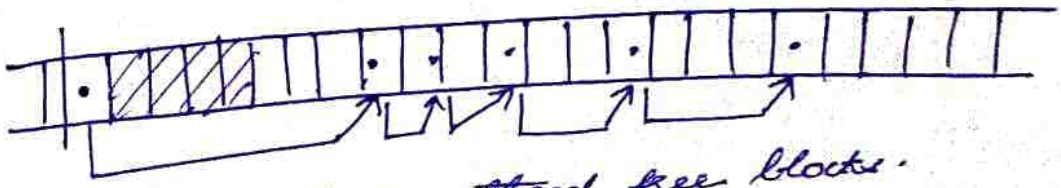
The Linux ext3 file system.

- ext3 is standard on disk file sys for Linux.
- The main difference b/n ext2fs and FFS concern their disk allocation policies.

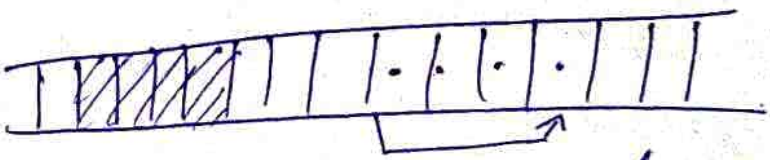
In ffs, the disk is allocated to files in blocks of 8 Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file.

ext3 does not use fragments; it performs its allocations in smaller units

ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a block group.



allocating scattered free blocks.



allocating continuous free blocks.

 block in use.  block selected by allocator.

 free block → bitmap search.

Block allocation policies

Journaling.

- ext3 implements journaling, with file s/m updates first written to a log file in the form of transaction
- On system crash, some transaction might be in journal but not yet placed into file s/m.
- Improves write performance on harddisks by turning random I/O into sequential I/O.

The linux proc file s/m:

- does not store data, rather its contents are computed on demand according to user file I/O requests
- proc must implement a directory structure, and the file contents within. it must then define a unique & persistent inode no. for each directory & files it contains.

when data is read from one of these files, proc collects the appropriate information, formats it into text form and places it into the requesting process read buffer

7. INTER PROCESS COMMUNICATION

The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather communication within the kernel is accomplished via scheduling states & wait queue structures.

Also implements system V Unix semaphores
- Process can wait for a signal or a semaphore
- Semaphores scale better
- Operations on multiple semaphore can be atomic.

Synchronization and Signals.
- Linux mechanism for informing a process that an event has occured is the Signal. Signals can be sent from any process to any other process, with restrictions on signals sent to processes owned by another user.
- Linux uses scheduling states & wait queue structures for communication in the kernel-mode.
- Once the event has completed, every process on the wait queue will be awoken. This procedure allows multiple processes to wait for a single event.

Passing data b/w processes.

Pipe mechanism allows a child process to inherit a comm. channel to its parent, data written to one end of the pipe can be read at the other.

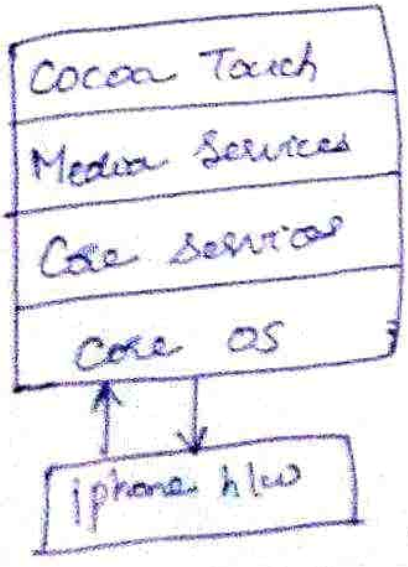
Shared mem offers an extremely fast way of communicating

To obtain synchronization, however, shared mem must be used in conjunction with another Interprocess communication mechanism.

8. MOBILE OS.

IOS

IOS consists of a no. of different s/w layers, each of which provides programming frameworks for the development of apps that run on top of the underlying h/w.



IOS layer.

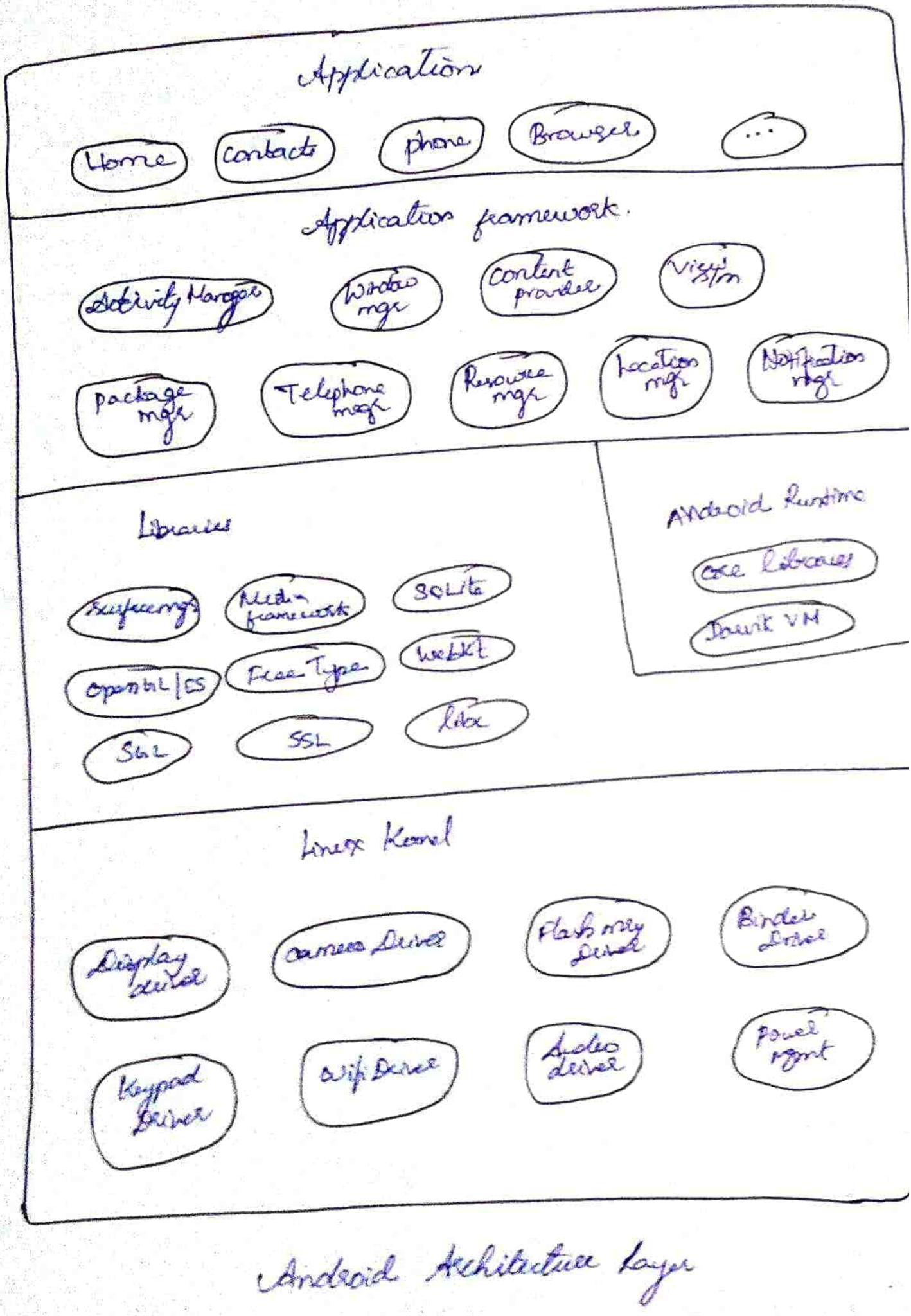
Each OS layer provides an increasing level of abstraction away from the complexity of working with the h/w. As an iOS developer you should, therefore, always look for solutions to your programming goals in the frameworks located in the higher level iOS layers before resorting to writing code that reaches down to the lower level layers.

The cocoa Touch layer provides the following framework for iphone app development.

- UIKit framework
- Map Kit framework
- Push Notification Service
- Message UIFramework.

9. ANDROID (SDK)

- Android is build on top of a Linux kernel which provides a h/w abstraction layer for it.
- A layer consisting of c/c++ libraries are embedded on top of Linux kernel.
- Android runtime has its own custom optimized java virtual machine called Dalvik VM.
- App framework provide various manager to access all h/w related services.



Android Architecture Layer

Application framework.

Appln framework sits on top of native libraries, android runtime & linux kernel. This framework come pre-installed with high-level building blocks that developers can use to program applns.

Activity manager-
- manages the lifecycle of applications

Contents provider
- Store & retrieve data and make it accessible to all appln.

View system
- handles GUI related tasks

Package Manager
- retrieve various kinds of info. related to the currently installed appln on the device

Resource manager.
- Provides access to non-code resources such as icons etc

Location manager.
- location based & related services

Notification
- executes & manages all notifications, alerts etc

Application SDK

- Designed for ready-to-deploy tablet and windows mobile applications.
- Allows developers to customize the appn.
  - Changes existing tasks/fns
  - Integrates new business logic & implemented

10. MEDIA LAYER

- Provide iOS with audio, video, animation & graphics capabilities. Do with the other layers comprising the iOS stack, the media layer comprise a no. of frameworks which may be utilized when developing iphone apps. On
  - Core Video framework
  - Core Text framework
  - Core Graphics framework
  - Core Image framework.

11. CORE SERVICE LAYER

The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built & consist of the following frameworks.

- Address Book Framework
- CF Network Framework
- Core Data Framework

CORE OS LAYER

The core OS layer occupies the bottom position of the iOS stack and as such is directly on top of the device h/w. The layer provides a variety of services including low level n/wing, access to external accessories & the usual fundamental OS services such as mey mgmt, file system handling & threads.

- Accelerate framework
- API for performing complex & large no. math
- External Accessory framework
- Security Framework.

FILE SYSTEM

Android uses several partitions to organise files and folders on the device just like windows OS. Each partitions has its own functionality

Main partitions are,

- /boot
- /system
- /recovery
- /data
- /cache
- /misc