ARUNAI ENGINEERING COLLEGE

(Affiliated to Anna University)
Velu Nagar, Tiruvannamalai —606603
Phone: 04175-237419/236799/237739
www.arunai.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF ENGINEERING

Second Year

Fourth Semester

CS8493- OPERATING SYSTEMS
Lecture By – R.SURESH AP/CSE

CS8493 OPERATING SYSTEMS

OBJECTIVES:

- To understand the basic concepts and functions of operating systems.
- To understand Processes and Threads
- To analyze Scheduling algorithms.
- To understand the concept of Deadlocks.
- To analyze various memory management schemes.
- To understand I/O management and File systems.
- To be familiar with the basics of Linux system and Mobile OS like iOS and Android.

UNIT I OPERATING SYSTEM OVERVIEW 7

Computer System Overview-Basic Elements, Instruction Execution, Interrupts, Memory Hierarchy, Cache Memory, Direct Memory Access, Multiprocessor and Multicore Organization. Operating system overview-objectives and functions, Evolution of Operating System.- Computer System Organization Operating System Structure and Operations- System Calls, System Programs, OS Generation and System Boot.

UNIT II PROCESS MANAGEMENT 11

Processes - Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication; CPU Scheduling - Scheduling criteria, Scheduling algorithms, Multiple-processor scheduling, Real time scheduling; Threads- Overview, Multithreading models, Threading issues; Process Synchronization - The critical-section problem, Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization, Critical regions, Monitors; Deadlock - System model, Deadlock characterization, Methods for handling deadlocks, Deadlock prevention, Deadlock avoidance, Deadlock detection, Recovery from deadlock.

UNIT III STORAGE MANAGEMENT 9

Main Memory – Background, Swapping, Contiguous Memory Allocation, Paging, Segmentation, Segmentation with paging, 32 and 64 bit architecture Examples; Virtual Memory – Background, Demand Paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples.

UNIT IV FILE SYSTEMS AND I/O SYSTEMS 9

Mass Storage system – Overview of Mass Storage Structure, Disk Structure, Disk Scheduling and Management, swap space management; File-System Interface - File concept, Access methods, Directory Structure, Directory organization, File system mounting, File Sharing and Protection; File System Implementation- File System Structure, Directory implementation, Allocation Methods, Free Space Management, Efficiency and Performance, Recovery; I/O Systems – I/O Hardware, Application I/O interface, Kernel I/O subsystem, Streams, Performance.

UNIT V CASE STUDY 9

Linux System - Design Principles, Kernel Modules, Process Management, Scheduling, Memory Management, Input-Output Management, File System, Inter-process Communication; Mobile OS - iOS and Android - Architecture and SDK Framework, Media Layer, Services Layer, Core OS Layer, File System.

TOTAL : 45 PERIODS

OUTCOMES:

At the end of the course, the students should be able to:

- Analyze various scheduling algorithms.
- Understand deadlock, prevention and avoidance algorithms.
- Compare and contrast various memory management schemes.
- Understand the functionality of file systems.
- Perform administrative tasks on Linux Servers.
- Compare iOS and Android Operating Systems.

TEXT BOOK :

1. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, -Operating System Concepts‖, 9th Edition, John Wiley and Sons Inc., 2012.

REFERENCES :

1. Ramaz Elmasri, A. Gil Carrick, David Levine, -Operating Systems – A Spiral Approach‖, Tata McGraw Hill Edition, 2010.
2. Achyut S.Godbole, Atul Kahate, -Operating Systems‖, McGraw Hill Education, 2016.
3. Andrew S. Tanenbaum, -Modern Operating Systems‖, Second Edition, Pearson Education, 2004.
4. Gary Nutt, -Operating Systems‖, Third Edition, Pearson Education, 2004.
5. Harvey M. Deitel, -Operating Systems‖, Third Edition, Pearson Education, 2004.
6. Daniel P Bovet and Marco Cesati, -Understanding the Linux kernel‖, 3rd edition, O‘Reilly, 2005.
7. Neil Smyth, -iPhone iOS 4 Development Essentials – Xcode‖, Fourth Edition, Payload media, 2011.

17/12/2019

UNIT-1.

Operating system:-

OS acts as an intermediatory between user of computer hardware.

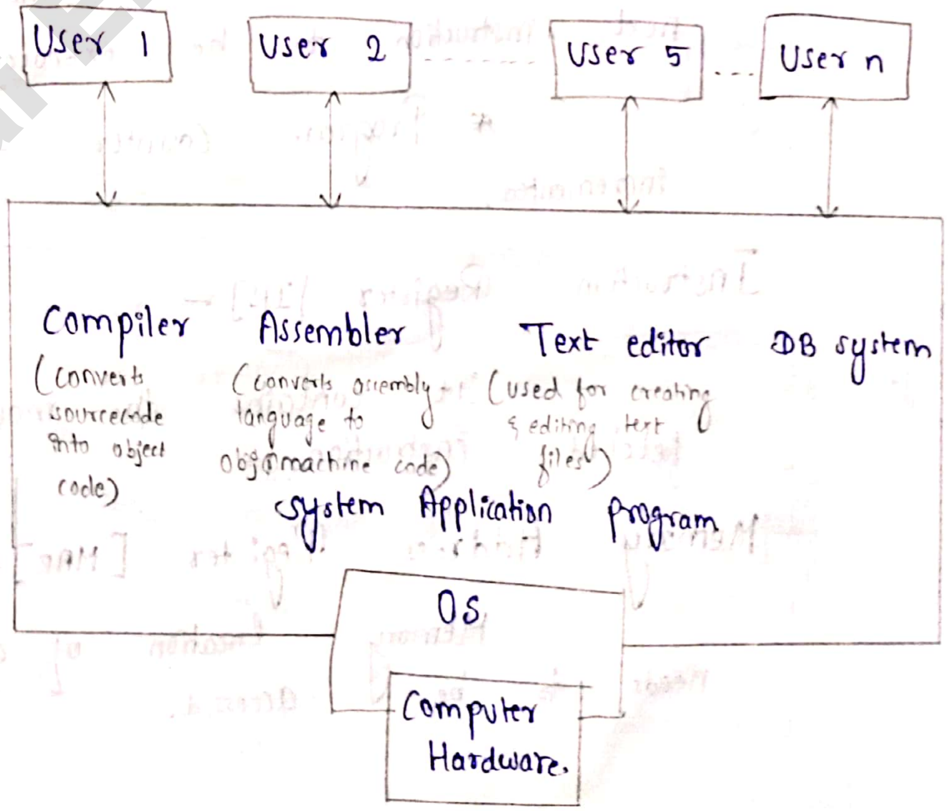
(2m) **Objective:-**

→ Execute User programs & make solving user programs easier.

→ Make Computer system convenient to use.

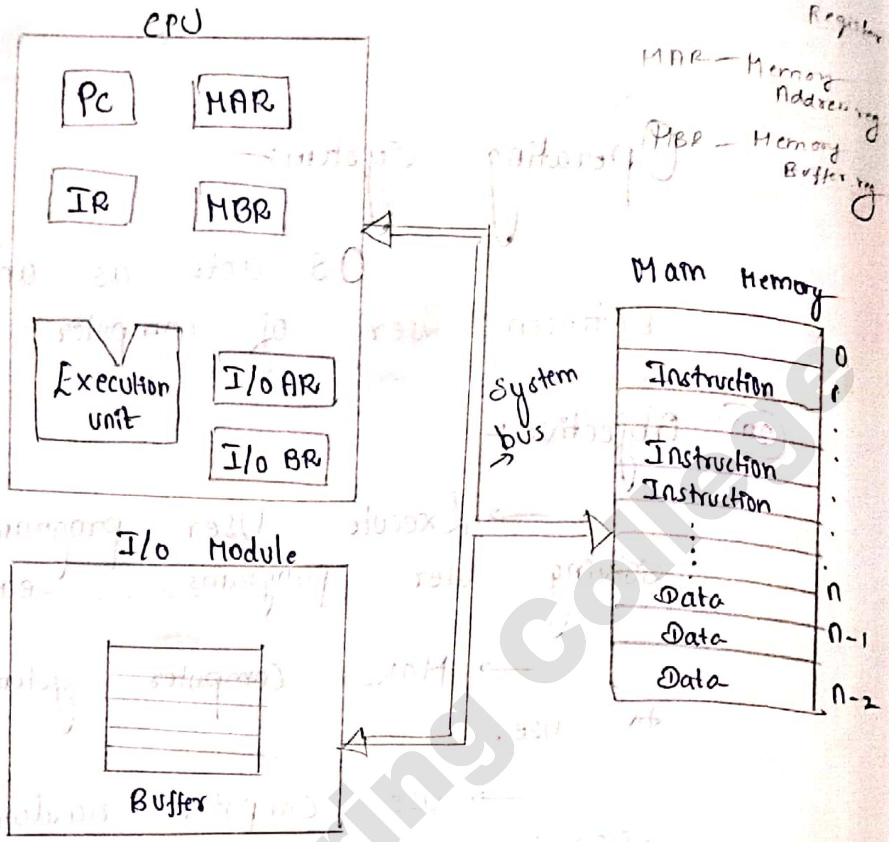
→ Use computer hardware in an efficient manner.

Computer System structure:-


User 1
User 2
User 5
User n
Compiler (converts sourcecode into object code)
Assembler (converts assembly language to obj/machine code)
Text editor (used for creating & editing text files)
DB system
system Application program
OS
Computer Hardware.

Basic Elements :-

PC - Program Counter
IR — Instruction Register
MAR — Memory Addressing
MBR — Memory Buffer reg



Program Counter [PC] :-

* It ~~marks~~ holds the address of next instruction to be executed.

* Program Counter is automatically incremented.

Instruction Register [IR] :-

It contains the most recently fetched instruction.

Memory Address Register [MAR] :-

Memory location of a data that needs to be accessed.

Memory Buffer Register:-

It stores the data that is being transferred to and from the memory to other ~~complex~~. components.

Input output Address Register:-

It specifies the address of particular I/o ~~As~~ device.

Input output Buffer Register:-

Exchange of data between I/o module and CPU.

Registers:-

It is an temperory storage area which is used to minimize a main memory reference by optimizing register use.

Control Register:-

which control the general behaviour of CPU or other digital device.

Eg:- Interrupt control.

Status Register:-

Contains information about state of processor.

(eg) C - carry flag Z - Zero flag

User Visible Register + I/o Modules.

- + Segment pointer
- + Stack pointer
- + Pc
- + IR

+ system Bus

Segment pointer :-

When memory is divided into segments, it is referenced by the segment an offset.

Stack pointer :-

It points to the top of the stack.

I/o Modules :-

It moves the data between computer and external environment.

It contains internal buffer for temperory holding the data.

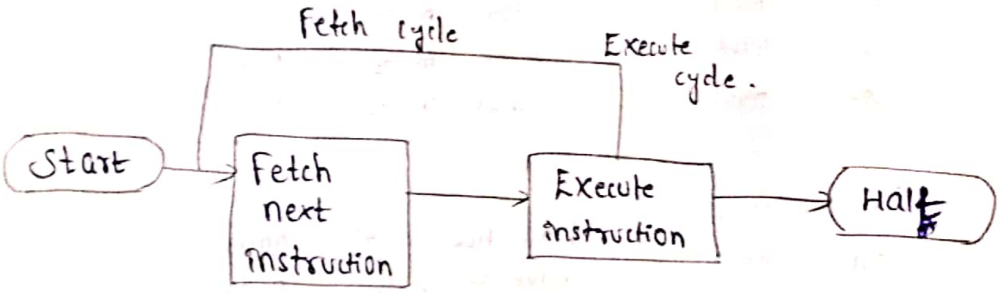
system Bus :-

It provides the Communication between memory processor and I/o module.

Instruction Execution :-

Instruction cycle :-

Processing require for a single instruction.



Steps to Execute the instruction:-

* The instruction at the address stored in program counter is fetched and loaded into IR.

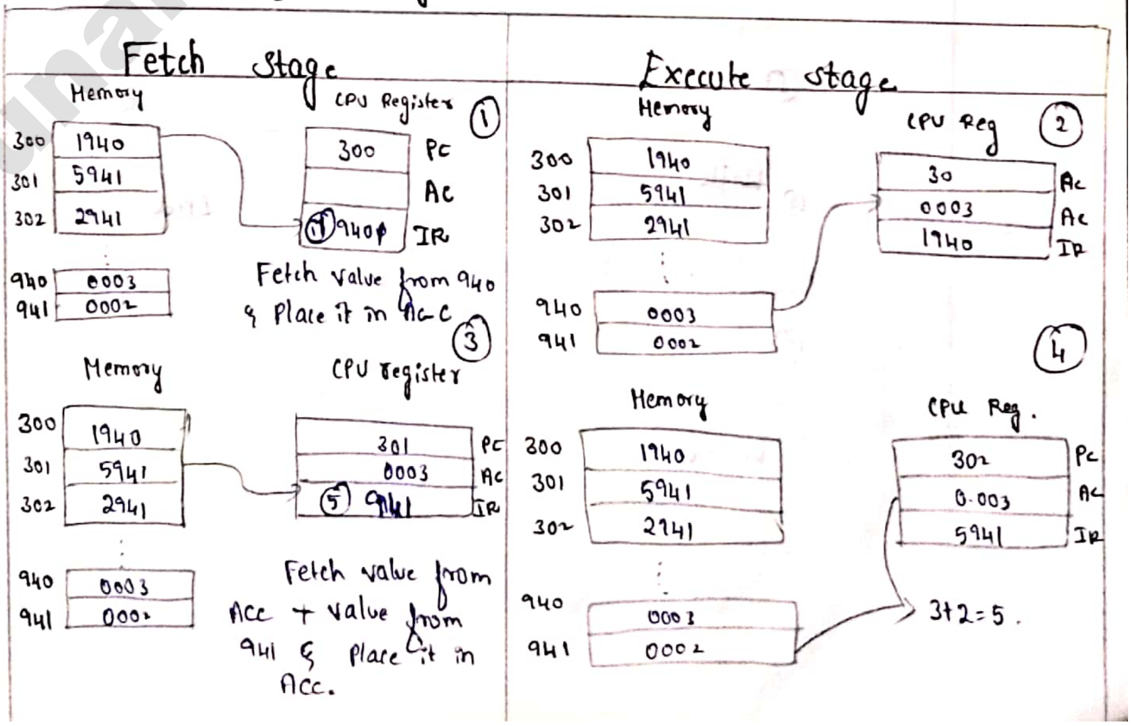
* Pc is implemented after fetching.

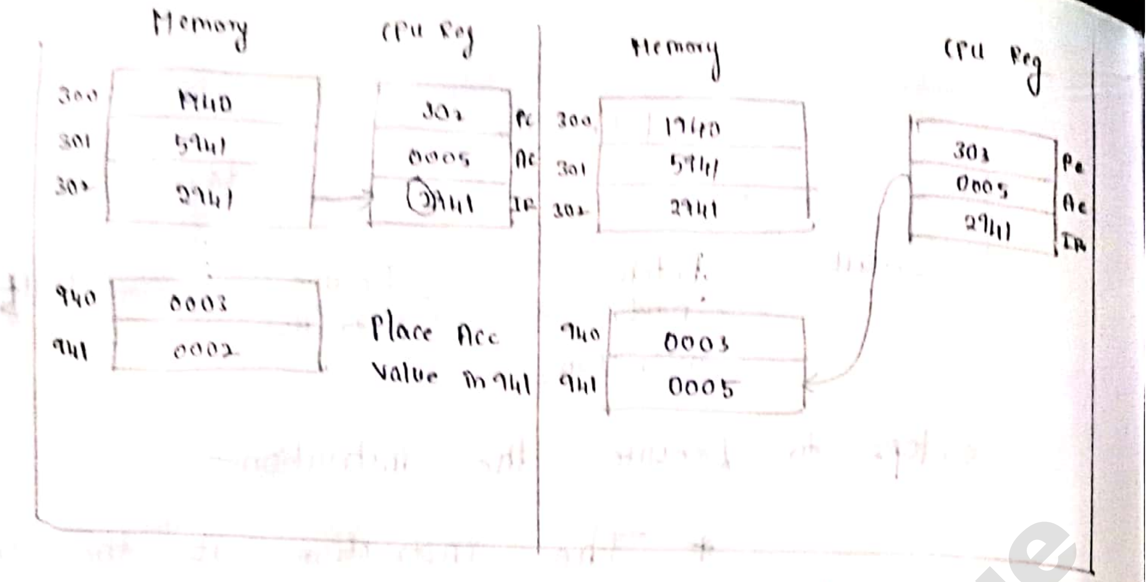
* Processor interpretes the bits stored in IR and perform the required action.

The actions fall into four categories,

(i) Processor – Memory
(ii) Processor – I/o
(iii) Data processing
(iv) Control.

Example of program Execution:-

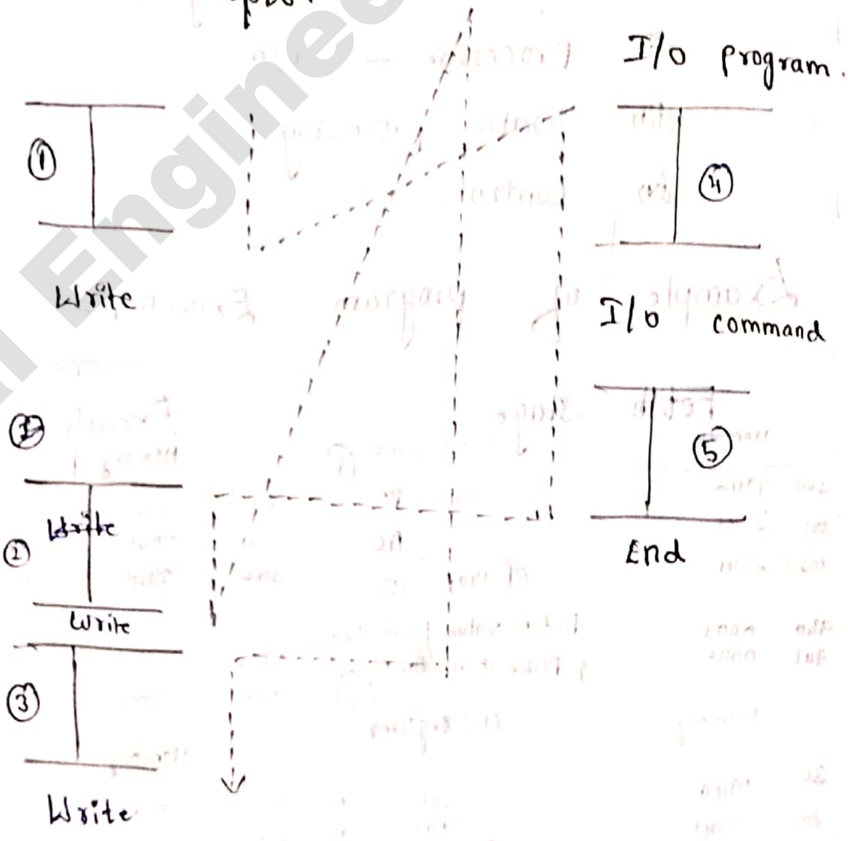




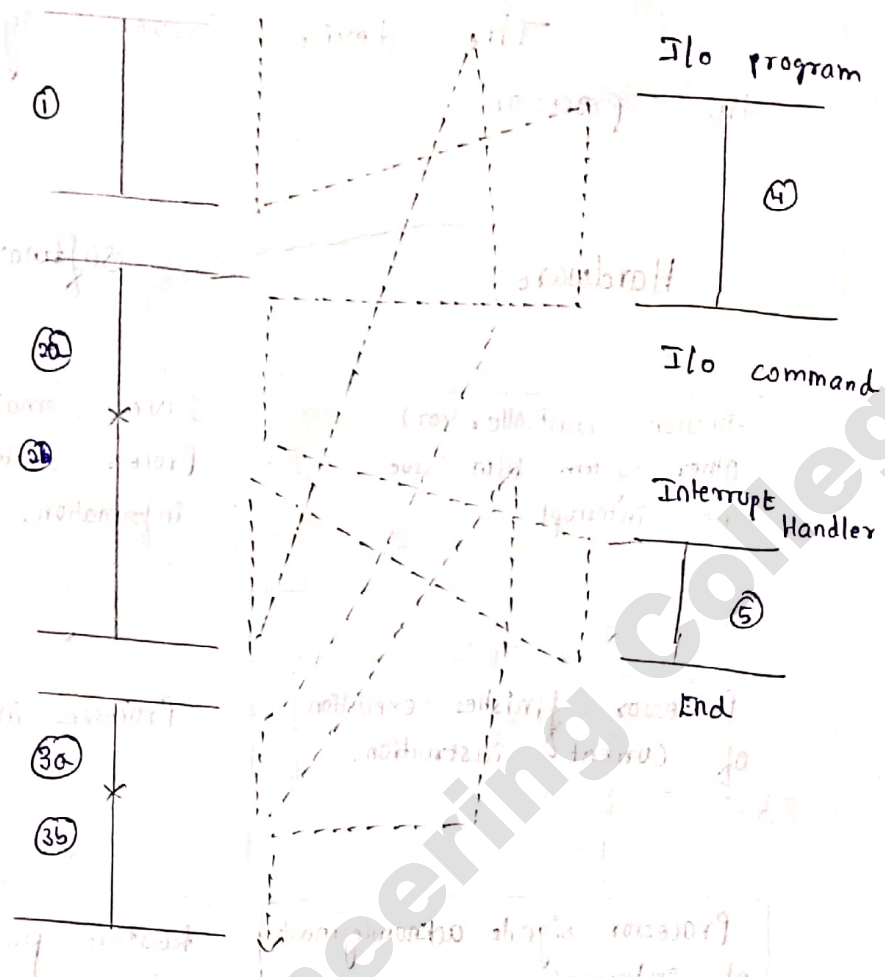
⊛ Interrupt :-

Interrupts are hardware signal that causes the current process to be suspended and control to is transfer to special program called Interrupt handler.

Without Interrupts :-



With Interrupt :-



Classes of Interrupts :-

Program — arithmetic overflow, division by zero, reference outside user allowed memory space.

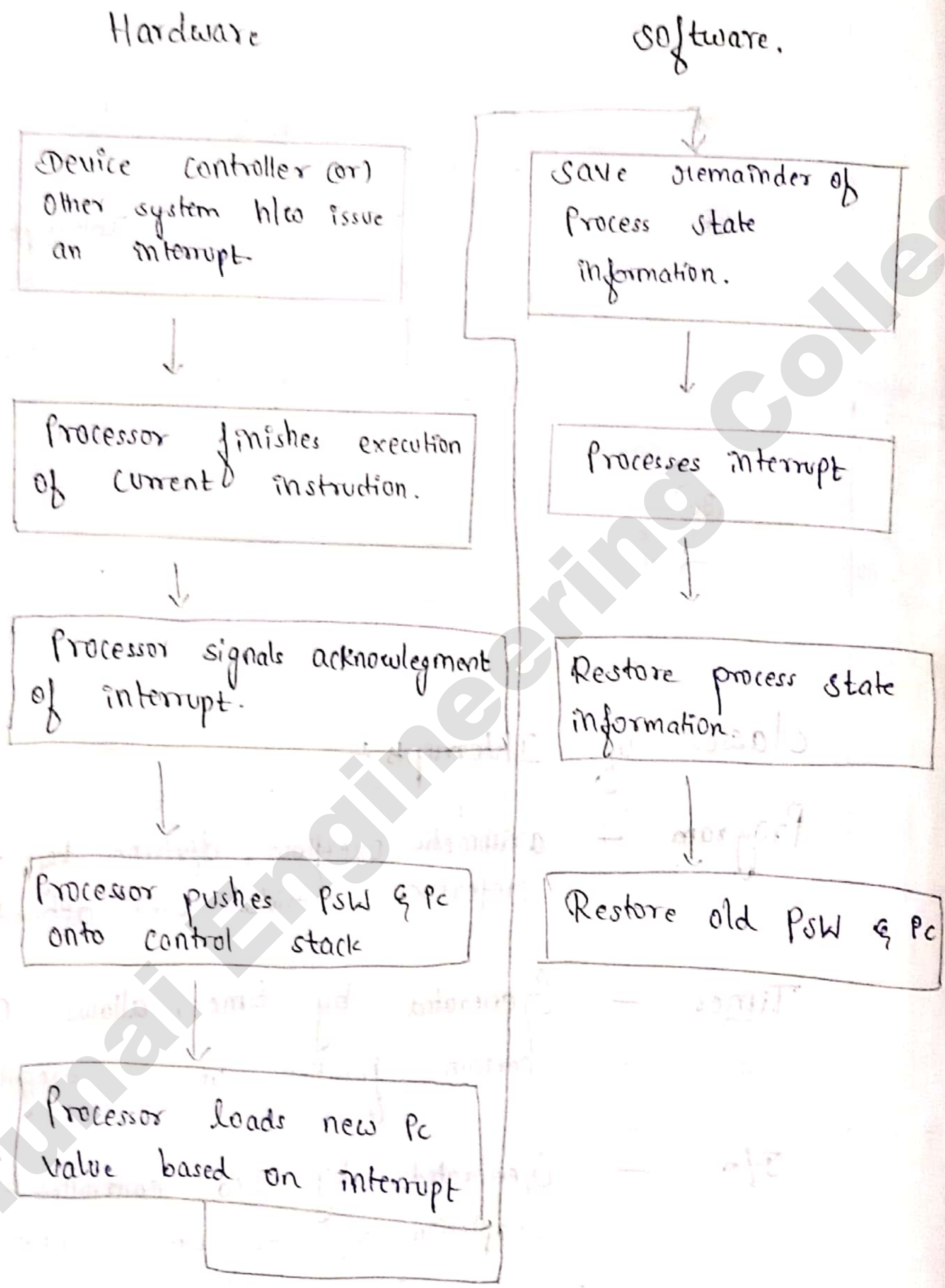
Timer — Generated by timer, allows OS to perform certain function on a regular basis.

I/o — Generated by I/o controller to indicate completion of I/o (or) error condition

H/w failure — Power failure (or) memory parity error.

Steps for Interrupt cycle:-

The device issues signal to the processor.


Hardware
Software.
Device controller (or) other system h/w issue an interrupt
Processor finishes execution of current instruction.
Processor signals acknowlegment of interrupt.
Processor pushes PSW & PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of Process state information.
Processes interrupt
Restore process state information.
Restore old PSW & PC

UNIT-II.

1) Consider the page reference string–

1, 2, 3, 2, 5, 6, 7, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 1

frame size = 4 i) FIFO

1	1	1	1	6	6	6	6
	2	2	2	2	4	4	4
		3	3	3	3	7	7
			5	5	5	5	3

1	1	1	1	2	2	2	2
4	5	5	5	5	3	3	3
7	7	6	6	6	6	5	5
3	3	3	4	4	4	4	1

Total no. of pages fault = 16

2) 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 4, 5, 3.

frame size = 4.

1	1	1	1	5	5	5	5	8	8	8
	2	2	2	2	1	1	1	1	9	9
		3	3	3	3	6	6	6	6	5
			4	4	4	4	7	7	7	7

8	3
9	9
5	5
4	4

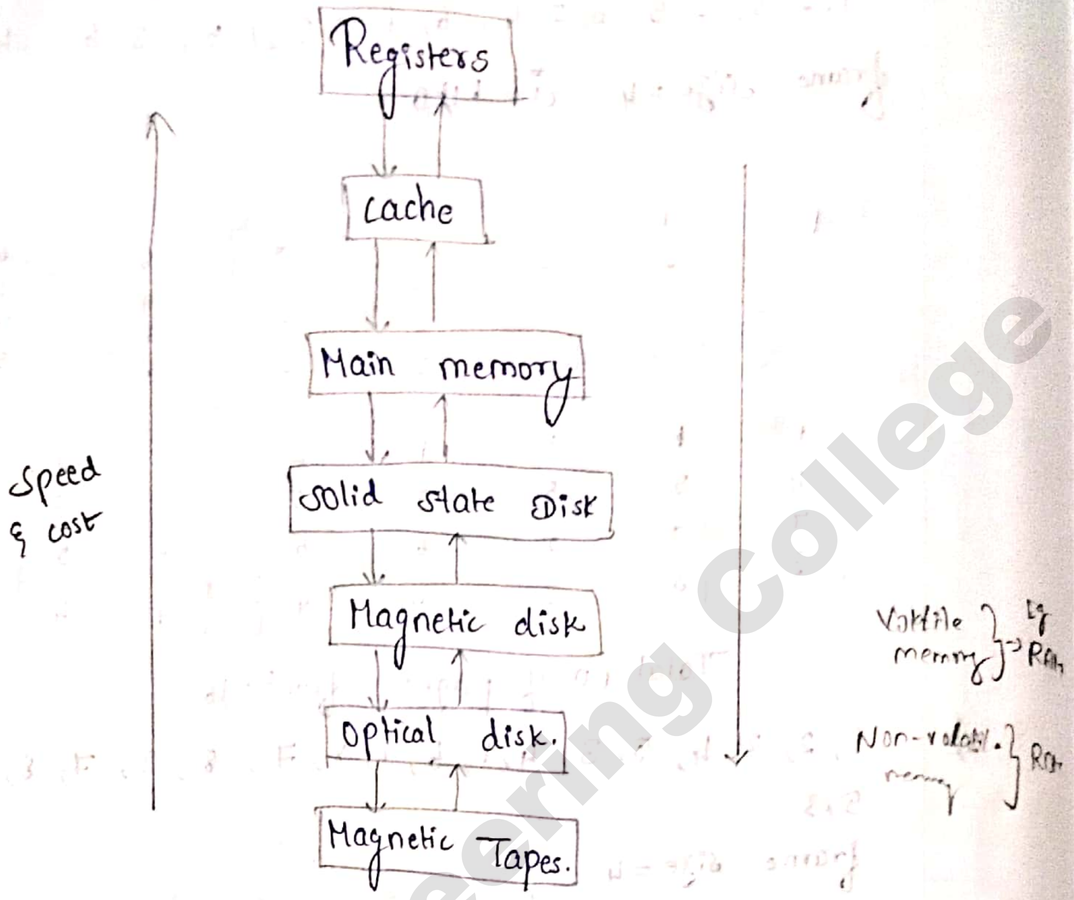
Total no. of pages fault = 13

3) 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Frame size = 3

7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

Total no. of pages fault = 15

Memory Hierarchy :-



Registers :-

It is a temperory storage location inside the CPU that holds data & addresses

Cache :-

Cache memory is a small drop of high speed memory between CPU & main memory

Solid state disk :-

It replaces hard disk by using flash based memory which is non-volatile

Magnetic Disk :-

It is a physical device for storing a data as well as pile the information is kept in sector.

A block is unit of transfer b/w disk and main memory.

Optical disk:-

Electronic data storage medium that can be written to read from using low powdered laser beam.

Magnetic tapes:-

It is a medium for magnetic recording made of thin magnetisable coating on a long narrow strip of plastic fill.

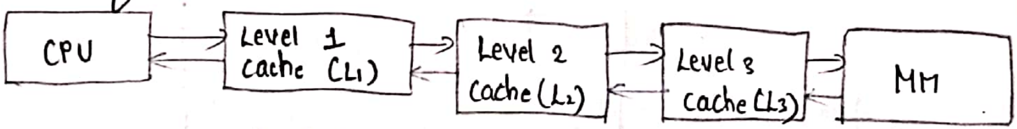
Cache Memory:- & Mapping:-

The small block of high speed memory between main memory & processor

Cache - Miss:- If a reference to a data item which is not present in a cache but it is available in main memory are it as cache miss.

Cache - hit:- If a data item is present in cache memory is called cache-hit.

Level of cache:-



fastest | fast | less fast | slow

- write through cache
- 8 to 64KB

- used to see recent access that is not picked by L1
- 64KB to 2MB

- Present on motherboard
- 3Mb.

Mapping Function:-

Transformation of data from main memory to cache memory is called cache memory mapping.

Types:-

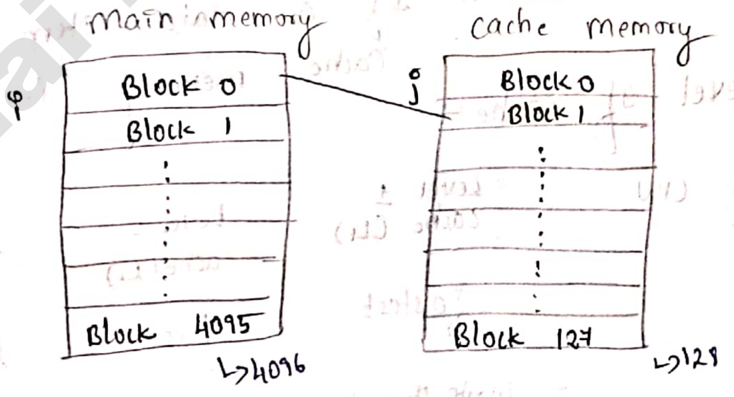
- * Direct Mapping
- * Associative Mapping
- * Set-Associative Mapping

Direct Mapping:-

If i^{th} block of main memory as to be placed at j^{th} block of ~~A~~ cache memory then the mapping is defined as

$$\boxed{j = i \% \text{ No. of blocks in cache memory}}$$

~~Direct Mapping:~~



If block 0 of main memory should be placed in cache memory calculate it by users component

$j = i\%$. no. of blocks in cache memory.

$j = 0\%.\ 128$

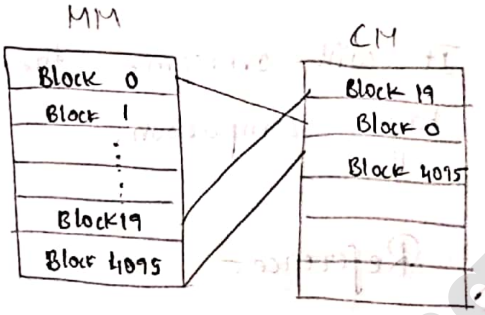
$\boxed{j=0}$ → refers location of cache memory

block no. of MM.

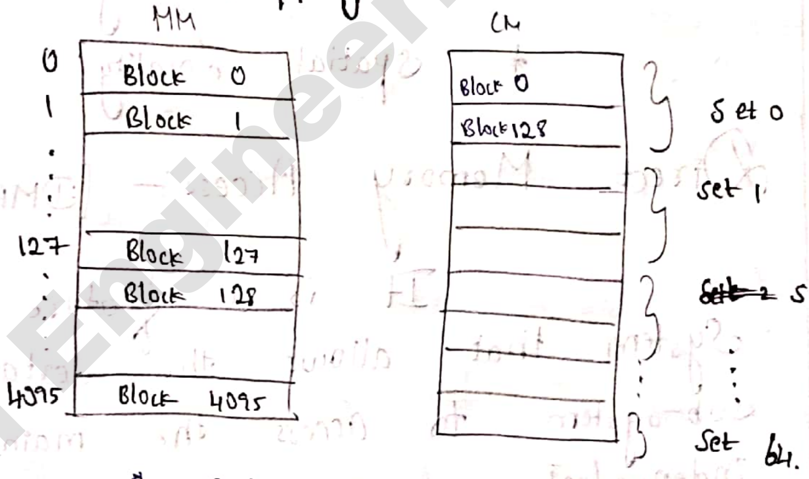
conflict miss occurs when two block numbers of main memory refers to the same address of cache memory.

Associative Mapping:-

Any block of main memory can be placed anywhere within the cache memory. This is the fastest & more flexible mapping technique.



Set Associative Mapping:-



j = i % No.of sets in cache

= 0 % 64

$\boxed{j=0}$ → refers to set of no. of cache memory

j = 128 % 64

$\boxed{j=0}$

Cache blocks are divided into sets, it should be always power of 2.

2 Variations :-

* 2-way set - Associative
* 4-way set - Associative.

2-way set - Associatione → 2 block /set

4-way set - Associative → 4 block /set

It will overcome the high conflictness & large tag comparison.

Locality of Reference :-

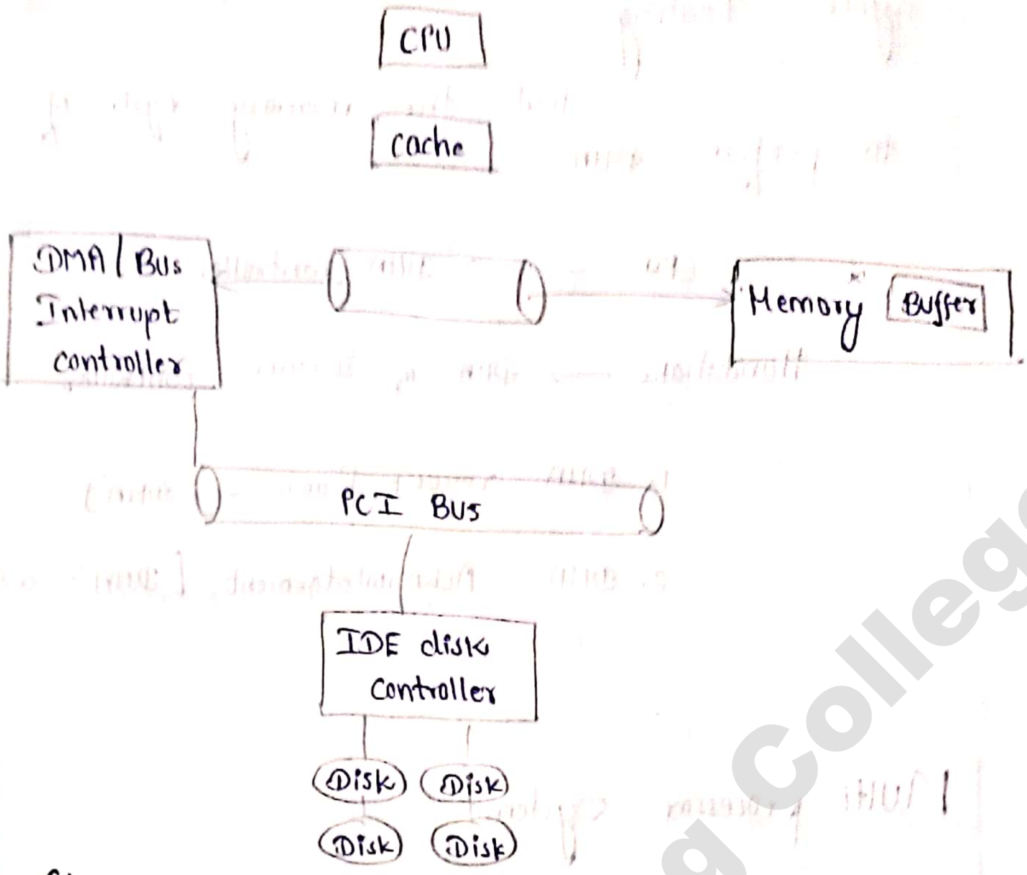
* Temporal Locality
* Spatial Locality

Direct Memory Access :- [DMA]

It is a feature of computer System that allows the certain hardware sub-system to access the main memory independent of CPU.

(or)

A block of data is allowed to transfer between a external device & main memory without continous interruption of CPU.



Steps in DMA Transfer:—

1) Device driver is told to transfer disk data to buffer at address x.

2) DD tells disk controller to transfer c bytes from disk to buffer at address.

3) Disk controller initiates DMA transfer.

4) Disk controller sends each byte to DMA controller.

5) DMA controller transfer bytes to buffer at x. increas memory address & decreasing c, until c=0.

6) when c=0, DMA interrupts CPU to signal transfer completion.

DMA Command block:—

1) Pointer to source of transfer
2) Pointer to destination of transfer
3) No.of bytes to be transferred.

Cycle Stealing:-

Steals the memory cycle of CPU to perform DMA.

CPU ⟶ DMA Controller.

Handshake ⟶ DMA & Device Controller

1, DMA request [Device ⟶ DMA]

2, DMA Acknowledgement [DMA ⟶ Device]

Multi-processor System:-

It is also known as parallel system which have two or more processors which does communication share the computer bus clock & the memory of peripheral devices.

Advantages:-

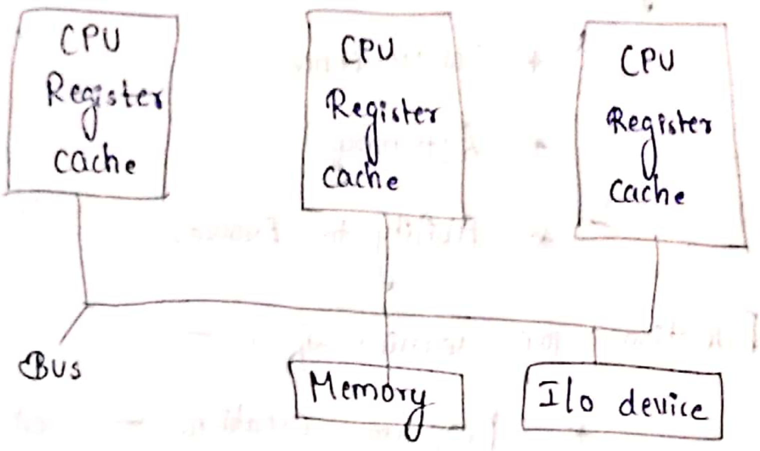
* Increased throughput
* Economy of scale
* Increased Reliability.

Types:-

⇒ Asymmetric [Master / slave]

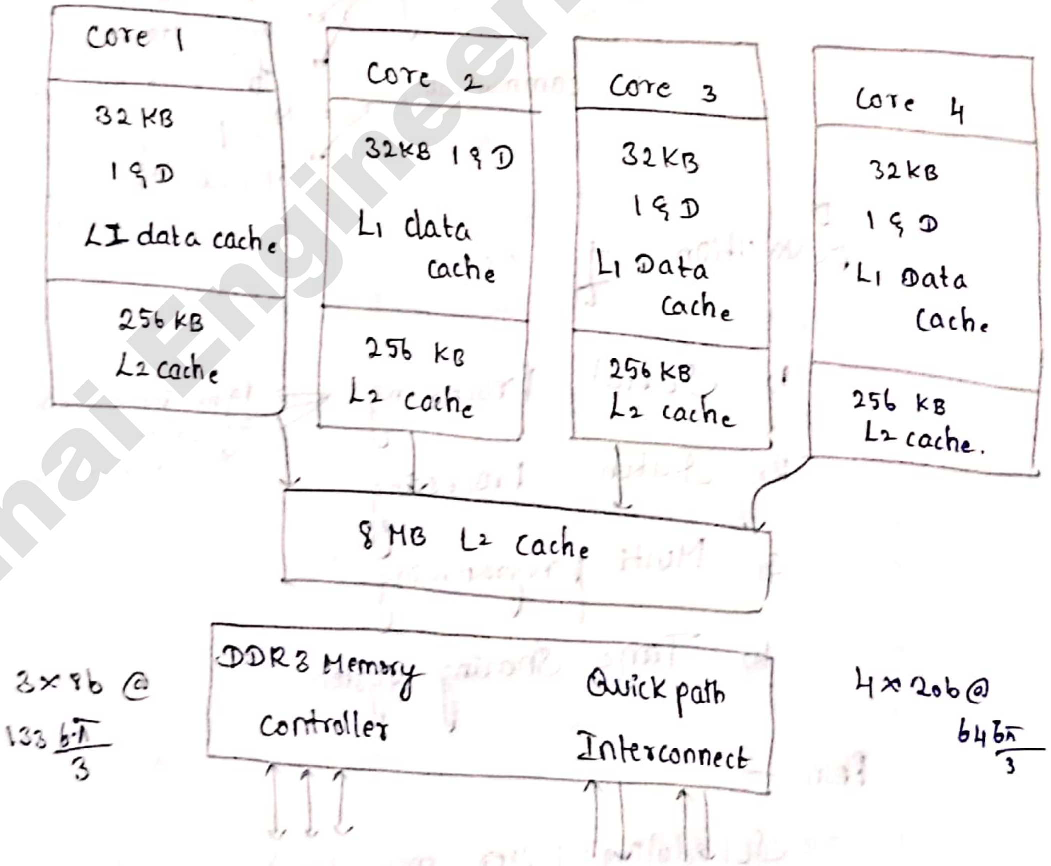
⇒ symmetric [No master / slave]

Symmetric :-



Multicore processor:-

It is processing system composed of two or more independent cores. These cores are integrated into a single integrated circuit die.



Os objectives & Functions:-

* Conventence

* Efficiency

* Ability to Evolve.

Functions (or) services of Os:-

* Program creation. → editors, compiler, debuggers etc..

* Program Execution → Program loaded into memory → necessary Resources.

* Access to I/o devices. → Read / write operation.

* controlled access to files → structure of data.

* System access. → Provides restriction for unauthorised access.

* Error detection & Responce → Power failure → N/w error → odd/even parity.

* Accounting → Response time → Usage statistics → for each resources.

* communication → Msg passing → Shared Memory.

Evolution of Os:-

1) Serial Processing → Pgm in machine code → cards → Errors - lights on → No error - o/p - printer

2) Batch Processing

3) Multi programming

4) Time Sharing systems.

Pblms:-

(Scheduling) (User may finish early / Takes time to complete the job)

(setup time) (Takes time to load compiler, linker etc)

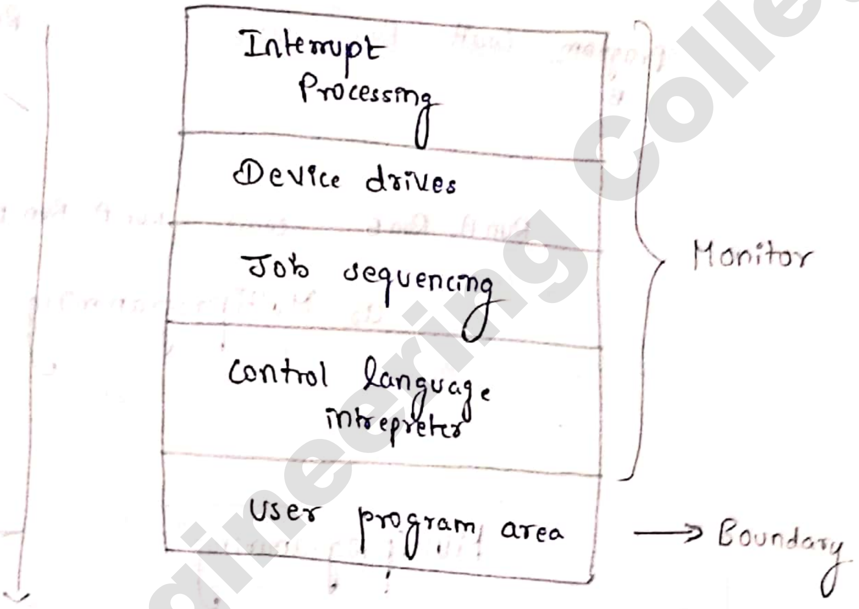
Batch systems:-

- Piece of software → monitor
- No direct access to processor.
- Submits the job on cards to computer operator, who batches the job sequentially (or) places the entire job to an i/p device.

2 views

1) Monitor point

2) processor point



→ Execute instructions from portion of MM.

→ Instructions are written in JCL [Job control language]

Memory protection:-

Monitor → user

Timer:-

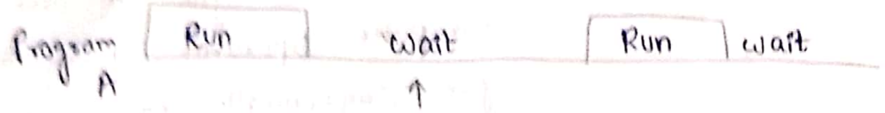
set of beginning of each job.

Privileged instruction:-

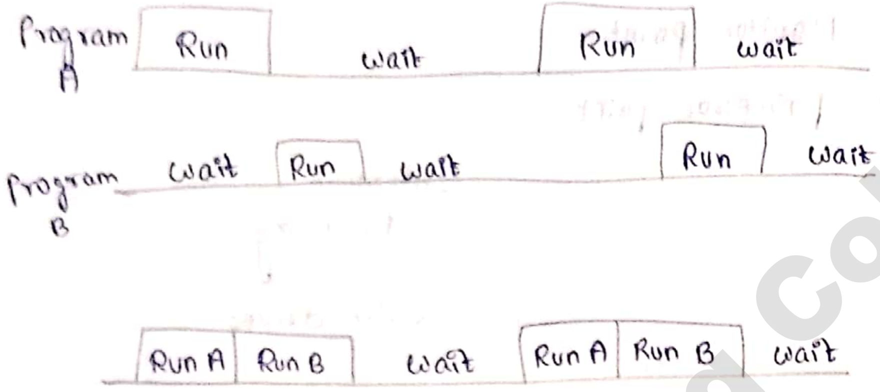
Certain machine instructions are priviledged.

→ if processor encounters it, it transfer control to monitor.

Multi programming:-



a, Uniprogramming.



a, Multiprogramming.

	Multiprogramming	Time sharing
Principal objective	Maximise processor use.	Minimise Response time.
Source of directive to os	JCL commands.	Commands entered at terminal.

Time sharing System:- Multiple users simultaneous access system, with os interleaving execution of each pgm in a short burst (or) quantum.

CTss ⟶ Compatible time sharing system.
↓
MIT by a group machine aided algorithm.

Operating system structure:-

- Simple structure.
- Layer approach.
- Micro kernels.
- Modules.
- Hybrid systems.

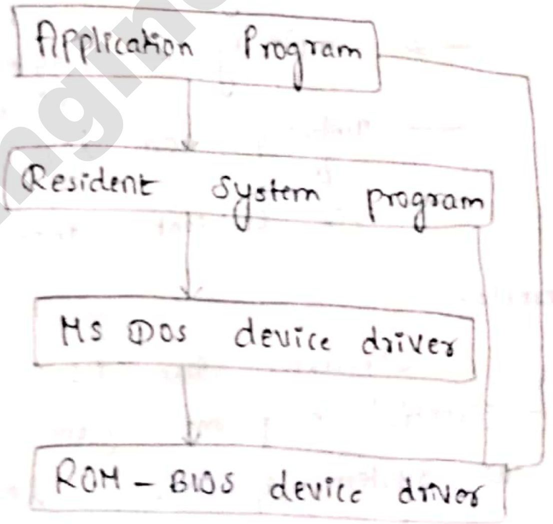
Common approach – partition the task into small modules rather than having monolithic structure.

1) Simple structure:-

=> doesn't have well defined structures.

=> simple, small & limited system.

eg:- MS DOS



Interfaces & levels of functionality are not well separated.

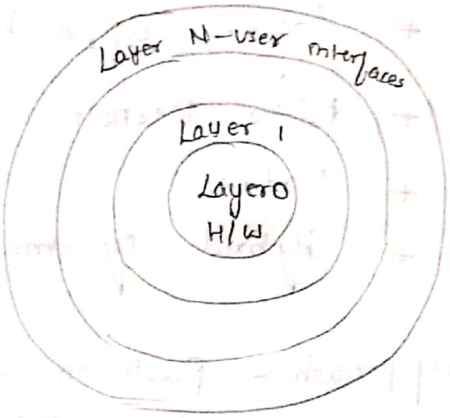
UNIX:-

1) kernel → series of interfaces & device drivers.

2) System Program.

~~Monothi~~ Monolithic structure, difficult to maintain.

Layer Approach :-



→ Information hiding

→ OS → abstract objects.
↓
data & operations that act upon object.

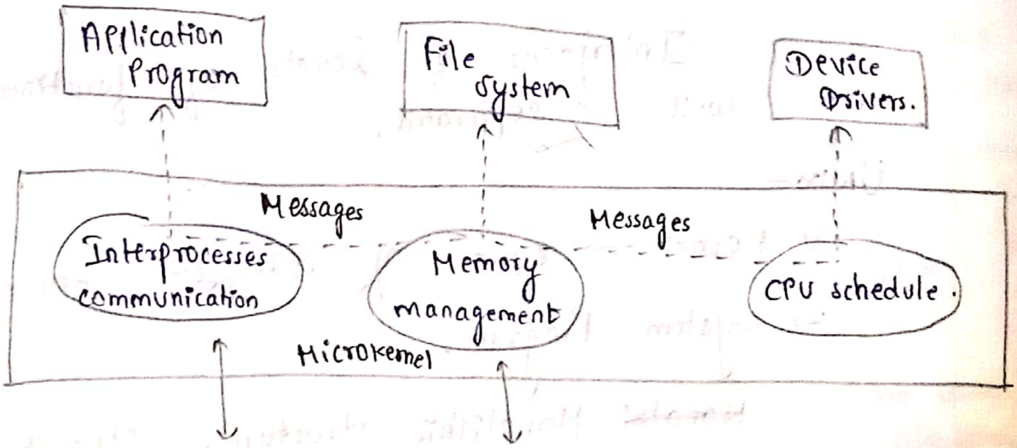
→ Data structures & set of routines invoked by high level layers.

→ Modularity
→ simplicity & debugging } Advantages

→ Defining each layer function
→ less efficient than other } Disadvantages.

MicroKernels :-

Structures the Os by removing all non-essential component, from kernel, & implementing them as system & user level programs.



→ Main function → provide communication by passing messages.

→ client & server never interact directly

→ Benefit < Extending OS is easier

Provide security & reliability since most services are running on user level than kernel

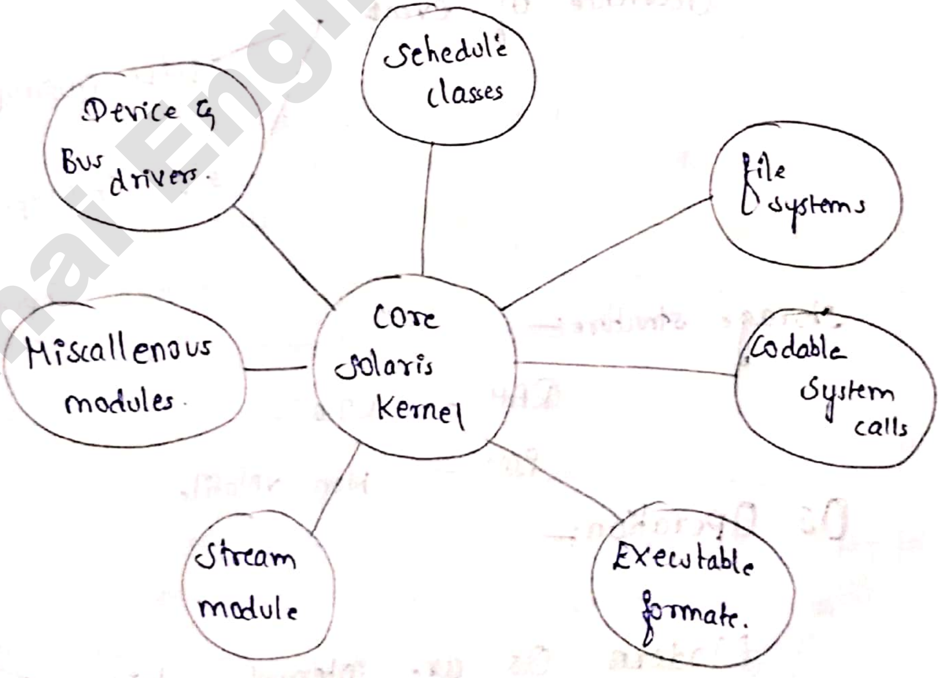
Eg:- Tru 64, Unix

Mac OS X kernel

Modules :-

Kernel has set of core components & links an additional services through modules.

New feature → recoupling every time, a change was made.

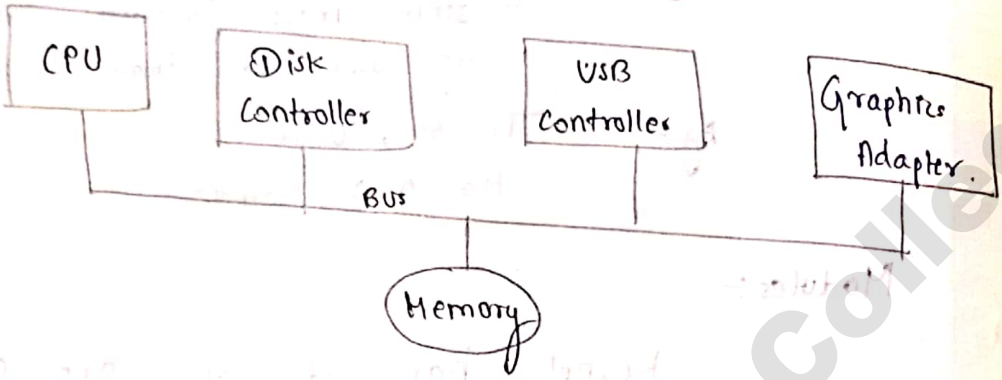


Hybrid systems :-

A combination of all other structure which address performance, security & usability issues.

Eg :- apple IOs, Android

Computer System Architecture :-



when powered on

Boot strap pgm → initial pgm

→ ROM

→ load Os & execute the system.

Occurence of event

- H/w interrupt → signal to CPU.
- s/w interrupt → execute a special operation called "system call"

Storage structure :-

RAM – Volatile

ROM – Non-volatile

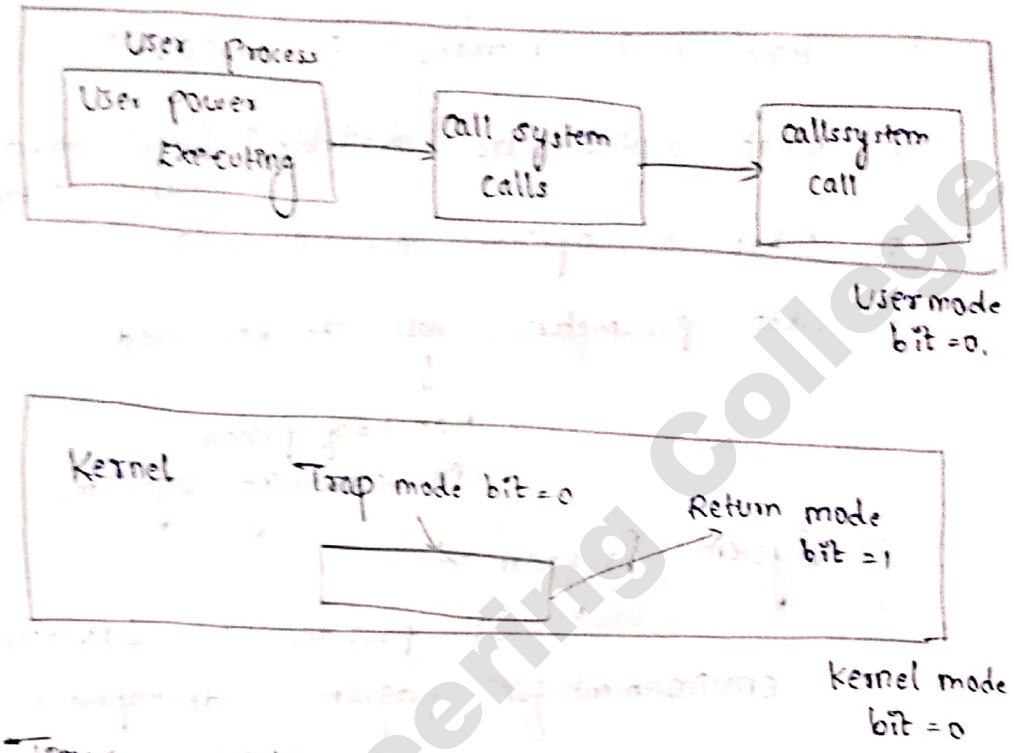
Os Operation :-

Modern Os are interrupt driven

- h/w interrupt
- s/w interrupt

8) Dual Mode & Multimode Operation :-

Mode bit < 0 (kernel) (or) privileged (or) system)
1 (user)



ii) Timer < setup for a specific period of time
Counter

X Sysgen :-

It is possible to design code and implement an OS for one machine at 1 site. The system must be configured are generated for each specific sights. This process is called sysgen.

The OS is distributed hard disk or CD ROM. The OS is available to IOS - image. This sysgen program is read from a file and ask the operator for the information concerning the specific configuration of a system.

Information:-

How will the boot disk be formatted?

How many sectors (or) partitions?

How much memory is available?

what devices are available? [device no, device type, device interrupt. no]

what OS options are derived?

What parameters value to be used?
↓
Max no. of processes.
CPU scheduling algorithm.

System Programs:-

It provides a convenient environment for program development & execution.

1) File Management – create, delete, dump, list & manipulate files & directories.

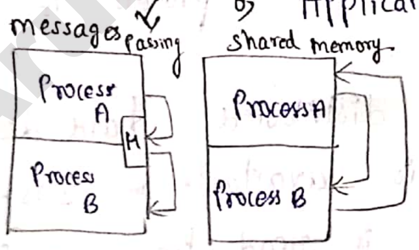
2) Status information → date / time, amount of memory, disk space, no. of users logging, debugging, system configuration.

3) File Modification → Text editors, special commands to search content of file.

4) Programming language support – loaders, Linkers, overlays etc...

5) Communication. → e-mail, Remote login, txt msgs to other windows

6) Application programs (or) System Utilities.
- → Web browsers
- → spreadsheet
- → Database system
- → compilers.

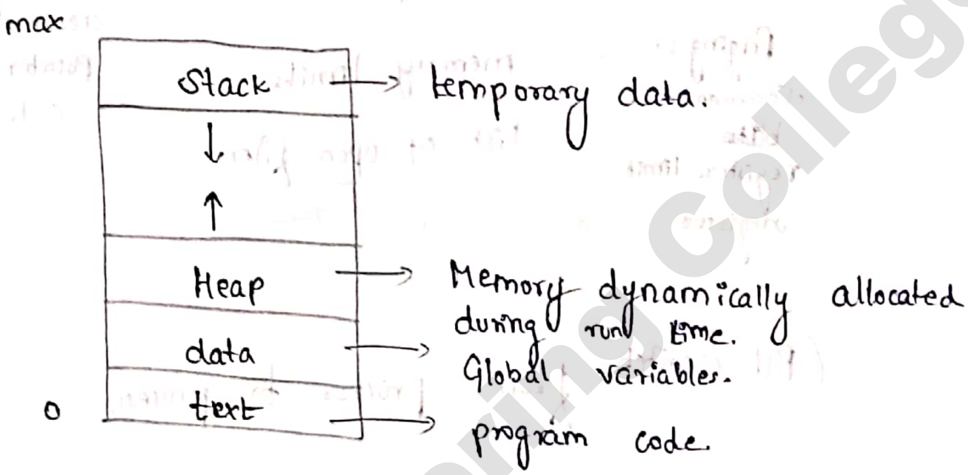


09/01/2020.

UNIT– II.

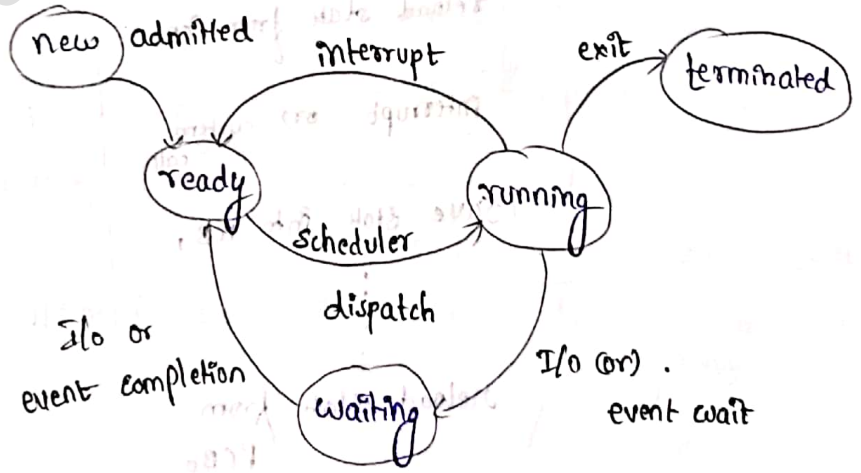
Process Management.

Process → Program in Execution
→ Active entity

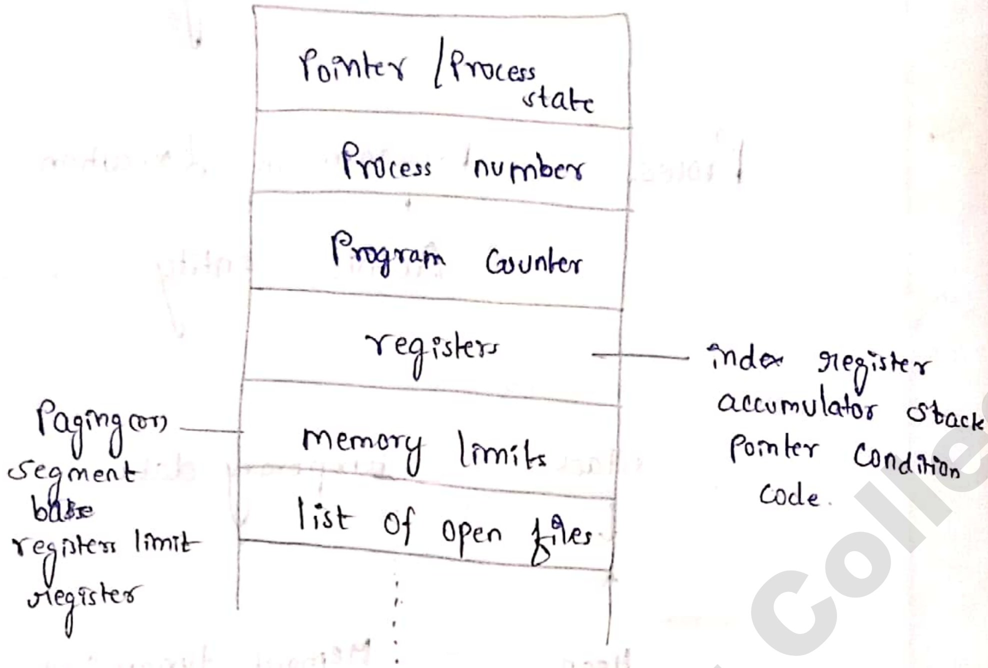


Program → Passive Entity
→ called Executive files.

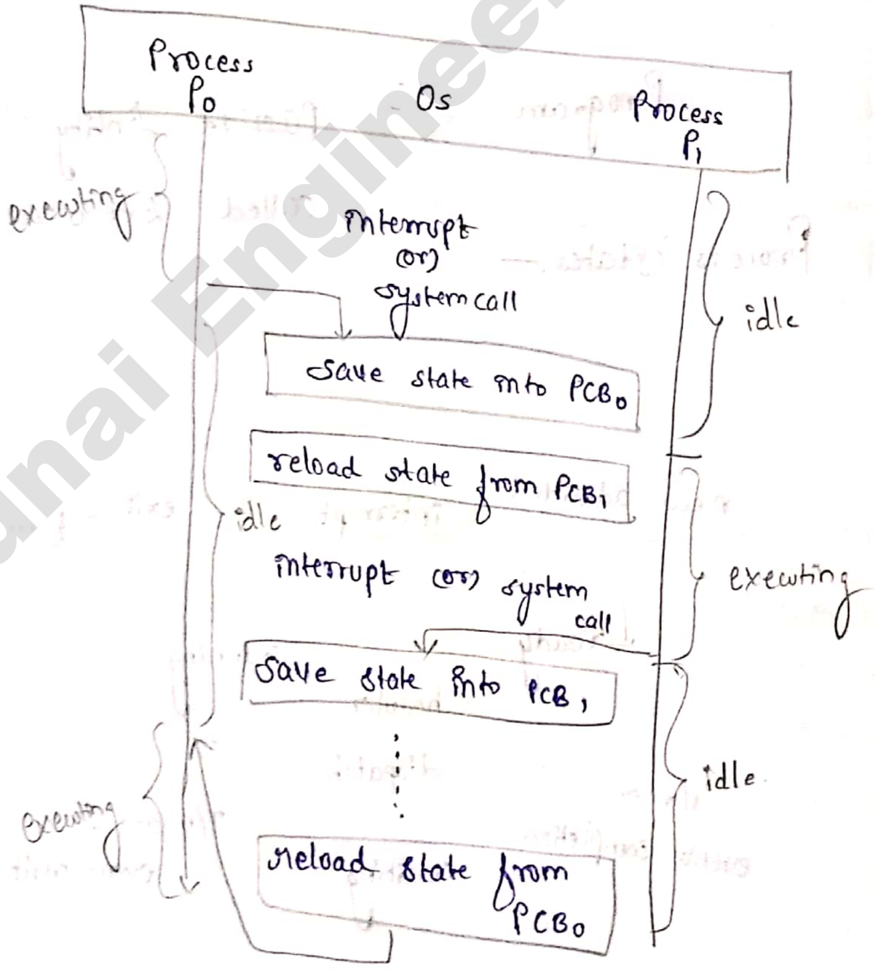
Process States:–



Process control Block:-


Pointer / Process state
Process number
Program Counter
registers
memory limits
list of open files
index register accumulator stack Pointer condition code.
Paging (or) segment base registers limit register

CPU switch from process to process.



Accounting Information:-

i) Amount of CPU and real time used.

ii) Time limits, ~~account number~~

iii) Account number

iv) Job or process number

~~I/o status in~~

I/o status information:-

i) List of I/o devices

ii) List of open files..

~~iii)~~

Process Scheduling:-

- → Scheduling Queues ← Job Queue, Ready ", Device "
- → Schedulers
- → Context switch.

Scheduling Queues:-

(i) Job Queue:-

As the processes enter the system they are put it into a job Queue.

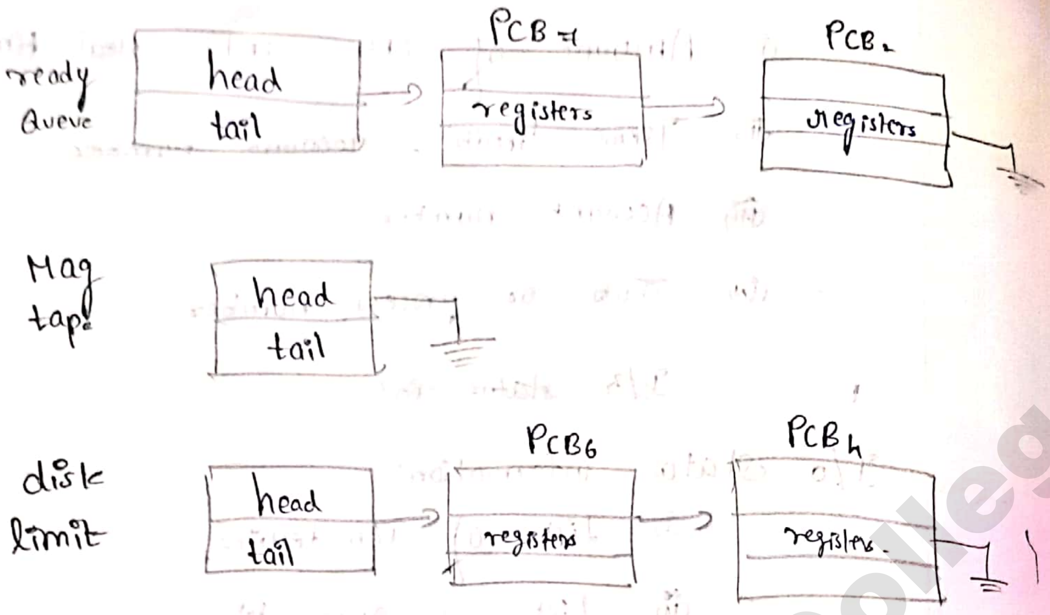
(ii) Ready Queue:-

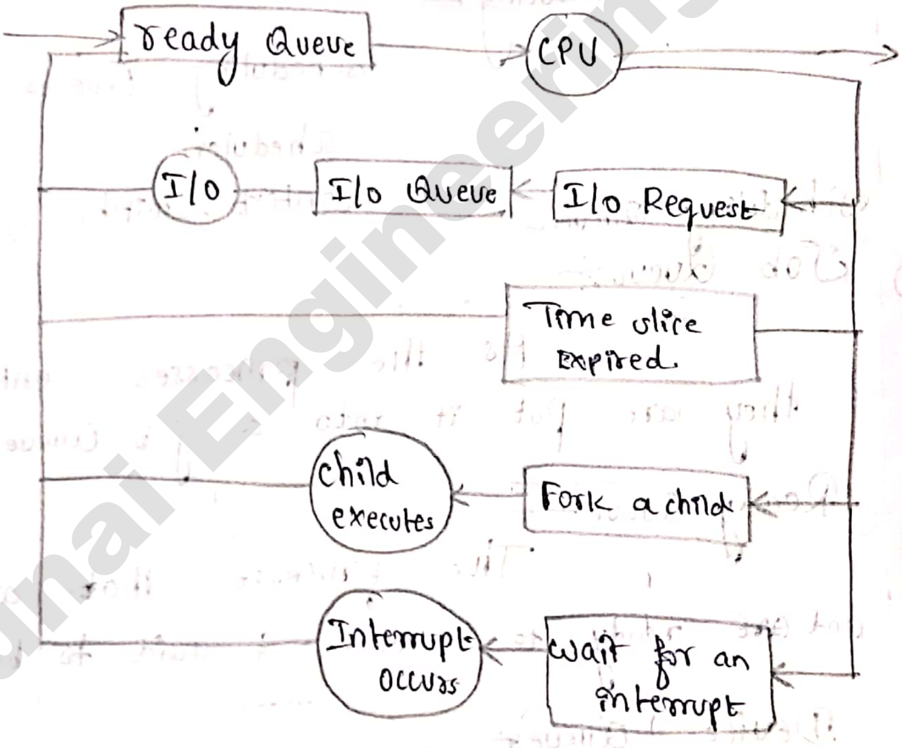
The processes that are receiving and ~~are~~ ready to execute & said to be Ready Queue.

(iii) Device Queue:-

The list of processes wait for a particular i/o device is called device Queue.

Queue header:-





Queue diagram representation of process scheduling.

Schedulers :-

(i) Long term scheduler or Job scheduler

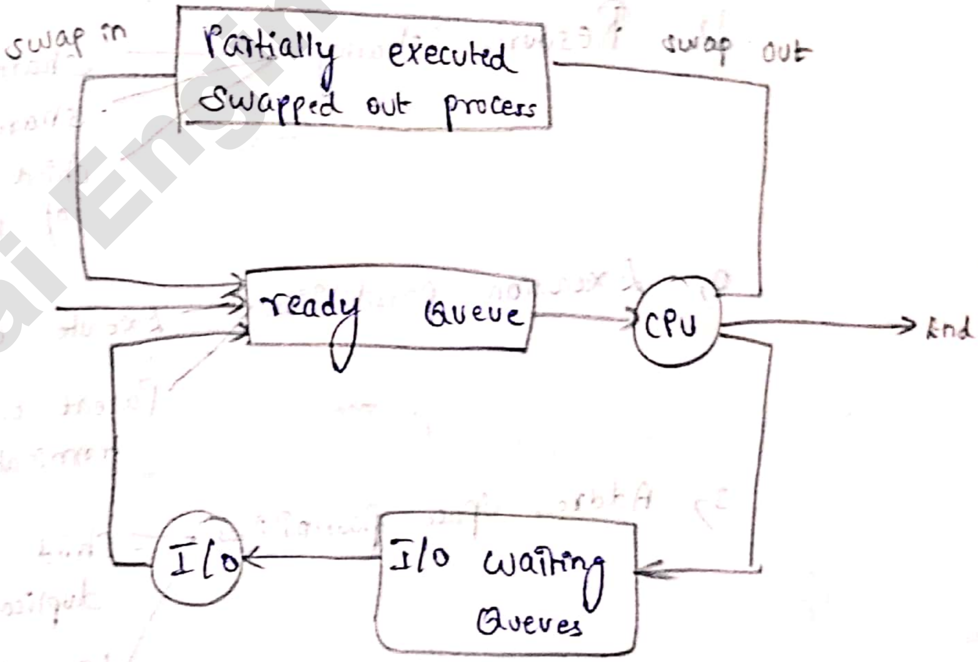
(ii) Short " " or CPU scheduler.

(iii) Medium " " ~~or~~

(i) — Control degree of multiprogramming
— mix ~~ra~~ of I/o & processor — bound.
— Invoked infrequently

(ii) — Invoked frequently.
— Determines which process is going to execute next.

(iii) — Reduce degree of multiprogramming.



Context switch:-

switching the CPU to another process requires to save the state of old process & load the saved state for the new process.

Context switch usually occurs between (1 – 1000 microseconds)

Operation of process:—

Process creation
- create . process system call
- Parent process

Process Termination —— child process → fork

1) Resource sharing
- shares all
- shares nothing
- child shares subset of parent.

2) Execution possibilities
- Execute concurrently
- Parent wait until child terminate.

3) Address space possibilities
- child is a duplicate of parent
- child has a program loaded into it.

fork () → create a child.

exedp () → replace process memory space with new program → Binary file.

wait () → for child completion

exit () → Normal end.

Process Termination :-

exit() → normal completion of execution.

wait () → for its child execution.

abort () → kills the process

→ Parent kills the child.

↓

Reasons :-

— Child exceeds the usage of resources.

— Task assigned to child is no longer needed.

— if parent exits.

Cascading Termination :-

When parent terminates all its child are terminated.

↓

If parent terminate 'init' will be the parent.

Interprocess Communication:-

* Co-operating process
* Independent process

Reason for cooperation:-

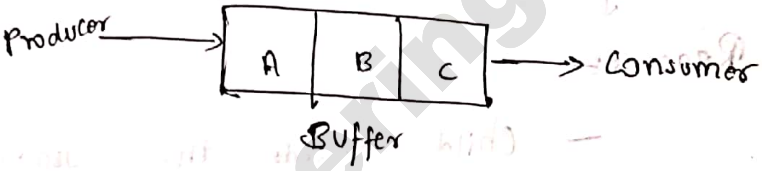
1, Information sharing

2, Computational speedup.

3, Modularity

4, Convenience.

eg:- ~~produce~~ Producer-consumer plan.



Types of Buffer:-

* Unbounded →infinite
* Bounded → fixed

#define Buffer size 5.

typedef struct {.....}

item;

item buffer [Buffer, size];

int in=0;

int out=0;

}

~~while(1)~~

{~~/*~~ ~~produce~~

Producer:-

```
while (1)
{ /* produce an item in next produced
while (((in+1) % Buffersize) == out ; // do
                                        nothing
buffer [in] = next produced;
in = (in+1) % Buffer-size;
}
```

Consumer:-

"in → free slot
out → filled slot"

```
while(1)
{
while (in == out) ; // do nothing
next_consumed = buffer[out];
out = (out+1) % Buffersize ;
}
```

$\boxed{in == out}$ ⇒ Buffer empty.

$\boxed{((in+1) \% Buffersize) == out}$ ⇒ Buffer full

Message passing system:-

IPC:- [Interprocess communication].

It process a machanism to allow the Processes to communicate & to synchronise their actions without sharing the same address space.

Message Passing system:-

Operations ⟨ send (msgs)
Receive (msgs)

Msg size - fixed (or) Variable.

Communication link
- Direct (or) Indirect
- Symmetric (or) Asymmetric
- Automatic (or) Explicit buffering
- fixed size (or) variable sized msgs.

Direct communication :-

Naming:-

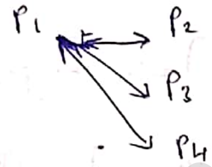
Send (P, msg)

Receive(Q, msg).

Communication link:-

Automatic establishment of link

$P_1 \longleftrightarrow P_2$



Indirect Communication :-

Send (M, msg)

receive (M, msg).

Link:-

=> Established b/w processes that share a Mailbox.

=> Link may be associated with more than 2 processes.

=> No. of different links may exist b/w each pair of processes with each ~~other~~ link corresponds to different mailbox.

Buffering :-

- Zero capacity
- Bounded capacity
- Unbounded capacity.

Synchronisation :-

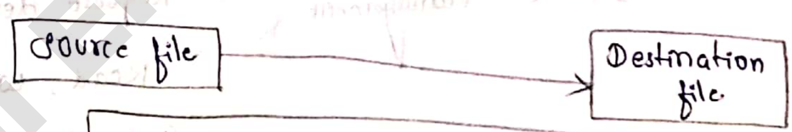
* Blocking send.
* Non-blocking send.
* Blocking receive.
* Non-blocking receive.

UNIT - I

System Call :- Process → OS.

It provides the interface b/w the Process and operating system.

It is generally written in assembly language.



Example system call sequence.

- Acquire input file name.
  Write prompt on screen.
  Accept input.
- Acquire output filenam.
  write prompt on sirren.
  Accept input.
- Open the input file
  if file does exist, abort
- Create o/p file.
  file does exist, abort.
- Loop.
  Read from i/p file
  Write to o/p file

Until read fails.
Close o/p file.
Write completion msg to screen.
Terminate normally..

Types of system call :-

- Process Control.
  - end, abort
  - load, execute
  - create process, terminate proce.
  - Get process attributes, set process attributes
  - wait for time, wait for event, signal
  - Allocate & free memory
- File Management.
- Device Management
- Information Management
- Communication.

File Management
- Create file, delete file.
- Open, close.
- Read, write, Reposition.
- Get file attributes, set file attributes.

Device Management
- Request device, Release devi
- Read, write, reposition
- Get device attributes, set device attributes.
- Logically attach (or) devic devi

Information maintanence :-
- Get time or date, set t or date.
- Get process, file, devic attrib
- set process, file devi attribute

Communication :-

Msg passing → direct or indirect.

Shared memory → access to regions of memory owned by other processes.

System calls :-

→ Create , delete, Communication device.

→ Send / Receive msgs.

→ Transfer status info.

→ Attach (or) detach remote device.

UNIT-2 :-

CPU Scheduling :-

(2m) There are two types of CPU schedular,

* pre-emptive ~~Createtive~~ scheduling — running to ready, waiting to ready

* Non-preemptive scheduling. — Running to waiting, Normal termination.

Scheduling criteria :-

1, CPU utilisation

2, Throughput

3, Turn around time

4, Waiting time

5) Response time

Scheduling Algorithm :-

1) ~~FCFS~~ FCFS.

2) SJF — Preemption, Non-preemption.

shortest job first

3) RR (Round Robin)

4) Priority Scheduling

b) FCFS:-

Process	Burst time.
P_1	24
P_2	3
P_3	3

Arrival order P_1, P_2, P_3.

Soln:- Gantt chart,

P_1	P_2	P_3

0 24 27 30.

W.T => $P_1 = 0$

$P_2 = 24$

$P_3 = 27$.

Avg waiting time } A.W.T $= \frac{0+24+27}{3}$

$= \frac{51}{3}$

$=$ 17 ms.

Turn around time } T.T => $P_1 = 24$

$P_2 = 27$

$P_3 = 30$.

ATT $= \frac{24+27+30}{3}$

$= \frac{81}{3}$

$=$ 27 ms

P_1, P_3, P_2

P_1	P_3	P_2

0 24 27 30.

$$A.W.T = \frac{0+24+27}{3}$$

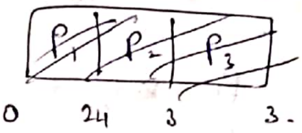
$$= \frac{51}{3}$$

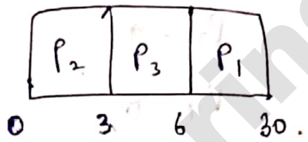
$= 17ms.$

2)

Process	Burst time	A.T
P_1	24	2.0
P_2	3	0.0
P_3	3	1.0

Gantt chart :-





$W.T \Rightarrow P_1 = 6-2 = 4$

$P_2 = 0-0 = 0$

$P_3 = 3-1 = 2.$

$$AWT \Rightarrow \cancel{4} = \frac{4+0+2}{3} = \frac{6}{3} = 2ms$$

$T.T \Rightarrow P_1 = 30-2 = 28.$

$P_2 = 3-0 = 3$

$P_3 = 6-1 = 5$

$$ATT = \frac{28+3+5}{3}$$

$$= \frac{36}{3} = 12ms.$$

Shortest Job First (SJF) :- (Non - Preemptive)

Process	B.Time
P_1	6
P_2	8
P_3	7
P_4	3

Gantt chart :-

P_4	P_1	P_3	P_2

0 3 9 16 24

P_1 = ~~0~~3

P_2 = 16

P_3 = 9

P_4 = 0.

$$AWT = \frac{3+16+9+0}{4}$$

= 7 ms

TT => P_1 = 9

P_2 = 24

P_3 = 16

P_4 = 3

$$TT \Rightarrow \frac{9+24+16+3}{4}$$

= 13 ms.

Process	B.T.
P_1	2
P_2	1
P_3	8
P_4	4
P_5	5

P2	P1	P4	P5	P3

0 1 3 7 12 20

$$AWT = \frac{0+1+3+7+12}{5}$$

$= 4.6ms$

$$TT = \frac{1+3+7+12+20}{5}$$

$= 8.6 ms.$

Non-preemptive SJF with arrival time:-

Process	BT	AT
P_1	6	0
P_2	8	1
P_3	7	2
P_4	3	3

Gantt chart,

P_1	P_4	P_3	P_2

0 6 9 16 24

WT ⇒ $P_1 = 0-0=0$

$P_2 = 16-1 = 15$

$P_3 = 9-2 = 7$

$P_4 = 6-3 = 3.$

$$AWT = \frac{0+15+7+3}{4}$$

$= 6.25 ms.$

TT = $P_1 = 6-0 = 6$

$P_2 = 24-1 = 23$

$P_3 = 16-2 = 14$

$P_4 = 9-3 = 6$

~~AT~~

$$ATT = \frac{6+23+14+6}{4}$$

$= 12.25ms.$

SJF [Preemption] :-

Process	BT	AT
P_1	8 ~~7~~	0
P_2	4 ~~3~~ 2	1
P_3	4	2
P_4	5 3	3

8 7
4 3 2
4
5

Gantt chart :-

P_1	P_2	P_2	P_2	P_4	P_1	P_3

0 1 2 3 5 10 17 26

W.T :-

P_1 = FT − BT − AT

= 17 − 8 − 0

= 9.

P_2 = 5 − 4 − 1

= 0

P_3 = 17 − 2

= 15

P_4 = 5 − 3

= 2.

$AWT = \frac{9+0+15+2}{4}$

= 6.5 ms.

TT ⇒ P_1 = 17 − 0 = 17

P_2 = 5 − 1 = 4

P_3 = 26 − 2 = 24

P_4 = 10 − 3 = 7

$ATT \Rightarrow \frac{17+4+24+7}{4}$

= 13 ms

2)

Process	BT	AT
P_1	~~7~~ 5	0.0
P_2	~~4~~ 2	2.0
P_3	~~1~~ 0	4.0
P_4	4	5.0

Gantt chart,

	P_1	P_2	P_3	P_2	P_4	P_1
0	2	4	5	7	11	16

W.T :-

P_1 = FT − BT − AT.

= 16 − 7 − 0.0

= 9

P_2 = 7 − 4 − 2

= 1

P_3 = 4 − 4

= 0

P_4 = 7 − 5

= 2

$$AWT = \frac{9+1+0+2}{4} = 3ms$$

TT :-

P_1 ⇒ 16 − 0 = 16

P_2 = 7 − 2 = 5

P_3 = 5 − 4 = 1

P_4 = 11 − 5 = 6

$$ATT = \frac{16+5+1+6}{4}$$

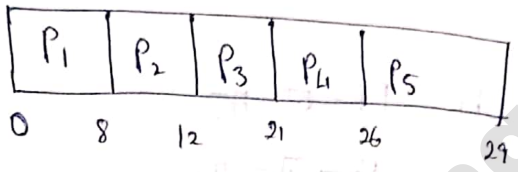
= 7ms.

Process	BT	AT.	
P_1	8	0.0	∅
P_2	4	1.001	1
P_3	9	2.001	2
P_4	5	3.001	3
P_5	3	4.001	4

[FCFS, SJF(NP), SJF(P)].

FCFS:—

Gantt chart:—



W.T ⇒

$P_1 = 0 - 0.0 = 0$

$P_2 = 8 - 1.001 =$ ~~7~~ 6.999

$P_3 = 12 - 2.001 = 9.999$

$P_4 = 21 - 3.001 = 17.999$

$P_5 = 26 - 4.001 = 21.999.$

~~AT~~ AWT $= \dfrac{0 + 6.999 + 9.999 + 17.999 + 21.999}{5}$

$= 11.3992$ ms.

TT ⇒

$P_1 = 8 - 0.0 = 8$

$P_2 = 12 - 1.001 = 10.999$

$P_3 = 21 - 2.001 = 18.999$

$P_4 = 26 - 3.001 = 22.999$

$P_5 = 29 - 4.001 = 24.999$

ATT $= \dfrac{8 + 10.999 + 18.999 + 22.999 + 24.999}{5}$

$= 17.1992$ms.

SJF [Non-preemptive].

Gantt chart:-

P1	P5	P2	P4	P3

0 8 11 15 20 27.

W.T => $P_1 = 0 - 0.0 = 0$

$P_2 = 11 - 1.001 = 9.999$

$P_3 = 20 - 2.001 = 17.999$

$P_4 = 15 - 3.001 = 11.999$

$P_5 = 8 - 4.001 = 3.999$

$$AWT = \frac{0 + 9.999 + 17.999 + 11.999 + 3.999}{5}$$

$$= \frac{43.996}{5} = 8.799 \text{ ms.}$$

TT => $P_1 = 8 - 0.0 = 8$

$P_2 = 15 - 1.001 = 13.999$

$P_3 = 27 - 2.001 = 26.999$

$P_4 = 20 - 3.001 = 16.999$

$P_5 = 11 - 4.001 = 6.999$.

$$ATT = \frac{8 + 13.999 + 26.999 + 16.999 + 6.999}{5}$$

$$= \frac{72.996}{5}$$

$$= 14.5992 \text{ ms.}$$

SJF [Preemption].

Gantt chart:-

P1	P2	P2	P2	P2	P5	P4	P1	P3

0 1 2 3 4 5 8 13 20 27

P1	~~8~~ 7	0
P2	4 ~~3~~ ~~2~~ 1	1
P3	9	2
P4	5	3
P5	3	4

W.T => $P_1 = FT - BT - AT$

$= 20-8-0$

$= 12$

$P_2 = 5-4-1$

$= 0$

$P_3 = 20-2$

$= 18$

$P_4 = 8-3$

$= 5$

$P_5 = 5-4$

$= 1$

$AWT = \frac{12+0+18+5+1}{5}$

$= 7.2$ms

T.T => $P_1 = 20-0 = 20$

$P_2 = 5-1 = 4$

$P_3 = 29-2 = 27$

$P_4 = 13-3 = 10$

$P_5 = 8-4 = 4$

$ATT = \frac{20+4+27+10+4}{5}$

$= 13$ms.

Preemptive Priority :-

Process	AT	Priority	B.T.
P_1	0	3	~~8~~ ~~7~~ ~~5~~ ~~4~~ 3
P_2	1	4	2
P_3	3	4	4
P_4	4	5	1
P_5	5	2	~~6~~ ~~5~~ 1
P_6	6	6	5
P_7	10	1	1

P_1	P_1	P_1	P_1	P_5	P_5	P_7	P_5	P_1	P_2	P_3	P_4	P_6

0 1 3 4 5 6 10 11 12 15 17 21 22 27.

W.T ⟹ P_1 = FT − BT − AT

= 15 − 8 − 0

= 7

P_2 = 15 − 1

= 14

P_3 = 17 − 3

= 14

P_4 = 21 − 4

= 17

P_5 = 12 − 5 − 6

= ~~7~~ = ~~6~~ = 1

P_6 = 22 − 6

= 16

P_7 = 10 − 10

= 0

$$AWT = \frac{7+14+14+17+16+1+}{7}$$

~~= 10.57 ms.~~

= 9.85 ms.

TT => $P_1 = 15 - 0 = 15$

$P_2 = 17 - 1 = 16$

$P_3 = 21 - 3 = 18$

$P_4 = 22 - 4 = 18$

$P_5 = 12 - 5 = 7$

$P_6 = 27 - 6 = 21$

$P_7 = 11 - 10 = 1$

$$ATT = \frac{15+16+18+18+7+21+1}{7}$$

$= 13.71 ms.$

Multilevel Queue Scheduling :-

→ Partition ready Queue into separate Queue

1, Foreground (interactive) – RR

2, Background (Batch) – FCFs.

→ processes are permanently assigned to each Queue based on some property.

Scheduling done b/w Queues :-

Fixed Priority Scheduling → Possibility of ~~Saturation~~ starvation

Time slice => 80% to FP.

20% to bP.

Eg :- 5 Queues :-

1, System Processes

2, Interactive processes

3, Interative editing processes.

4, Batch processes.

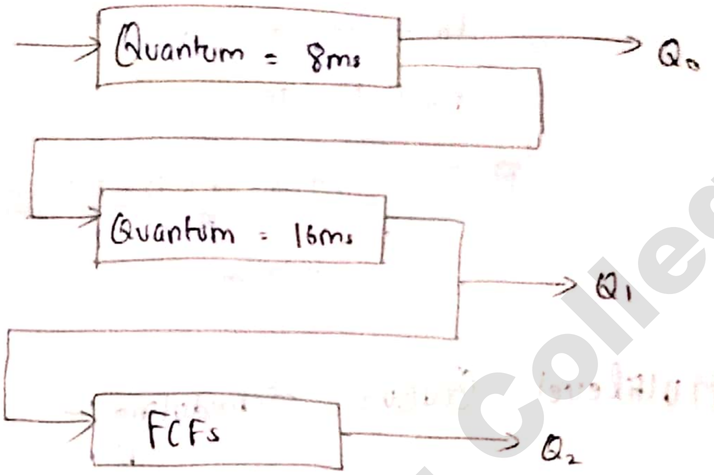
5) Student Processes.

Multilevel feedback Queue:-

⇒ Allows proces to move b/w Queues

⇒ Aging ⟶ to solve starvation

Eg:-



Parameters Required:-

1) No. of Queues.

2) Scheduling algorithm for each Queue.

3) Methods Used to when to upgrade a process.

4) When to demote a process.

5) which queue the process should enter when it requires service.

Multiple process scheduling :-

Self - scheduled [symmetric multiprocessing]

↳ common (or) separate ready queue.

Master slave [Assymmetric Multiprocessing]

↓ ↳ Executes only user code.

→ Scheduling decision

→ I/o processing

Real time scheduling :- Air traffic, Telecommunications, Robotics

Hard real time systems → complete a task within a time limit.

soft " " " → critical process receive priority than others,

Types :-

Periodic Schedulers. → fixed Arrival time.

Demand driven " → variable " "

~~Deadline~~ Deadline " → priority determined by deadline.

Issues in Realtime scheduling :-

1) Dispatch latency.

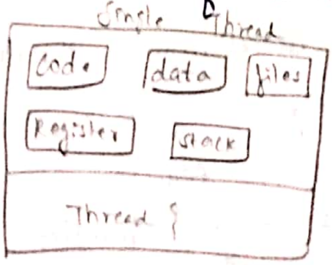
2) priority inversion & Inheritance.

Threads :- [basic unit of CPU utilisation]

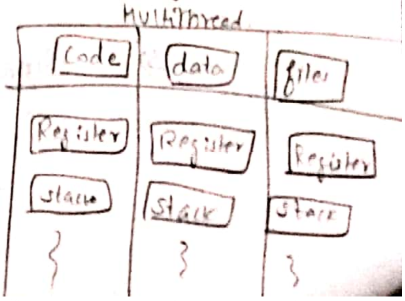
Light weight process.

consists of thread ID, stack, register, code.

Single Thread



Multithread



(eg) - web server application,

Benefits :-

- * Responsiveness
- * Resource sharing
- * Economy
- * Utilisation of multicore architectures.

User Threads :-

Kernel " :-

Multithreading models :-

- * One to one [one user level thread] → one Kernel thread
- * Many to many [Multiple many user level thread to many Kernel thread]
- * One to many [maps one user level threads to many Kernel threads]

Threading Issues :-

1) Fork & exec system call.

- Exec immediately after fork.
  ↓
  duplicates the thread that invoke it
- Fork called later
  ↓
  duplicates all threads.

2) Cancellation

- Asynchronous - one thread cancel the other.

Defered cancellation - thread cancel by itself.

3) Signal handling < Sychronous (eg) illegal memory access
Asynchronous (eg) terminate a process.

- Default signal handler.
- User defined signal handler.

4) Thread pools:-

- Amt of time required to create a thread
- terminate after its service
- No bound on no. of threads running concurrently

Process Synchronisation:-

⇒ Concurrent access to shared data results in data inconsistency.

↓

⇒ ensure order of execution of co-operating processes.

Race condition:-

This situation arises where several processes access and manipulate the shared data concurrently, the final value of " " depends upon on which process finishes last.

Critical section problem:-

Each process as a segment of code called critical section in which a process may be changing the common variable updating a table writing a file and sold.

Requirement to be satisfied for a soln to critical section problem:-

⇒ Mutual Exclusion.

⇒ Progress.

⇒ Bounded waiting.

General structure of process Pi

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

Two process solution:-

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

Alg-2:-

```
do
{
flag[i] = true;
while (flag[j]);
    CS
flag[i] = false;
    RS
} while(1);
```

Alg-3:-

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

do
{
flag[i] = true;
```

```
        turn = j;
        while (Flag [j] && turn == j);
              cs
        flag [i] = false;
              Rs.
        } while (1);
```

Synchronisation H/w.

H/w support for cs structure of process pi:-

```
repeat
      disable interrupts
            cs
      Enable interrupts
            Rs.
forever!
```

(i) Test & set synchronisation H/w.

shared, boolean lock = false;

structure of process pi

```
do
{
 while (Test & set(lock));
        cs
    lock = false ;
      Rs
   } while(1);
boolean Test & set (Boolean * Target).
{
  boolean rv = *target;
  * target = true ;
} return rv;
```

Mutual Exclution with swap :-

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

Structure of process Pi

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

Process can enter into CS while lock = true & Key = false

Mutex:-

Acquire:-

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

Release:-

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

2m) Busy wait:-

A process repeatedly check to see if a lock is available.

Semaphore:-

It is a synchronisation tool that doesn't require busy wait.

It is a variable that as an integer value which may be initialised to non-negative integer.

wait (P) = "to test" => decrement semaphore value.
signal (v) = "to increment" => increment " "

```
wait(s)
{
 while (s≤0) do;
   s--;
}
signal(s)
{
 s++;
}
```

Structure of process pi:-

```
do
{
 wait (mutex);
  cs
 signal (mutex);
  Rs
} while(1);
```

Synchronisation of 2 process:-
statement in P1:-

```
S1;
signal (mutex);
```

Statement in P_1:-

```
        wait(mutex);
        S2;
```

spinlock:-

Process spins while waiting for a lock.

Semaphore implementation:-

```
        typedef struct
        {
          int value;
          struct process *c;
        } semaphore;
```

2 operations:-

```
        block()
        wakeup()
```

Semaphore operations:-

```
      void wait(Semphore s)
      {
        s.value --;
        if(s.value <0)
      {
        add this process to s.L,
        block();
       }
      }
```

Process 'block' itself & enters into waiting Queue, if semaphore value is '-ve'

```
     void signal (semphore s)
     {
      s.Value ++;
      if(s.value <= 0)
     {
      remove a process p from s.L;   -semaphore list
      wakeup(p);
     }
     }
```

DeadLock & starvation :-

S, Q ⟶ semaphores.

```
   P0                 P1

wait (S)           wait (Q)
wait (Q)           wait (S)
   :                  :
   :                  :
signal(S);         signal(Q);
signal(Q);
```

Type of semaphores :-

- * Binary semaphore. ⟶ 0 or 1
- * Counting semaphore. ⟶ +ve integer

Classical problem of synchronisation :-

- * Bounded buffer.
- * Reader / writer problem
- * Dining philosopher problem.

Bounded buffer :-

shared Data : { Mutex = 1, Full - No. of full slots (0), empty - " empty slots (n) }

Structure of producer process :-

```
do
{
 Produce an item in next .
 wait (empty);
 wait (mutex);
 add next p to buffer;
 signal(mutex);
 signal (full);
}while(1);
```

Structure of consumer process:-

```
do
{
 wait(full);
 wait(mutex);
 ---------
remove an item from buffer to next
 ---------
signal(mutex);
signal(empty);
 ------
Consume the item in next L;
} while(1);
```

Reader Writer problem:-

Writer:-

```
wait(wrt);
 -----
 -----
waiting is performed
 -----
signal(wrt);
```

Shared variable:-

```
semaphore rw mutex = 1
semaphore mutex = 1;
 readcount = 0.
```

Reader:-

```
while(true).
{
 waib(mutex);
 readcount ++;
 if (readcount == 1)
wait(wrt);
signal(mutex);
// reading is performed.
wait(mutex);
read count --;
if(read count==0)
signal(wrt);
signal(mutex);
}
```

① Dining philosophor problem:-

structure philosopher :-

```
do
{
wait (Chopstick [i]);
wait (chopstick [(i+1)%5]);
------
eat
------
signal (chopstick [i]);
signal(chopstick (i+1)%5]);
} while(1);
```

Problem: if all philosopher feels hungry at same time

soln:- 1) Allow atmost 4 philosopher to simultaneously sit.

Start eating when both chopstiks are available.

odd philosopher – left & then Right chopstick

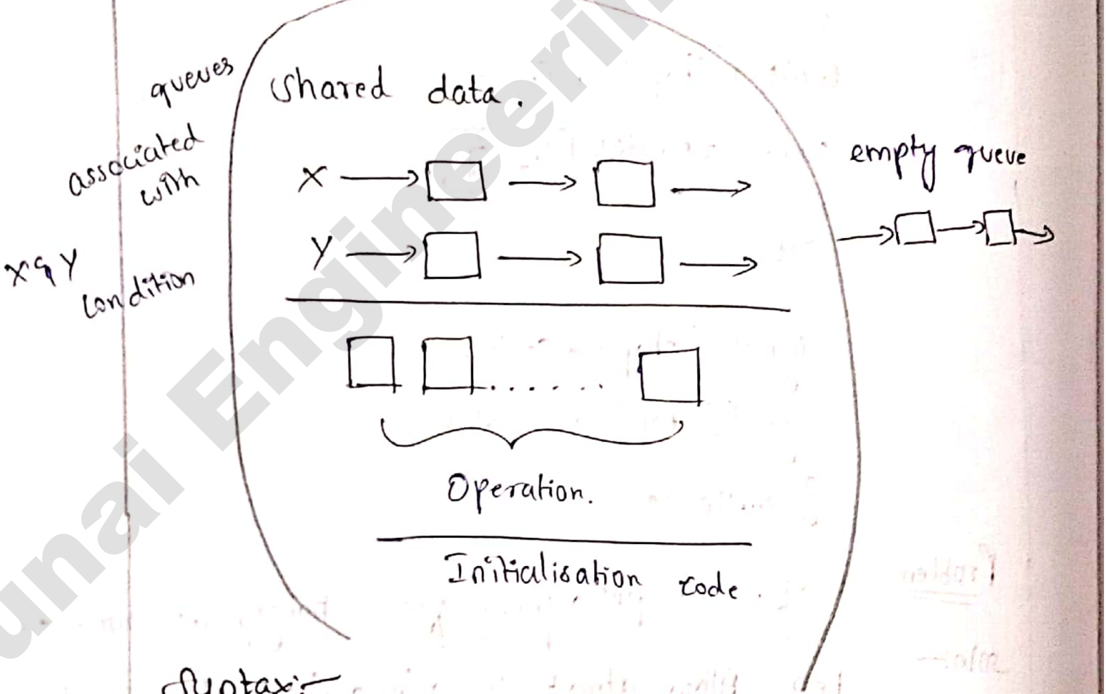
Even " – Right & left chopstick.

Critical Regions:-

It is a high level synchronisation construct in which a critical Region requires variable of type T shared by many processess can be declared as a : shared T

Monitors:-

It is a high level synchronisation constraint in which the procedures & data are group together in a single module or Package.



Syntax:-

```
monitor monitorname
{
shared variable declaration.
Procedure body P1 (...) {....}
   "      "    P2 (...) {....}
}
 Procedure body Pn (...) {...}
{
  Initialization code
 }
}
```

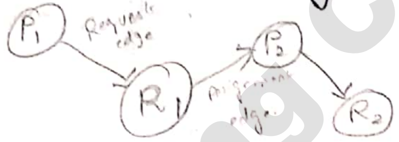
Solution to Dining philosopher problem:-

```
monitor dp
{
enum { thinking, hungry, eating } state[5];
Condition self[5];
void pickup (int i)
 {
  state[i] = hungry;
   test(i);
   if(state [i] != eating)
  self[i].wait;
 }
void  test (int i)
 {
  if(state (i+4) % 5 != eating && (state[i]=hungry)
                              && state[(i+1) %5] != eating))
 state[i] = eating;
  self[i]. signal;
 }


void  putdown(int i)
{
  state[i] = thinking;
  test[(i+4) %5];
 test[(i+1)%4];

}
```

```
initialisation_code()
{
  for(int i=0; i<5;i++)
   state[i]= thinking;
 }
}
```

Deadlock:-

Situation where a process tends to wait indefinitely for a resource held by other process. writing



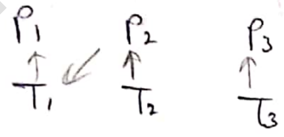
System Model:-

- Finite no. of resources [memory space, CPU cycles, I/o devices]

Requisition of resource in the ~ing order:-

1) Request
2) Use
3) Release.

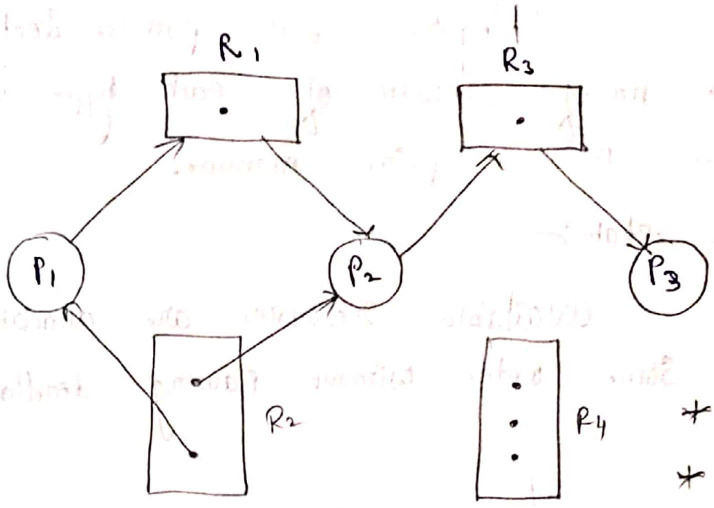
Eg:-



Deadlock Characterisation:-

1) Mutual Exclusion. (any one should be in non-sharable res)
2) Hold & wait (holding & request other resources)
3) No-preemption.
4) circular wait.

Resource - Allocation Graph:-



* Request edge.
* Assignment edge.
* claim edge.

Methods for Handling deadlock:-

① Deadlock prevention & Avoidance.

② Deadlock detection & Recovery.

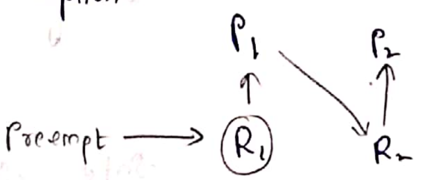
③ Ignore deadlock completely.

Deadlock prevention:-

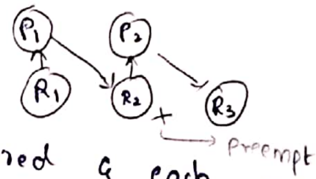
1) Mutual Exclusion.

2) Hold & wait → all resources should be allocated at the beginning
→ Process can request resource only if doesn't hold any "

3) No - preemption



4) Circular wait



If all resources are ordered & each process request. resource in an increasing order of enumeration.

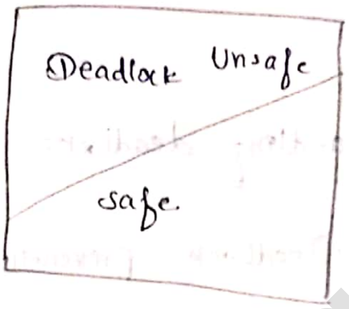
F(tapedrive) = 1
F(disk drive) = 5
F(Printer) = 12

① Deadlock Avoidance:-

Requires each process declare the max no. of resource of each type required by process in a prior mannar.

Safe state:-

available resources are allocated to process in some order without causing deadlock.



	Maxneed	Current need
P0	10	5
P1	4	2
P2	9	2

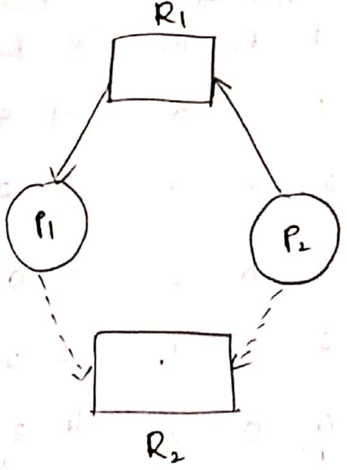
Total = 12 magnetic tape drives.

Totally allocated = 9

free = 3.

Safe sequence <P1, P0, P2>

Resource allocation graph:-



suitable for resource of single instance.

Banker's Alg :

Multiple instance of each resource type.

Data structures:-

Available : Resources & instances

Max : Max need of resources.

Allocation : Resources alloted to process .

Need : current requirement of resources by Process.

1) Consider a system consists of 5 proccesses P1, P2, P3, P4 & P5 & there are 3 resources namely

R1 → 10 instances
R2 → 5 "
R3 → 7 "

	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3			
P2	2	0	0	3	2	2	3	3	2
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

What is the content of need matrix This system as the stable stat solve it using Bankers algorithm.

Need Matrix = Max – alloc

	R1	R2	R3
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

P1 :-

Need ≤ available.

(7 4 3) ≤ 3 3 2 → False

P2 :-

N ≤ A

(1 2 2) ≤ 3 3 2 → True

New Available work } = Work + Allocation

= (3 3 2 + 2 0 0)

$\boxed{\text{New avai} = (5\ 3\ 2)}$

P_3 :-

$$N \le A$$

$$(6 \quad 0 \quad 0) \le (5 \quad 3 \quad 2) \longrightarrow \text{False}$$

P_4 :-

$$N \le A$$

$$(0 \quad 1 \quad 1) \le (5 \quad 3 \quad 2) \longrightarrow \text{True}$$

New available work } = work + Allocation

$$= (5 \quad 3 \quad 2) + (2 \quad 1 \quad 1)$$

$$\boxed{\text{New avail} = (7 \quad 4 \quad 3)}$$

P_5 :-

$$N \le A$$

$$(4 \quad 3 \quad 1) \le (7 \quad 4 \quad 3) \longrightarrow \text{True}$$

New available work } = work + Allocation

$$= (7 \quad 4 \quad 3) + (0 \quad 0 \quad 2)$$

$$\boxed{\text{New Available} = (7 \quad 4 \quad 5)}$$

P_1 :-

$$n \le A$$

$$(7 \quad 4 \quad 3) \le (7 \quad 4 \quad 5) \longrightarrow \text{false.}$$

~~New avai work } = work + Allocation~~

~~= (7 4 5) + (0 1 0)~~

~~New avail = (7 5 5)~~

(0 0 2)

If the request is granted the system may be in deadlock state after granting the request the available resources (0 0 2) so, the system tends to be unstable state.

P_3 :-

$$N \le A$$

$$(6 \quad 0 \quad 0) \le (7 \quad 4 \quad 5).$$

New avail work } = work + Allocation

$$= (7 \quad 4 \quad 5) + (3 \quad 0 \quad 2)$$

$$\boxed{\text{New avail} = (10 \quad 4 \quad 7)}$$

P_1 :—

$$(7 \quad 4 \quad 3) \le (10 \quad 4 \quad 7)$$

New available = work + allocation

$$= (10 \quad 4 \quad 7) + (0 \quad 1 \quad 0)$$

$$\boxed{\text{New avail} = (10 \quad 5 \quad 7)}$$

$$\boxed{\text{Safe sequence is } \{P_2, P_4, P_5, P_3, P_1\}}$$

2)

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2				
P_1	1	0	0	0	1	7	5	0	1	5	2	0
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

i) what is the content of need matrix !

ii) If the system tends to be safe state.

iii) If a request from process P_1 arrives for (0, 4, 2, 0) can be requests be granted immediately

Soln:- Need matrix } = Max − alloc

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

P_0 :-

$N \leq A$

$(0\ 0\ 0\ 0) \leq (1\ 5\ 2\ 0)$ — True.

$NA = (1\ 5\ 2\ 0) + (0\ 0\ 1\ 2)$

$\boxed{NA = (1\ 5\ 3\ 2)}$

P_1 :-

$N \leq A$

$(0\ 7\ 5\ 0) \leq (1\ 5\ 3\ 2)$ ← False

P_2 :-

$N \leq A$

$(1\ 0\ 0\ 2) \leq (1\ 5\ 3\ 2)$ → True

$NA = (1\ 5\ 3\ 2) + (1\ 3\ 5\ 4)$

$\boxed{NA = (2\ 8\ 8\ 6)}$

P_3 :-

$N \leq A$

$(0\ 0\ 2\ 0) \leq (2\ 8\ 8\ 6)$

$NA = (2\ 8\ 8\ 6) + (0\ 6\ 3\ 2)$

$\boxed{NA = (2\ 14\ 11\ 8)}$

P_4 :-

$N \leq A$

$(0\ 6\ 4\ 2) \leq (2\ 14\ 11\ 8)$ → True.

$NA = (2\ 14\ 11\ 8) + (0\ 0\ 1\ 4)$

$\boxed{NA = (2\ 14\ 12\ 12)}$

P_1 :-

$N \leq A$

$(0\ 7\ 5\ 0) \leq (2\ 14\ 12\ 12)$

$NA = (2\ 14\ 12\ 12) + (1\ 0\ 0\ 0)$

$\boxed{NA = (3\ 14\ 12\ 12)}$

Safe sequence $\} = \{P_0, P_2, P_3, P_4, P_1\}$.

$P_1 \Rightarrow (0\ 4\ 2\ 0)$

" − (Need)

$\Rightarrow (0\ 4\ 2\ 0) - (0\ 7\ 5\ 0)$

$P_1 \Rightarrow (0\ 3\ 3\ 0)$.

~~P_1:-~~

~~$N \leq A$~~

~~$(0\ 3\ 3\ 0) \leq (1\ 5\ 2\ 0)$~~

Need matrix:-

P_0	0	0	0	0
P_1	0	3	3	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

$P_1 \Rightarrow (0\ 4\ 2\ 0) - (1\ 5\ 2\ 0)$

$\text{New}_{\text{available}} = (1\ 1\ 0\ 0)$.

P_0:-

$N \leq A$

$(0\ 0\ 0\ 0) \leq (1\ 1\ 0\ 0) \longrightarrow$ True

$NA = (1\ 1\ 0\ 0) + (0\ 0\ 1\ 2)$

$\boxed{NA = (1\ 1\ 1\ 2)}$

P_1:-

$N \leq A$

$(0\ 3\ 3\ 0) \leq (1\ 1\ 1\ 2) \longrightarrow$ False

P_0:

N ≤ A

(1 0 0 2) ≤ (1 1 1 2) → True

NA = (1 1 1 2) + (1 3 5 4)

$\boxed{NA = (2\ 4\ 6\ 6)}$

P_2:

N ≤ A

(0 0 2 0) ≤ (2 4 6 6) → True

NA = (2 4 6 6) + (0 6 3 2)

$\boxed{NA = (2\ 10\ 9\ 8)}$

P_4:

N ≤ A

(0 6 4 2) ≤ (2 10 9 8) → True

NA = (2 10 9 8) + (0 0 1 4)

$\boxed{NA = (2\ 10\ 10\ 12)}$

P_1:

N ≤ A

(0 3 7 0) ≤ (2 10 10 12)

NA = (2 10 10 12) + (1 0 0 0)

$\boxed{NA = 3\ 10\ 10\ 12}$

3) The OS contain 3 Resources the no. of instance of each resource are (7, 7, 10) the current resource allocation is given below.

Process	current allocation			Max need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8
P_2	2	0	3	4	3	3
P_3	1	2	4	3	4	4

(5 4 10)

i) Is the current allocation is in a safe state

ii) Can the request from process P_1 (1, 1, 0) be granted.

soln:- (i) Available = No. of instance − sum of Alloc

= (7, 7, 10) − (5, 4, 10)

$\boxed{\text{Available} = (2, 3, 0)}$

Need Matrix ⇒ Max − alloc.

	R1	R2	R3
P_1	1	4	5
P_2	2	3	0
P_3	2	2	0

P_1:- N ≤ A

(1, 4, 5) ≤ (2 3 0) ⇒ False

P_2:- (2 3 0) ≤ (2 3 0) ⇒ False.

P_3 :- $(2\ 2\ 0) \leq (2\ 3\ 0)$ => True,

$NA = (2\ 3\ 0) + (1\ 2\ 4)$

$\boxed{NA = (3\ 5\ 4)}$

P_1 :-

$N \leq A$

$(1\ 4\ 5) \leq (3\ 5\ 4)$ => ~~False.~~ ~~True.~~ False

~~P_2~~ ~~$NA = (3\ 5\ 4) + (2\ 2\ 3)$~~

~~$NA = (5\ 7\ 7)$~~

P_2 :-

$N \leq A$

$(2\ 3\ 0) \leq (3\ 5\ 4)$ => True.

$NA = (3\ 5\ 4) + (2\ 0\ 3)$

$\boxed{NA = (5\ 5\ 7)}$

P_1 :-

$N \leq A$

$(1\ 4\ 5) \leq (5\ 5\ 7)$ => True

$NA = (5\ 5\ 7) + (2\ 2\ 3)$

$\boxed{NA = (7\ 7\ 10)}$

Safe sequence = $\{P_3, P_2, P_1\}$.

$P_1[(1,1,0) - (1\ 4\ 5)]$

$P_1\ (0\ 3\ 5)$.

Available = $(2\ 3\ 0) - (1\ 1\ 0)$

$= (1\ 2\ 0)$

P_1	0	3	5
P_2	2	3	0
P_3	2	2	0

P_1:-

$(0, 3, 5) \le (1, 2, 0) \Rightarrow$ False

P_2:-

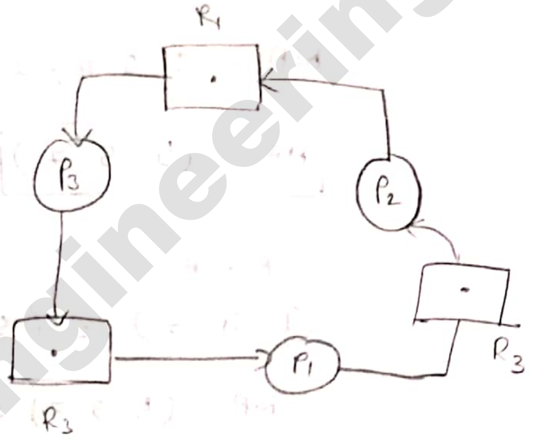
$(2\ 3\ 0) \le (1, 2, 0) \Rightarrow$ False

P_3:-

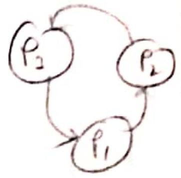
$(2, 2, 0) \le (1, 2, 0) \Rightarrow$ False.

∴ The request from process P_1 has no safe sequence hence it is denied.

Deadlock detection:-



wait for graph



It is process of determining that the deadlock exist an identifying the processes & resources involved in a dealock.

Deadlock Recovery:-

1) Process termination ⟶ Aborting one process at a time to remove circular wait.

2) Resource preemption.

3) Recovery through rollback → after process termination rollback to safe state.

4) Starvation → select a victim for specific no of item.

Selection parameters :-

Priority of process.

What percentage of execution that process has completed?

Resource used by process.

No. of resources required to complete its execution.

Type of process < Batch
Interactive.

Resource preemption :-

- select a victim process

↓ factors

- Priority

- CPU time used by process.

- No. of process affected by this process.

1) Banker's Algorithm:-

Process	Max A B C D	Alloc Need	Available.
P0	6 0 1 2	4 0 0 1	
P1	1 7 5 0	1 1 0 0	3 2 1 1
P2	2 3 5 6	1 2 5 4	
P3	1 6 5 3	0 6 3 3	
P4	1 6 5 6	0 2 1 2	

i, How many resources of type A, B, C, D are there

ii, What is the content of need matrix.

iii Is the system in the safe state why!

iv, If a request from process P4 arrives for additional resource (1 2 0 0) can the request be granted?

Soln:- i) Resources => (A B C D) = (9 13 10 11)

ii) Need Matrix:-

Max - alloc

	A	B	C	D
P0	2	0	1	1
P1	0	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

iii) P_0 => $N \le A$

$(2\ 0\ 1\ 1) \le (3\ 2\ 1\ 1)$ => True.

$NA = (3\ 2\ 1\ 1) + (4\ 0\ 0\ 1)$

$\boxed{NA = (7\ 2\ 1\ 2)}$

P_1 :-

$N \not\le A$

$(0\ 6\ 5\ 0) \not\le (7\ 2\ 1\ 2)$ => False

P_2 :-

$N \le A$

$(1\ 1\ 0\ 2) \le (7\ 2\ 1\ 2)$ => True

$NA = (7\ 2\ 1\ 2) + (1\ 2\ 5\ 4)$

$\boxed{NA = (8\ 4\ 6\ 6)}$

P_3 :-

$N \le A$

$(1\ 0\ 2\ 0) \le (8\ 4\ 6\ 6)$ => True

$NA = (8\ 4\ 6\ 6) + (0\ 6\ 3\ 3)$

$\boxed{NA = (8\ 10\ 9\ 9)}$

P_4 :-

$N \le A$

$(1\ 4\ 4\ 4) \le (8\ 10\ 9\ 9)$ => True

$NA = (8\ 10\ 9\ 9) + (0\ 2\ 1\ 2)$

$\boxed{NA = (8\ 12\ 10\ 11)}$

P_1 :-

$N \le A$

$(0\ 6\ 5\ 0) \le (8\ 12\ 10\ 11)$

$NA = (8\ 12\ 10\ 11) + (1\ 1\ 0\ 0)$

$\boxed{NA = (9\ 13\ 10\ 11)}$

safe sequence = $\{P_0, P_2, P_3, P_4, P_1\}$

(iv)

$P_4(1\ 2\ 0\ 0) - P_4(1\ 4\ 4\ 4)$

$P_4(0\ 2\ 4\ 4)$

$NA = (1\ 2\ 0\ 0) - (3\ 2\ 1\ 1)$

$= (2\ 0\ 1\ 1).$

~~For~~

$N \leq A$

Need Matrix:-

$$\begin{array}{c|cccc}
 & A & B & C & D. \\
P_0 & 2 & 0 & 1 & 1 \\
P_1 & 0 & 6 & 5 & 0 \\
P_2 & 1 & 1 & 0 & 2 \\
P_3 & 1 & 0 & 2 & 0 \\
P_4 & 0 & 2 & 4 & 4
\end{array}$$

P_0 :-

$N \leq A$

$(2\ 0\ 1\ 1) \leq (2\ 0\ 1\ 1) \Rightarrow F$

P_1 :-

$N \leq A$

$0\ 6\ 5\ 0 \leq 2\ 0\ 1\ 1 \Rightarrow F$

$1\ 1\ 0\ 2 \leq 2\ 0\ 1\ 1 \Rightarrow F$

$0\ 2\ 4\ 4 \leq 2\ 0\ 1\ 1$

24/02/2020.

UNIT – III

Storage Management.

Memory Input:-

Purpose – Execute all pgms → Main memory (data)

Background:-

Memory → large array of words.

- instructions fetch according to values of PC
- decoded
- Results stored back in memory.

Address Binding:-

Program resides in disk & need to be brought to main memory.

Address may vary

source program addr → symbolic addr

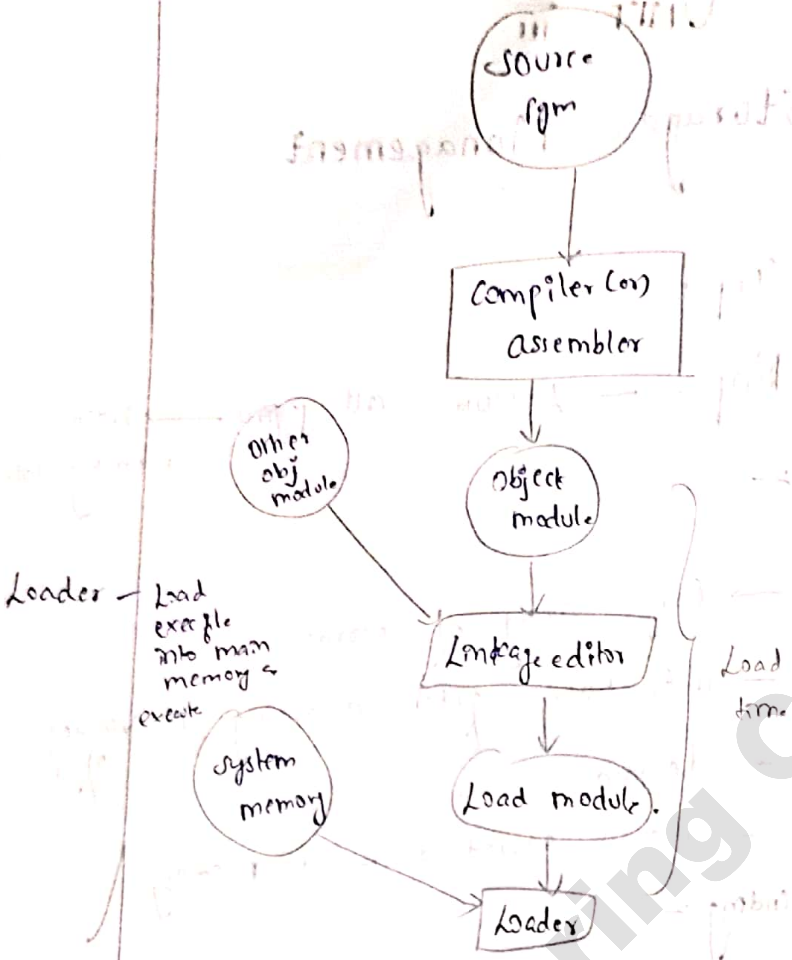
↓ ← compiler

relocatable. addr

↓ linkage editor (or) loader

absolute addr.

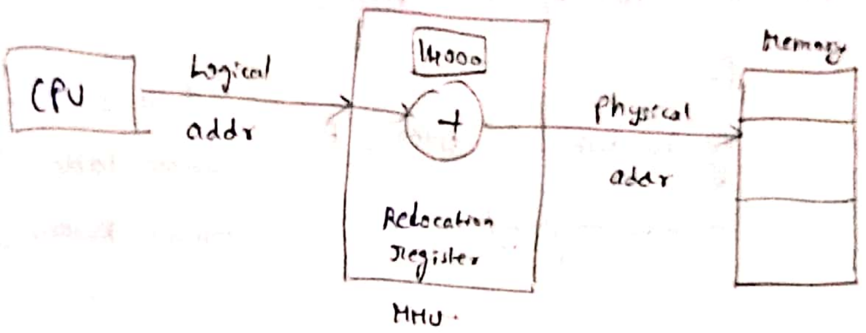
Multistep processing of user pgm:-



Logical (vs) Physical address space :-

Logical addr (or) Virtual addr } → Logical address space. address generated by CPU.

Physical addr → Physical address space. address seen by memory unit



Dynamic Relocation using Relocation Register.

Dynamic Loading:—

A Routine is not loaded until it is called.

Overlays:—

It enables a process to be larger than the amount of memory allocated to it.

Main idea:— Keep in memory only those instructions & data that are needed at any given time.

When other instructions are needed it is loaded into the space occupied previously by the instructions that are ~~load~~ no longer needed.

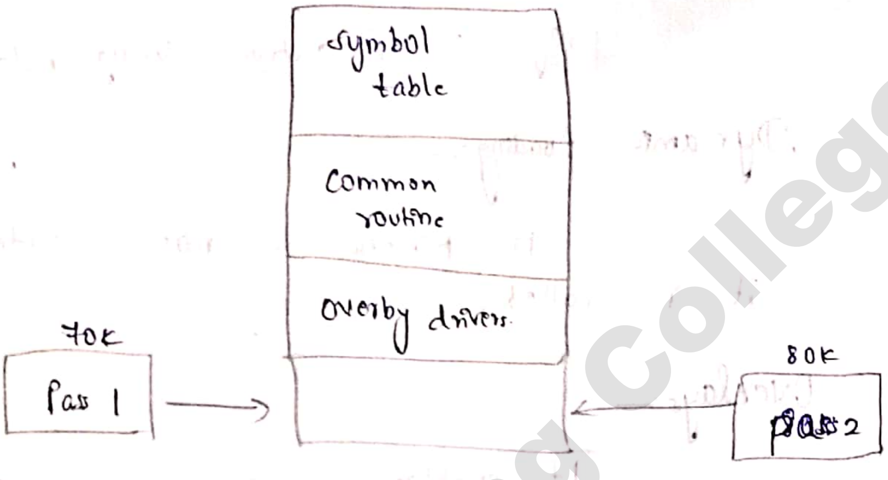
Consider 2 pass assembler,

Pass 1	70 KB	Require 200KB.
Pass 2	80KB	
symbol table	20KB	
common Routines	30KB	

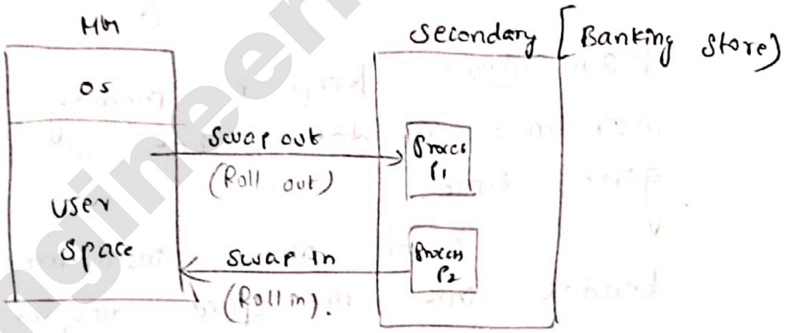
Assume 150KB is Available :-

Pass 1
symbol table } overby A
Common routine

Pass 2
symbol table } overby B.
common Routine



Swapping :-

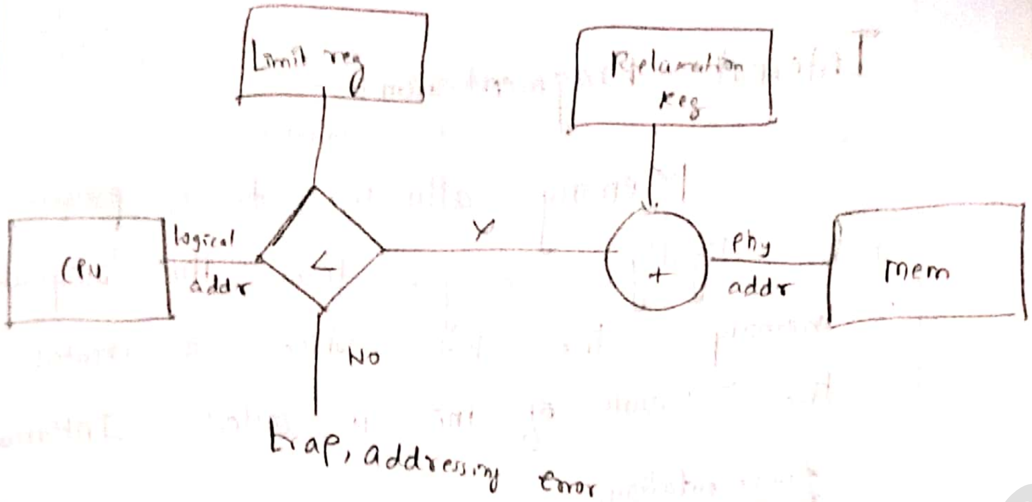


Reasons :-

1, Time Quantum Expires
2, High priority process arrives.
3, Interrupts.
4, Process puts into waiting Queue.

Contiguous Memory allocation :-

It is the process of allocating consequtive memory blocks to a process.



lim reg → Continue range of log addr.

Relocation Reg → value of smallest phy adder.

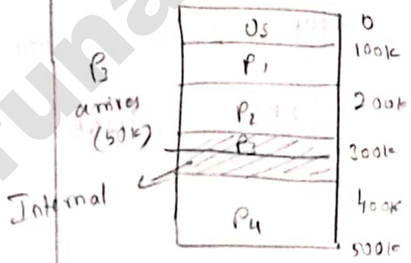
Two variables. [Partition allocation]

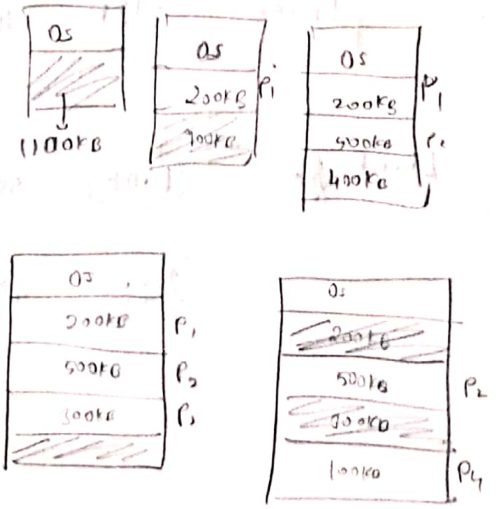
1) Single partition Allocation
- Allocate overall memory to user without any use of h/w or s/w.
- ① Divide memory into 2 partitions [user & OS]

2) Mutual Allocation → Partition of memory into multiple partition.

Fixed partitioning (fixed sized blocks)

① Dynamic partitioning [partitions created dynamically depending on size of process]





soln for External frag
↓
Compaction

Internal Fragmentation :-

Memory allocated to a process may be slightly larger than the requested memory the hole which is created as the result of this is called Internal fragmentation.

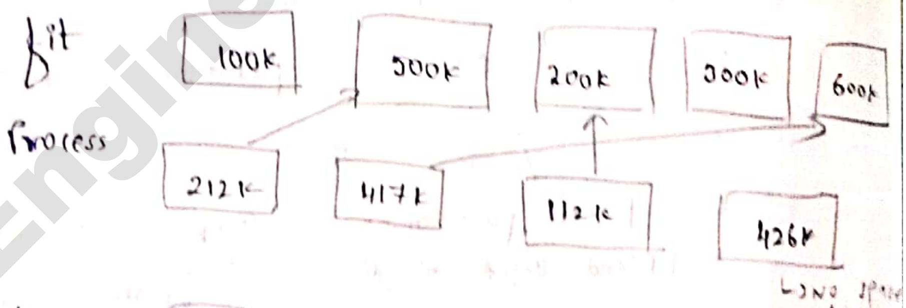
Compaction :-

Shuffle the memory content to place all the free memory together in one large block.

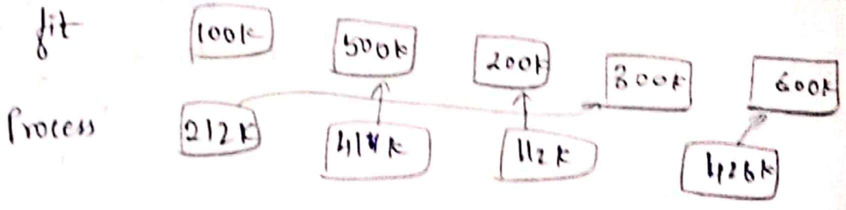
The compaction is possible only if relocation is dynamic.

Allocation strategies :- (Partition allocation Method)

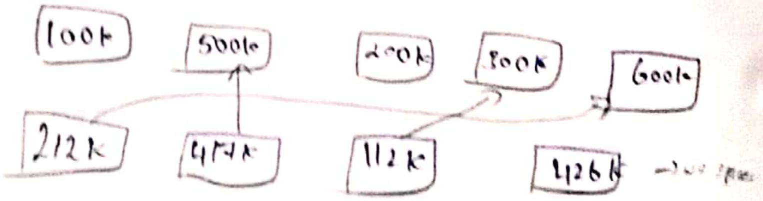
1) First fit



2) Best fit

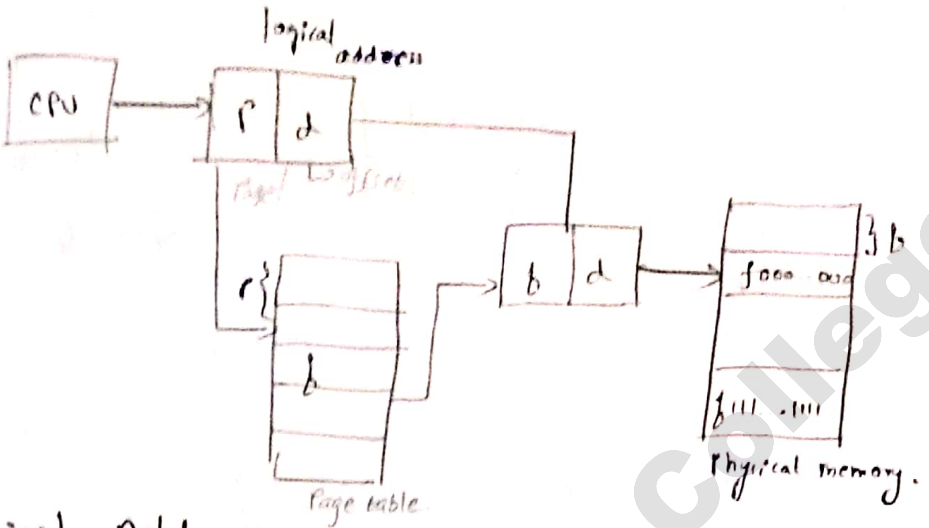


3) Worst fit

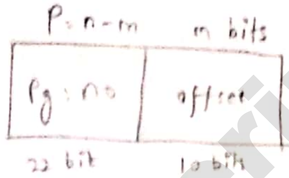


Paging:-

Memory Management scheme that allows physical address to be non-contiguous

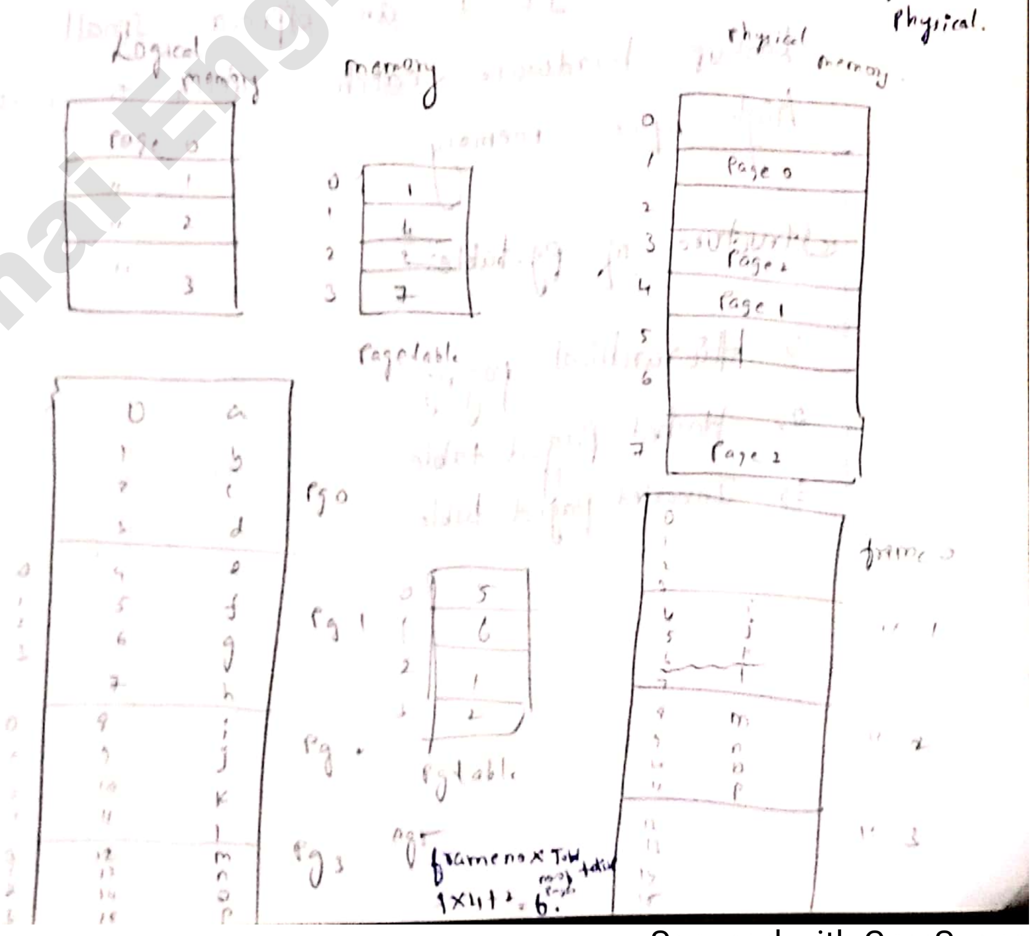


logical Address:-

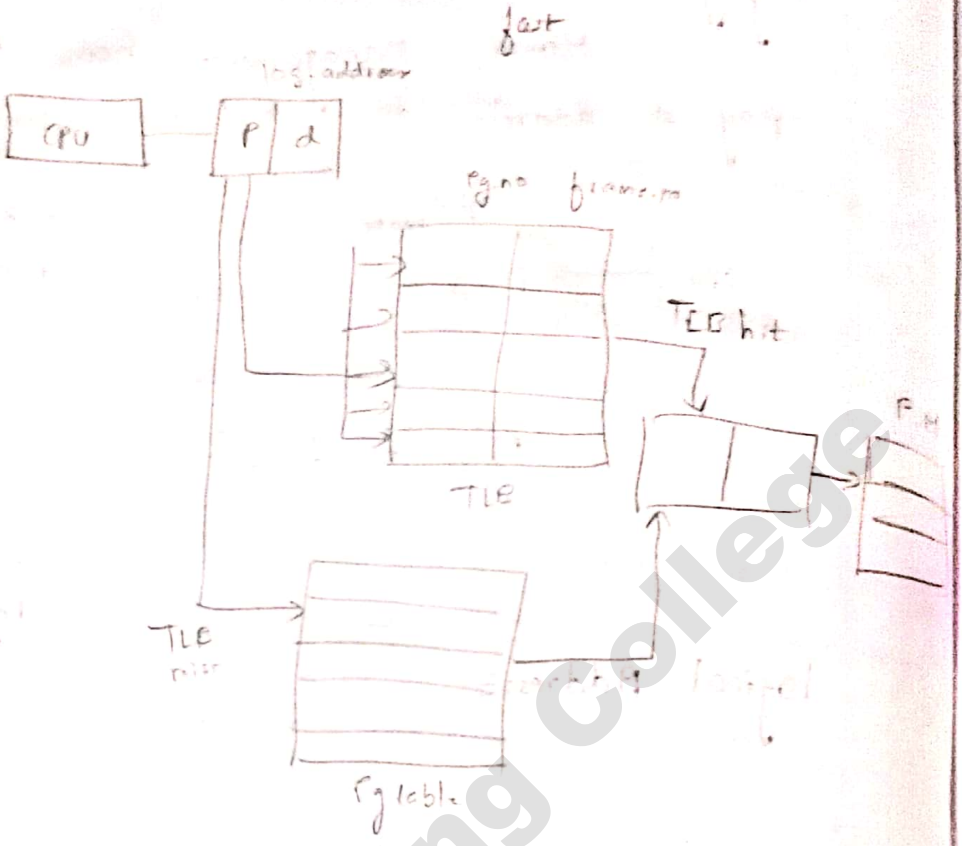


Pg. No → index into page table

Page table → Contains base address of each page in physical.



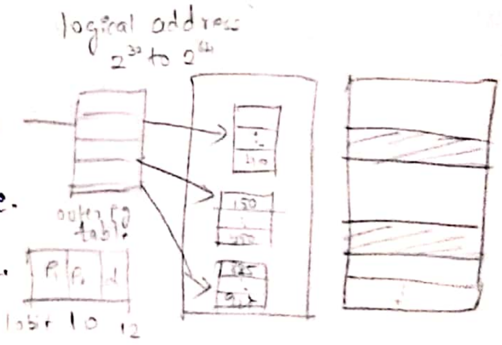
TLB [Translation Lookaside buffer].



It is an special small passed lookup hardware catch which is fast associative & high speed memory.

Structure of pg-table:-

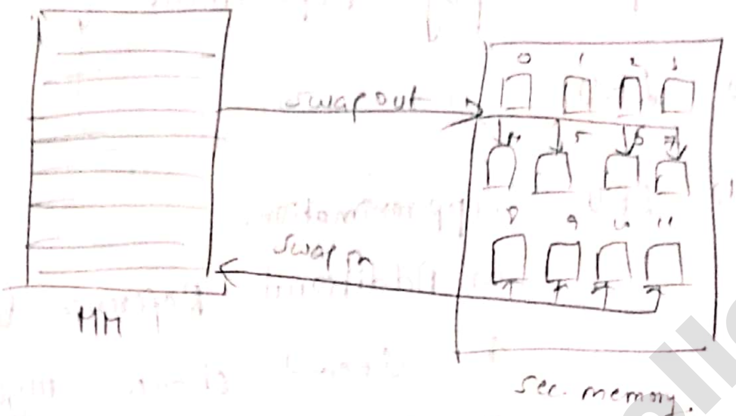
1) Hierarchical paging.
2) Hashed paged table.
3) Inverted paged table.



Refer book material

Demand paging:-

The pages are loaded into memory only they are demanded during execution.

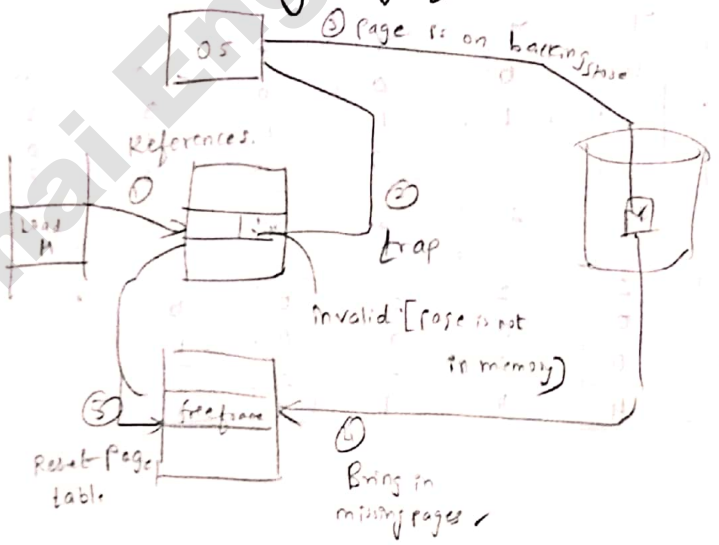


Swapper ⟶ A swapper swaps the entire process.

Pager ⟶ Pager swaps individual page in this process.

Lazy swap ⟶ It never swaps a page into memory unless the page is needed.

(7) Steps in handling pg-fault:-



Page Replacement :-

1) FIFO

2) Optimal page Replacement.

3) LRU " "

4) LRU approximation.

   (i) Additional Reference bit Algorithm.

   (ii) Second chance Algorithm

   (iii) Enhanced second chance "

5) Counting base Page Replacement

6) Page Buffering Algorithm.

Reference String

7 0 1 2 0 3 0 4 2 3 0 2 1 2 0 1 7 0 1

7	7	7	7	0	0	0	0
	0	0	0	0	3	3	3
		1	1	1	1	0	0
			2	2	2	2	4
*	*	+	+	*	-		

2	2	2	2	0	0
3	0	0	0	0	1
0	0	2	2	2	2
4	4	4	1	1	1

Belady's Anamoly:-

The no.of page fault increases with increasing no of frames.

Thrashing:-

A high paging activity is called Thrashing. A Process spends more time in paging then its execution.

LRU Page Replacement:-

LRU (←):-

7	7	7	2	2	0	0	0
	0	0	0	0	0	4	4
		1	1	3	3	3	2

3	3	1	1	1	1	
4	0	0	2	2	7	
2	2	2	2	0	0	

7	7	7	7	3	3
	0	0	0	0	0
		1	1	1	4
			2	2	2

Optimal Page Replacement (⟶).

Reference string :-

7 0 1 2 0 3 0 4 2 3 0 2 1 2 0 1 7 0 1

7	7	7	2	2	2	2	2	7
	0	0	0	0	4	0	0	0
		1	1	3	3	3	1	1

No. of page fault = 9

7	7	7	7	3	3	3	7
	0	0	0	0	0	0	0
		1	1	1	4	1	1
			2	2	2	2	2

No. of page fault = 8

FIFO :-

Time when Pg was brought into memory Replace the oldest page.

Problem :-

Belady's Anamoly.

LRU :- (Least Recently used)

Replace the page that has not been used for longest period of time.

time stamp.

Implementation of LRU :-

1) Counter

2) Stack.

4 7 0 7 1 0 1 2 1 2 1 2 7 2 1 2

↑ ↑
a b

2	7
1	2
0	1
7	0
4	4

Stack before a

Stack after b.

LRU approximation:—

Reference bit ⟶ 0

Additional Reference bit:— 1

Recording reference bits at regular time intervals.

11111111 ⟶ Page that is used atleast once

11000100 ↓ Used recently than this.

01110111 ↑

Second chance Alg:—

Basic FIFO

0 ⟶ replace it

⟶ given second chance Change it to 0.

Counting Based Page Replacement :-

Keep a counter of no.of references that have been made to each page.

Least Frequently used (LFU) :-

Replace the page with least count.

Most Frequently used (MFU) :-

Replace the page with the larger count Value. ~~Value~~

~~Page Ac Buffering Algorithm~~ :-

Optimal Page Replacement } Replace the page that will not be Used for longest period of time.

Allocation of frames :-

(i) Equal allocation ⇒ $\frac{\text{No.of free frames}}{\text{No.of process}}$

(ii) Proportional " ⇒ Process which have more logical space get more frame

(iii) Priority " → High priority gets more frames.

* Global Allocation :-
+ Local Allocation :-

1) Consider a system consists of 64 frames & 4 process. the virtual memory size is given as . V(1) = 16, V(2) = 128, V(3) = 64, V(4) = 48. Allocate free Page frame by using equal allocation & proportional allocation policy.

soln:-

Equal Allocation ->

$$\frac{\text{No. of free frames}}{\text{No of process}}$$

$$= \frac{64}{4}$$

$= 16$ frames/ process.

Proportional Allocation =>

sum of all virtual memory } $= 16+128+64+48$

$= 256.$

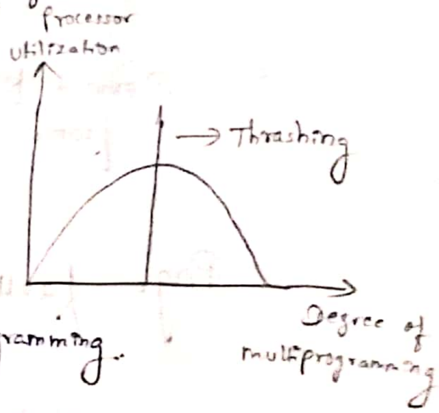
$P_1 = \frac{16}{256} \times 64 = 4$ frames

$P_2 = \frac{128}{256} \times 64. = 32$ frames

$P_3 = \frac{64}{256} \times 64 = 16$ frames

$P_4 = \frac{48}{256} \times 64 = 12$ frames

Thrashing :-



-> High paging activity

-> Long-term (or) medium term scheduler detects degree of multiprogramming.

-> Local allocation -> Limits effects of thrashing

-> Global allocation -> increases thrashing

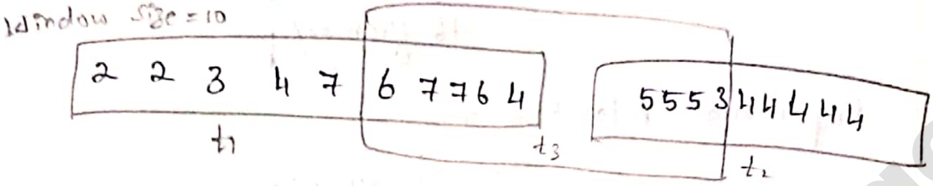
The problem of thrashing can be removed by

* Working set Model =>
* Page fault frequency

Working set Model :-

Set of Pages in most ~~memory~~ reason reference.

(i.e) pages use by a process within a window of time.



Based on locality of reference

Temporal

Spatial.

Δ - too small ⇒ entire locality

Δ - too large ⇒ several "

Δ - ∞ ⇒ entire program.

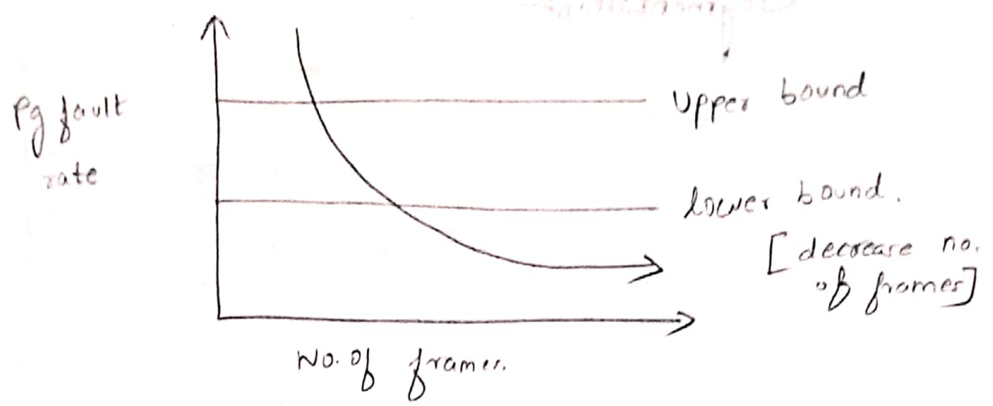
Demand for frame ← $\boxed{D = \sum WSS_i}$

if D > W , thrashing occurs.

Page fault frequency :-

fault frequency increase ⟶ increase size of memory pool.

fault frequency decrease } ⟶ decrease size of memory pool.



Shared pages[Address of paging].

Common code shared by more than one processes.

2m ⊛ Reentrant code (or) pure code (used for sharing.)

↓

Non-self modifying code never change during execution.

Process P1	Pg table P1	Process P2	Pgtable P2	Process P3	Page table P3
ed 1	3	ed 1	3	ed 1	3
ed 2	4	ed 2	4	ed 2	4
ed 3	6	ed 3	6	ed 3	6
data 1	1	data 2	7	data 3	2

Segmentation:

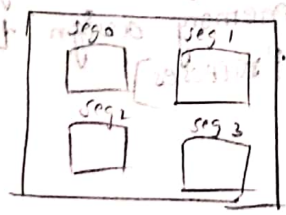
→ Logical memory divided into no. of segments.

→ Memory divided into segments

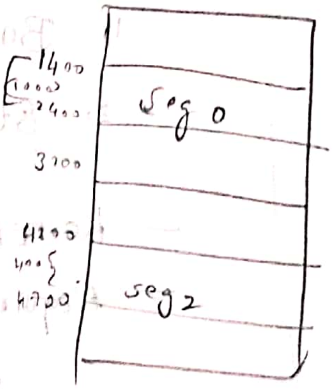
<segment name, offset>

Name length.

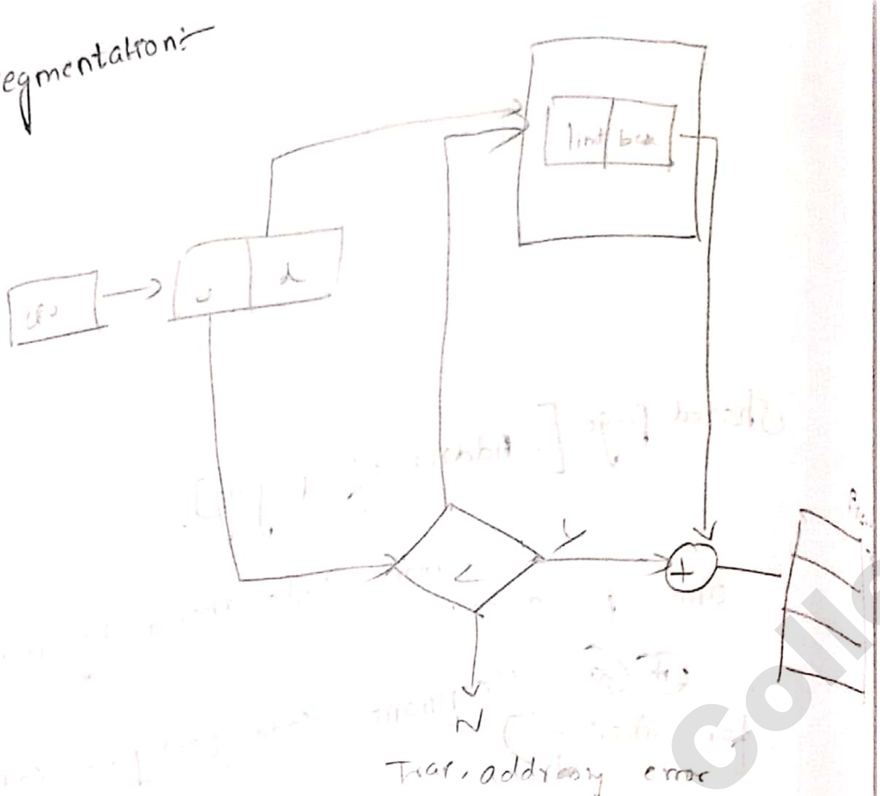
0	▨
1	data 1
2	data 3
3	ed 1
4	ed 2
5	▨
6	ed 2
7	data 2



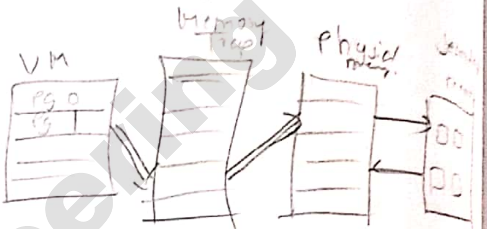
	Length limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	.	.
4	:	:
5	:	:



Segmentation:-



Virtual Memory:-



It is a technique that allows the execution of the processes that may not be completely in ~~Level~~ memory.

The program size may be larger than physical memory.

It abstract the main memory into an extremely larger uniform array of storage.

Allocating Kernel memory [① Discuss the strategies use for managing free memory asign to kernel Processes]

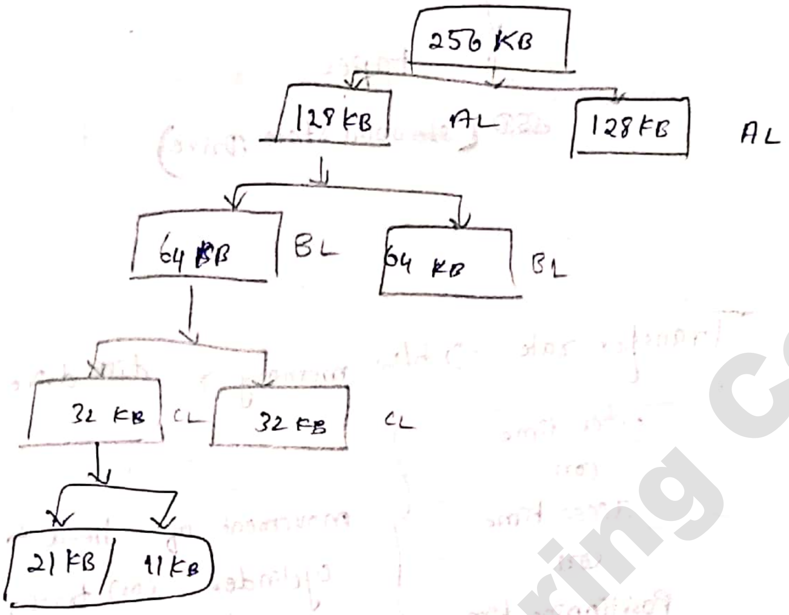
1) Buddy segment
2) Stab Allocation

— Kernel request memory for data structure [Process descriptor file object semaphore
— Requires contiguous memory

Buddy system:-

Memory — power of 2 Allocator.

Memory size => 256 KB Request -> 21 KB.



Advantages:-

* Coalescing / compaction.

Disadvantages:-

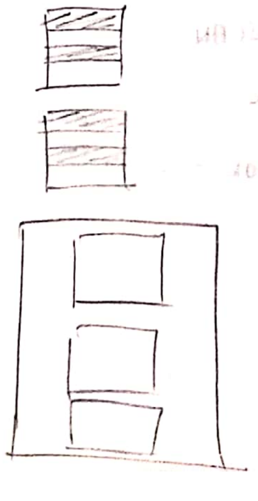
+ Internal fragmentation.

* Leads to. wastage of memory

Allocating Kernel Memory:-

1, Buddy system

2, ~~Sla~~ slab allocation.



UNIT - IV

Mass storage structure:-

Magnetic disk

" tapes

SSD (~~sto~~ solid state Drive)

Transfer rate => b/w memory & disk drive

seek time (or) access time (or) Positioning time } movement of head to desired cylinder (or) track.

Rotational latency ⟶ mov of head to desired sector.

Problems:-

Disk schedule:-

Type of "

1) First come first serve [FCFS].
2) SSTF [shortest seek time first]
3) SCAN
4) C—SCAN
5) LOOK
6) C-LOOK

1) On a disk with 200 cylinders numbered from 0 to 199 compute the no. of tracks the disk arm must move to satisfy all the request in the disk queue. If the request are 98, 183, 37, 122, 14, 124, 65, 67 and initial head position is at 53.

Soln:

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

(i) FIFO.

0
14
37
53
65
67
98
122
124
183
199

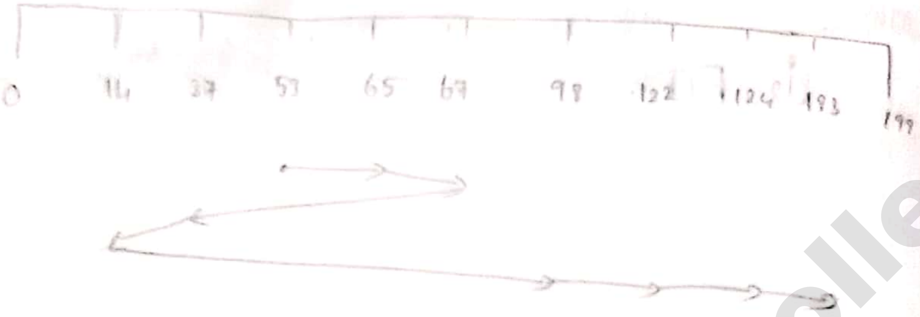
Next track Accessed	No. of tracks Accessed
98	45
183	[illegible] 85
37	[illegible] 146
122	[illegible] 85
14	[illegible] 108
124	[illegible] 110
65	59
67	2
	640 → Total head movement

Avg seek Length $= \frac{640}{8}$

$= 80$.

(ii) SSTF :-

98, 183, 37, 122, 14, 124, 65, 67.



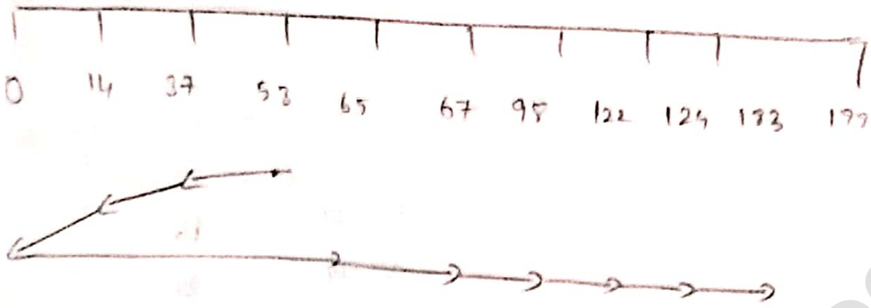
Next track accessed	No. of track accessed.
98	
183	12
	2
	30
	23
	14
	24
	2
	59
	236

Avg seek length $= \frac{236}{8}$

$= 29.5$

(iii) SCAN [Elevator algorithm].

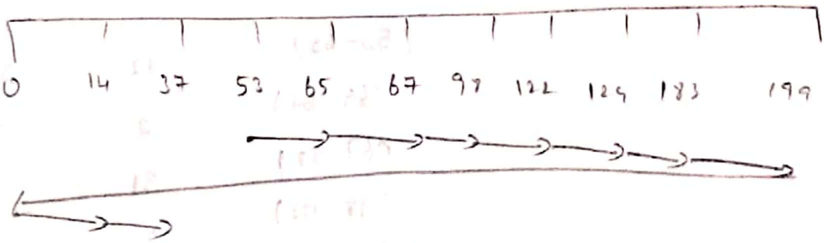
(Assume head moving towards 0)



53 to 37	16
37 to 14	23
14 – 0	14
0 – 65	65
65 – 67	2
67 – 97	31
98 – 122	24
122 – 124	2
183 – 124	59
	236 cylinders.

~~Avg seek~~ length = ~~$\frac{236}{8}$~~ = ~~29.5~~.

(iv) C–SCAN:– [It touches both end but it traverse in single direction

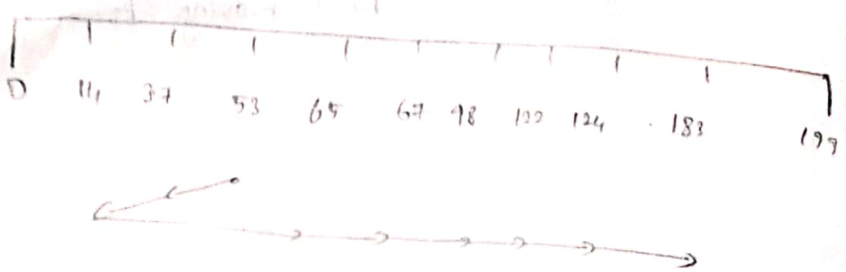


53 – 65	12
65 – 67	2
67 – 98	31
98 – 122	24
122 – 124	2
183 – 124	59
199 – 183	16
199 – 0	199
0 – 14	14
37 – 14	23

382 cylinders.

Avg seek. = 47.75.

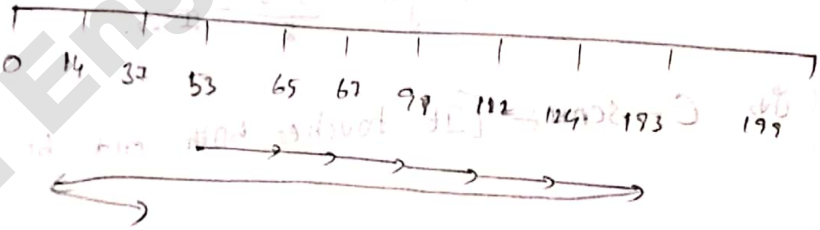
v) Look :-



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

Avg seek length = $\frac{208}{8}$ = 26

(vi) C-Look :- [It never touches the ends bt in traverse in single direction]

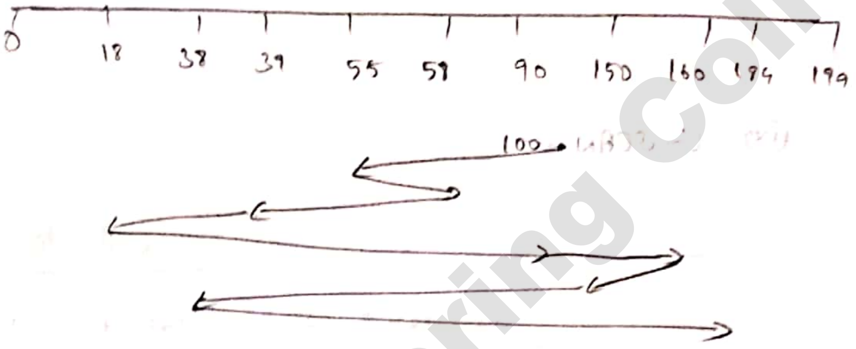


(53-65)	12
(65-67)	2
(67-98)	31
(98-122)	24
(122-124)	2
(124-183)	59
(183-14)	169
(14-37)	23
	322

Avg seek length : 40.25

{55, 58, 39, 18, 90, 160, 150, 38, 184}.

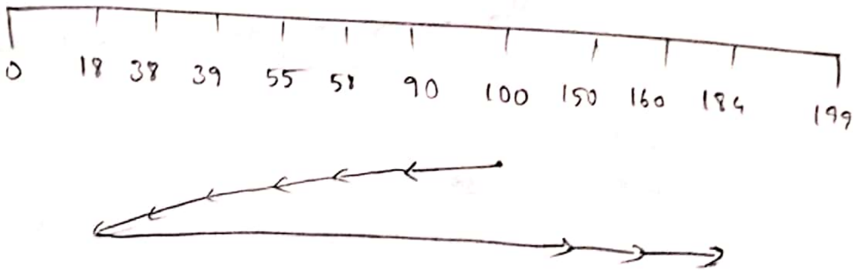
(i) FCFS.



45
3
19
21
72
70
10
112
146

498 /9 = 55.3

(ii) SSTF:-

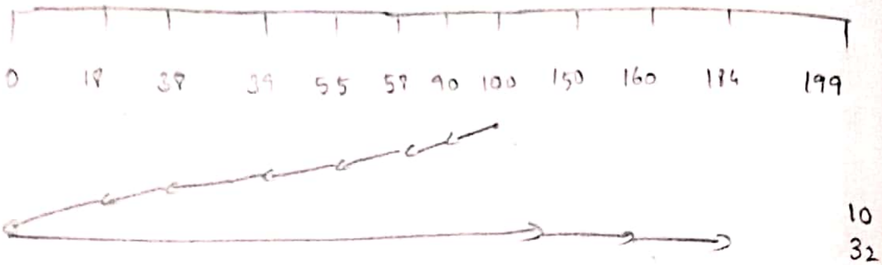


10
32
3
16
1
20
132
10
24

248 /9

= 27.5

(iii) SCAN:-

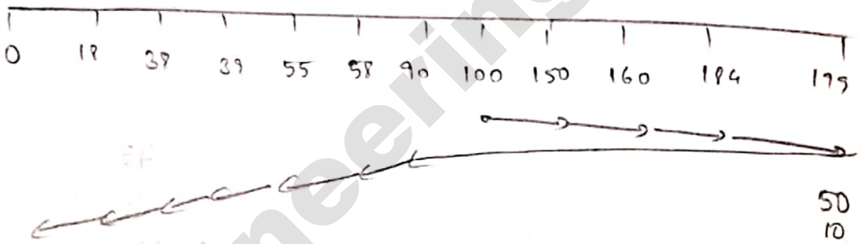


10
32
3
16
1
20
18
150
10
24

= 284 $\frac{284}{9}$

= 31.5

(iv) C-SCAN:-

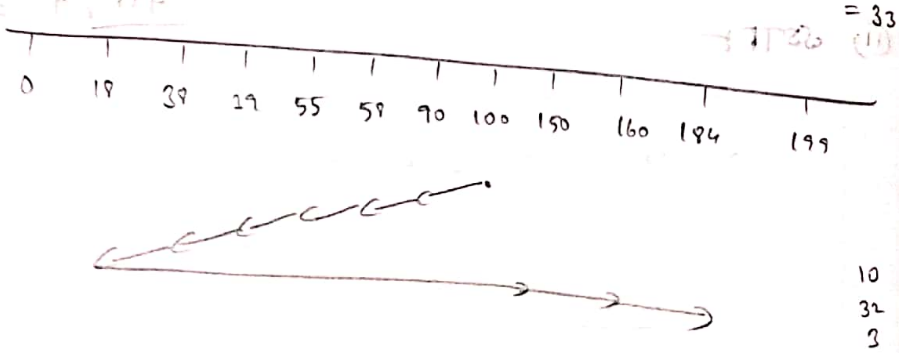


50
10
24
15
109
32
3
16
1
20
18

298 / 9

= 33.1

(v) LOOK

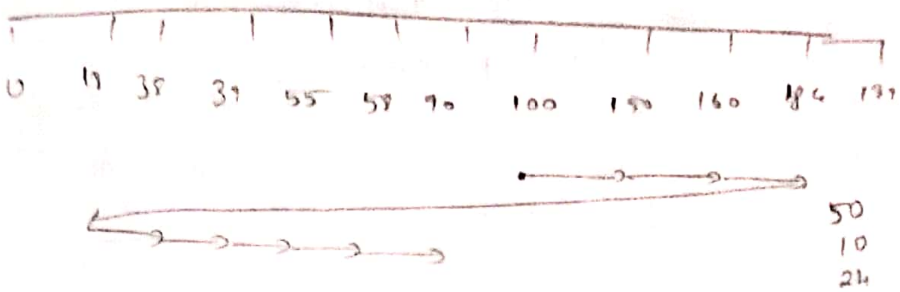


10
32
3
16
1
20
132
10
24

248/9

= 27.5

vi) C - Look:-



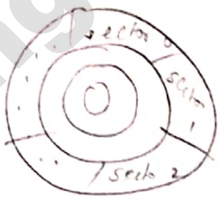
50
10
24
166
20
1
16
3
32

322/9
= 35.7

Disk structure:-

Disk drive logical blocks $\xrightarrow{\text{mapped}}$ sectors

large one dimensional array.



CAV (constant Angular velocity)
CLV (constant Linear velocity).

Methods of disk attachment:-

1) Host attached storage ← I/o (Ports → Protocols (SATA) Serial advanced Technology attachments)

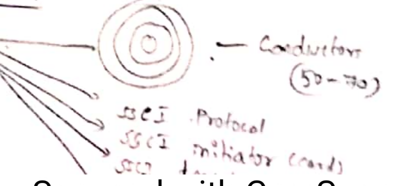
2, N/w attached storage.

Architecture

1, SCSI architecture

2, Fibre channel architecture

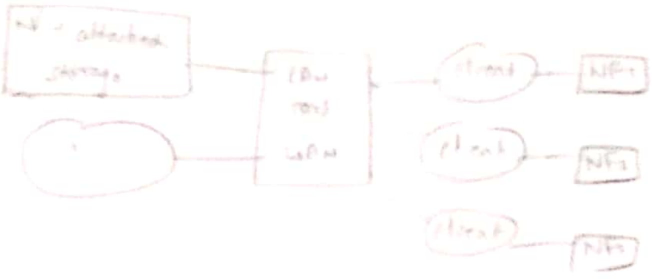
- copper cable
- NO of devices >



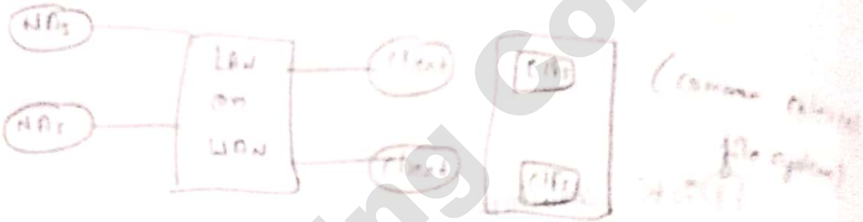
2) N/w attached storage:-

↓

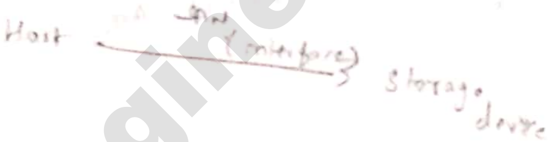
RPC : for clients
(Remote procedure call)



Window

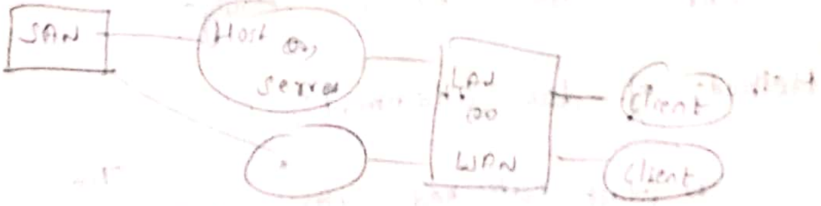


3) Storage area n/w.



Storage devices:-

1) Storage array
2) Tape library.



Disk Management:-

Loading OS into MM
&

1) Disk formating 2) Boot ~~Block~~ 3) Bad blocks.

i) Partitioning

ii) Logical

→ OS tables allocated &

Low level formating (or)

→ division of track & Sectors

→ Sectors

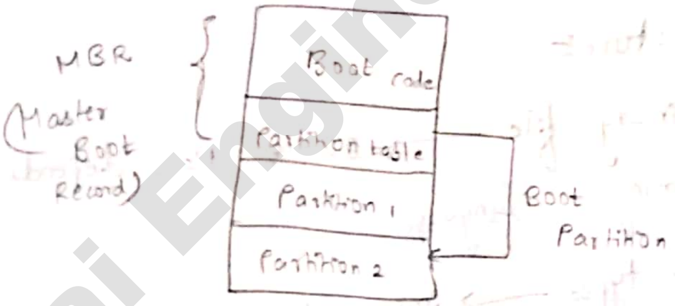
→ ECC (Error Correction Code)

→ 512 to 1024 bytes (data)

Boot Block:-

The Boot Block is present on in separate partition called Boot disk or system disk on Boot partition.

Windows 2000



Bad Blocks:-

→ head crash [R/w head touches the tracks (or) Sectors]
↓
Bad blocks.

→ IDE [Integrated Drive Electronics]
→ bad blocks managed logically
eg) FORMAT } CHKDSK } command in MS-DOS.

→ sector sparing (or) forwarding

→ sector skipping.

Swap space Management :-

File concepts :-

Group of similar records stored in secondary Memory

↓ data information.

Directory → Collection of files.

2m Attributes of file :-

1) Name.
2) Identifier
3) Type
4) Location
5) Size
6) Protection

File structure :-

Binary file → 0's & 1's. depends on Programming language.

file types → images, text, doc, video.

Default → exe files.

Operations on file :-

1) File creation - space, new entry.
2) Writing to a file - filename, data to be written, file pointer
3) Reading from a file - filename, current position to be read.

~~Directory structure:~~

Types of operation performed in Directory,

* Searching files.
* File creation.
* " Deletion.
* ~~Visit~~ Listing
* Renaming a file.

ⓧ Directory structure:-

=> Single Level Directory

=> Two " "

=> Tree -structured "

=> Acyclic graph "

=> General " "

FILE CONCEPT

A System Stores datao on Various Storage devices. The OS however, for the convenience of use of data on these devices provides a uniform logical view of the data Storage to the users. The operating System abstracts from the physical properties of its Storage to the users, and defines a logical Storage unit known as a file.

A file is a collection of related data Stored as a named unit on the Secondary Storage. It can Store different types of data, Such as text, graphics, database, executable code, Sound, Videos, etc...,

File Attributes

A file in a System is identified by its name. The file name helps a user to locate a Specific file in the System. Different OS follows different file naming conventions. However, most OS accept a file name as a String of Characters or numbers or Some Special Symbols as well. Apart from the file name, some additional information is also associated with each file. The file attributes related to a file may vary in different OS.

Some of the Common file attributes are as follows:

- Name
- Size
- Type
- Identifier
- Location
- Date and Time
- protection

File operations

File operations are the functions that can be performed on a file. The various operations that can be performed on a file are

- Create a file
- Open
- write to a file
- Read a file
- Seek file
- Close file
- Delete file
- Append file
- Rename file

File Types

The file name is devided into two parts, with the two parts seperated by a period ('.') Symbol, where the first part is the name and Second part after the period i the file extension.

File Types and Extensions

Archives. - arc, zip, tar.

Batch - bat, sh

Backup file - bak, bkf

Executable - exe, com, bin

Lib - lib, a, so, dll

Image - bmp, jpeg, gif, jfif, dib

object - obj, o

Text - txt, doc

word processor - wp, txt, rtf, doc

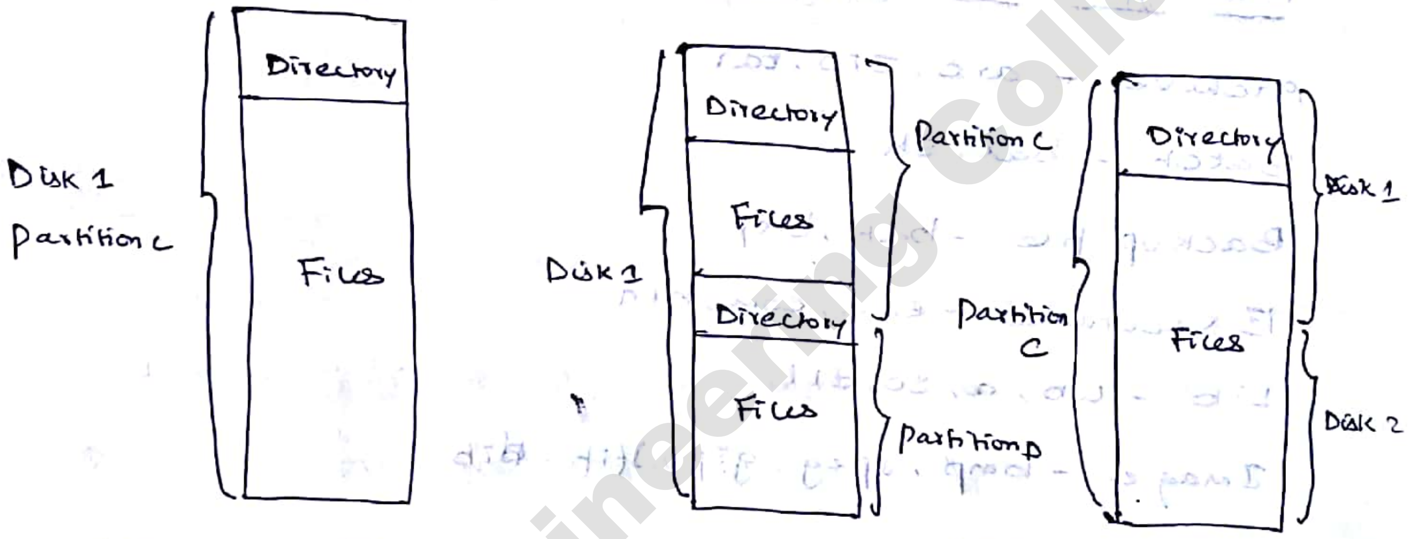
File structure

The file structure refers to the internal Structure of the file, that is, how a file is internally Stored in the System. The most Common file structures recognized and enforced by different OS are as follows

- Byte Sequence
- Record Sequence

DIRECTORY STRUCTURE

A Computer Stores numerous data on disk. To Manage this data, the disk is divided into one or more partitions and each partition contains information about the files stored in it. This information is stored in a director.



(a) Single disk Single partition

(b) Single disk multi partition

(c) multi disk Single partition

Various File System organization Schemes

Different operations that can be performed on a directory as follow

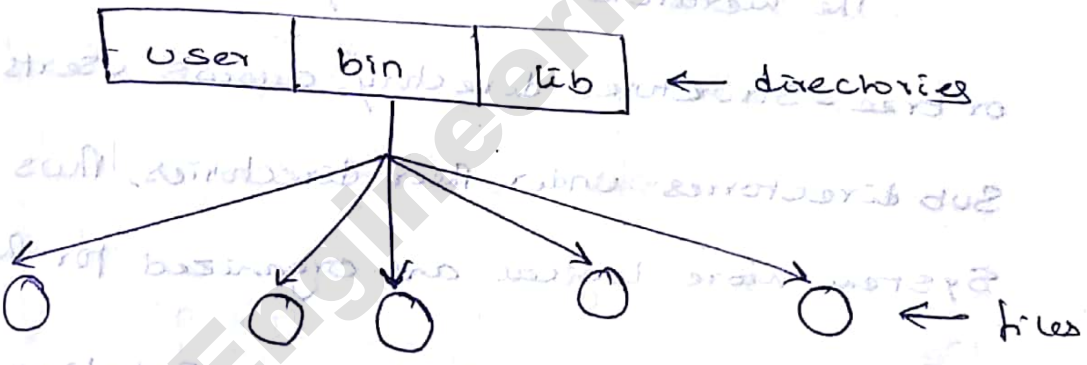
- Create a file
- Search a file
- List a directory
- Rename a file
- Delete a file

There are various Schemes to define the Structure of a directory. The most commonly used Schemes are as follows:

- Single level directory
- Two level directory
- Hierarchical directory

Single level Directory

is the Simplest directory Structure. There is only one directory that holds all the files. Sometimes this directory is referred to as root directory.

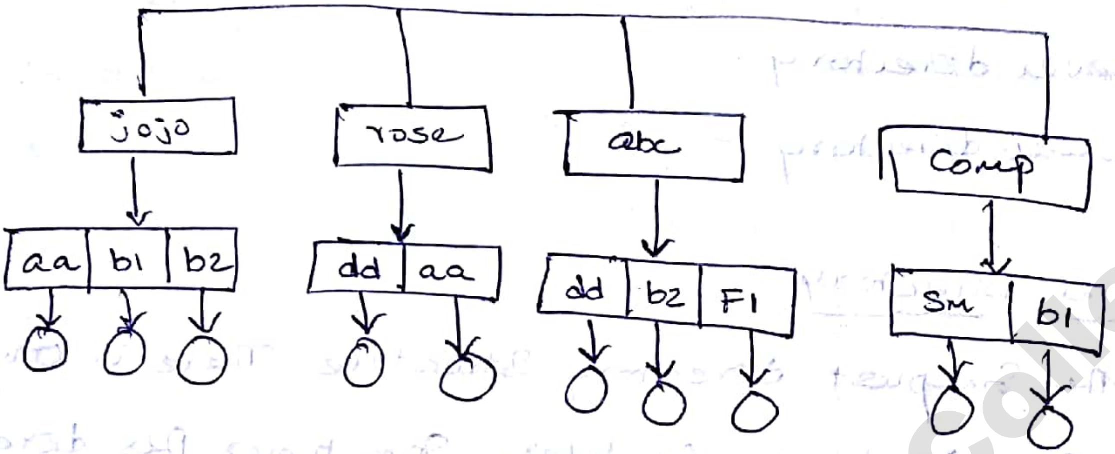


The main drawback of this System is that no two files can have the Same name.

Two - level Directory

In a two - level directory Structure, a Seperate directory known as User File Directory (UFD) is Created for each user.

Unlike Single-level directory structure, only the file names in a directory should be unique.



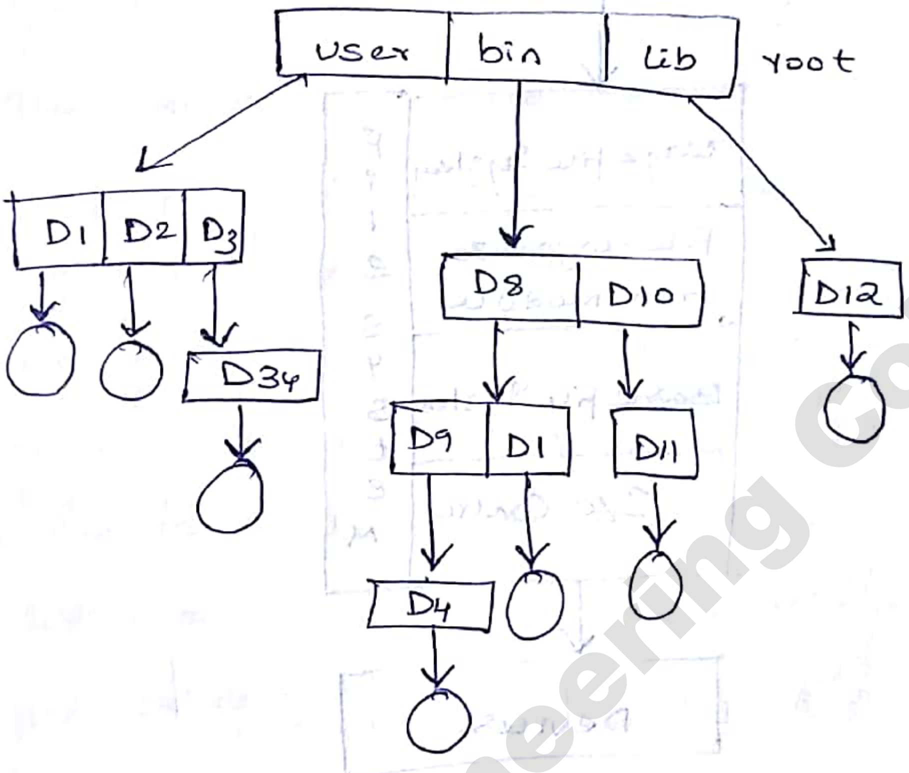
Hierarchial Directory

The hierarchial directory, also known as tree of directory or tree-structured directory, allows users to have sub directories under their directories, thus making the file system more logical and organized for the user.

Further, he wants to define a subdirectory which states the kind of furniture available under each type, say: sofa, bed, table, chair, etc.

All the hierarchial directory structure has the root directory at the higher level, which is the parent directory for all directories and sub directories. The root directory generally consist of system library files.

All files or directories at the lower levels are called child directories and a directory with no files or subdirectory is called a leaf.

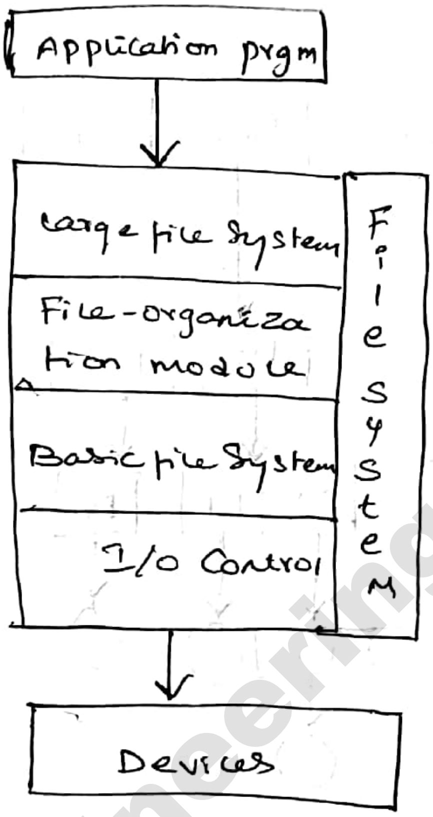


Hierarchical Directory Structure

FILE SYSTEM STRUCTURE

Every operating System imposes a file System that helps to organize, manage and retrieve data on the disk. The file System resides permanently on the disk. The design of the file system involves two key issues.

The first issue includes defining a file and its attributes operations that can be performed on a file, and the directory Structure. The second issue includes creating data structures and algorithms for mapping the logical file system onto the secondary storage devices.



File system Layers

Fig. Shows the file system is made up of different layers, where each layer represents a level. Each level uses the features of the lower levels to create new features that are used by higher levels.

The next component in the file system is the basic file system that issues generic commands to the appropriate device driver to read and write physical block on the disk.

The component at the next level of the file system is the file-organization module that organizes the files. It knows the physical block address and logical block address, allocation method and location of a file.

The logical file system at the next level manages all the information about a file except the actual data. It manages the directory structure to provide the necessary information to the file organization ~~memo~~ module when the file name is given. It maintains file structure using the File control Block (FCB) that stores information about a file, such as ownership, permissions and location of the file content.

File System Implementation

There are several on-disk and in-memory structures that are used to implement a file system. Depending on the operating system and the file system, these may vary, but the general principles remain same. Many of the structures that are used by most of the operating systems are discussed here!

The ondisk structure include;

- Boot control block
- partition control block

Further, each partition has a directory structure, with root directory at the top.

The directory structure helps to manage and organize the files in the file system. On creation of a new file, a new FCB is allocated that stores info, such as file permissions, ownership, size and location of the data blocks.

In UFS this is called the i-node.

Boot block	Super block	Free Spale mgnt	i-nodes	Root directory	Files and Directori

The in-memory structure helps in improving performance of the file system.

The in-memory structure include:

- In-memory partition table
- In-memory directory structure
- System-wide open-file table

- per-process open file table

Allocation methods

An important function of the file system is to manage the space on the secondary storage. It includes keeping track of the number of disk blocks allocated to files and the free blocks available for allocation.

The main issues related to disk space allocation are

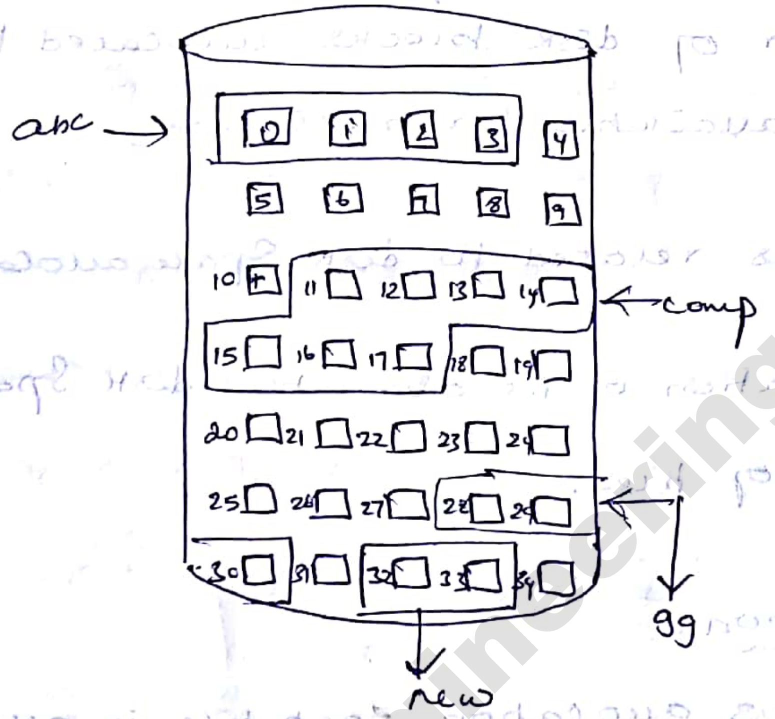
- optimum utilization of the available disk space
- Fast accessing of files.

- Contiguous Allocation.

In contiguous allocation, each file is allocated contiguous blocks on the disk, that is one after the other. It is relatively simple to implement the file system using contiguous allocation method. The directory entry for each file contains the file ~~ sa~~ name, the disk address of the first block and the total size of the file.

Contiguous allocation supports both sequential and direct access to a file. For sequential access, the file system remembers the disk address of the last block referenced and when required, reads the next block.

For direct address to block b of a file that starts at location L, the block L+b can be accessed immediately



Directory

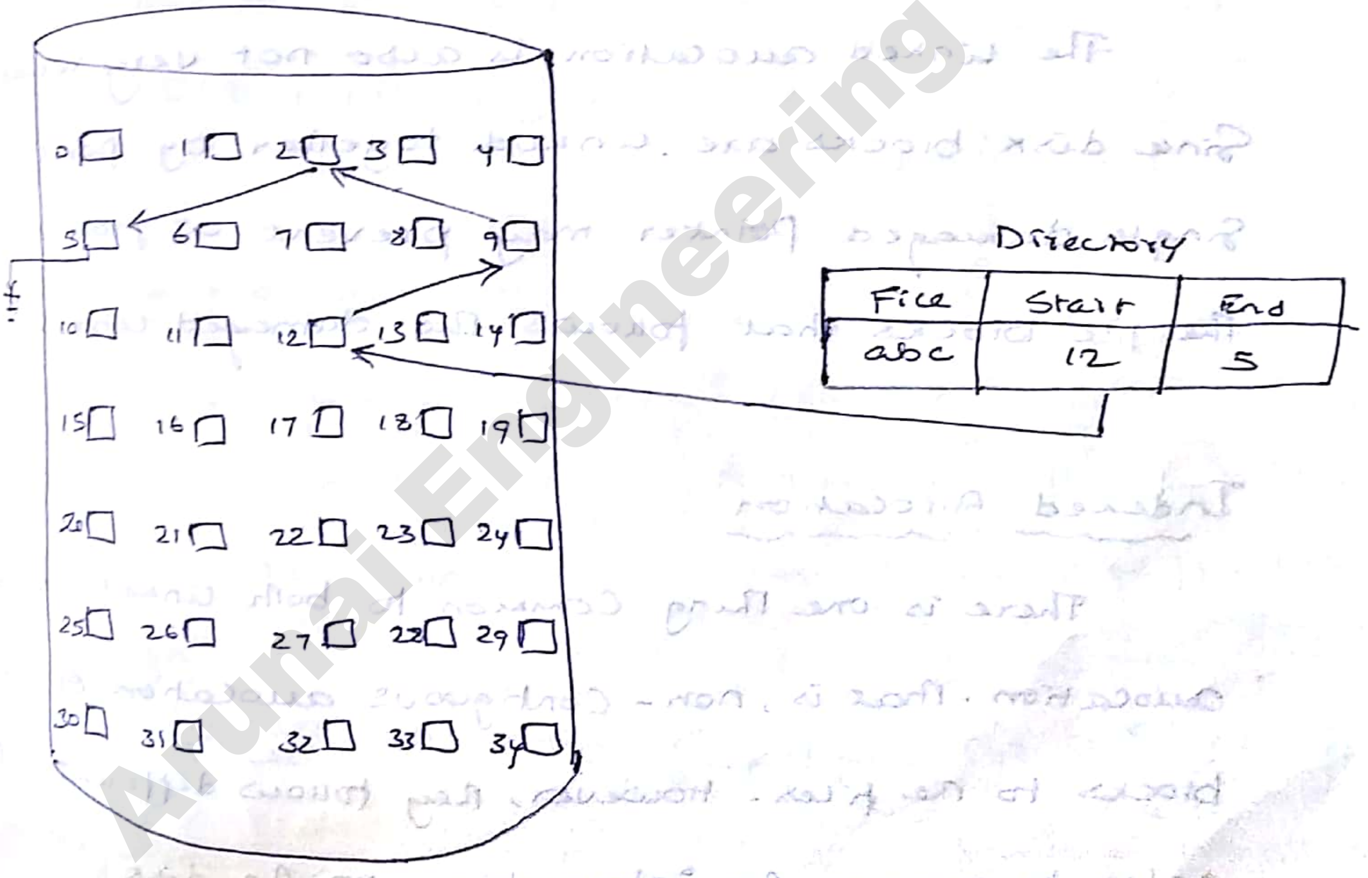
File name	Start address	length
abc	0	4
comp	11	7
99	28	3
new	32	2

Contiguous allocation has a significant problem of external fragmentation. One solution to this problem is compaction. An alternative to expensive compaction is to reuse the space.

Linked Allocation

The file size generally tends to change over time. The contiguous allocation of such files results in the several problems. Linked allocation method overcomes all the problems of contiguous allocation method.

The directory entry contains the file name, and the address of the first and the last blocks of the file



File	Start	End
abc	12	5

An Example of Linked Allocation

A total of four disk blocks are allocated to the file. The directory entry indicates that the file starts at block 12. It

Then continues at block 9, block 2 and finally ends at block 5.

The main disadvantages of using linked allocation are the slow access speed, disk space utilization by pointers and low reliability of the system.

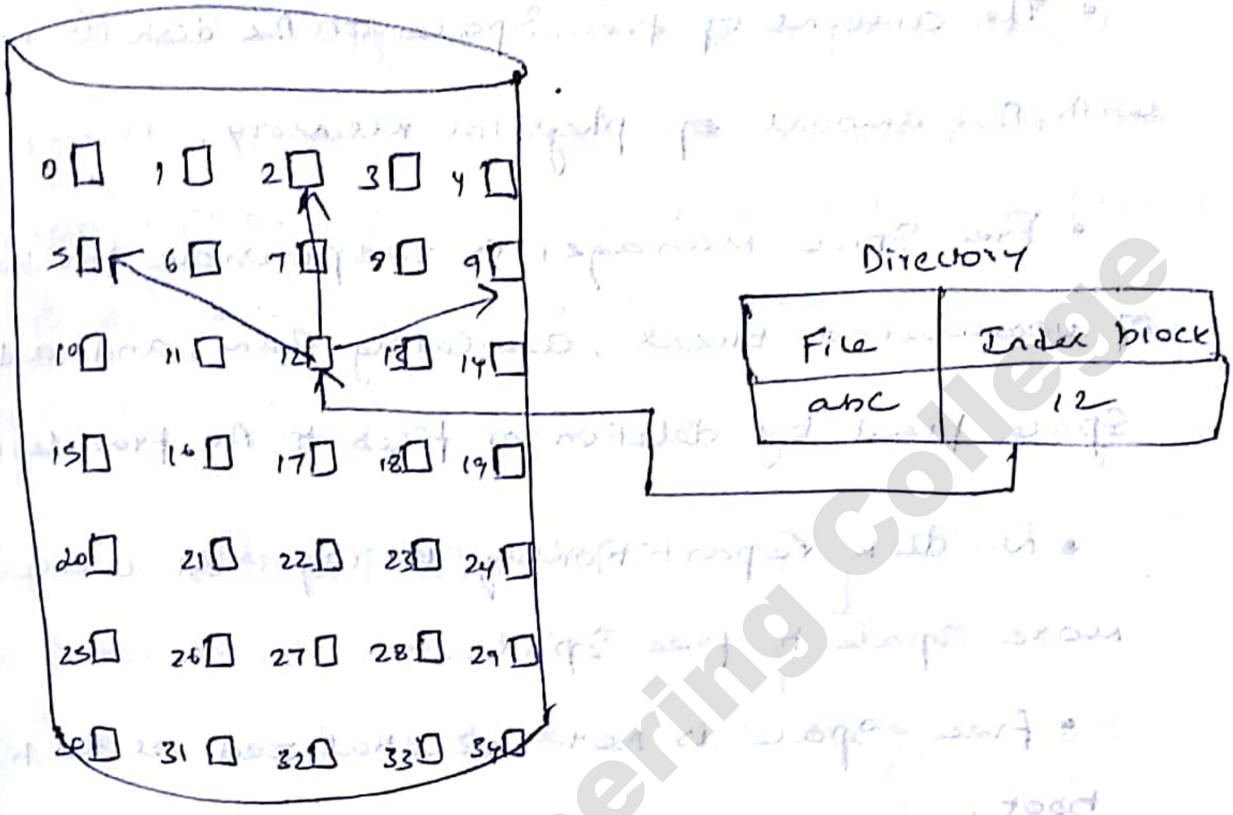
Contiguous blocks are grouped together as a cluster and allocation to files takes place as cluster rather than blocks.

The linked allocation is also not very reliable. Since disk blocks are linked together by pointers, a single damaged pointer may prevent us from accessing the file blocks that follows the damaged link.

Indexed Allocation

There is one thing common to both linked and indexed allocation, that is, non-contiguous allocation of disk blocks to the files. However, they follow different approaches to access the information on the disk.

Linked allocation supports sequential access, whereas, indexed allocation supports sequential as well as direct access.



Free space management

Free Space on the disk implies the Space that has not been allocated to any file or directory. The file System maintains a free space list to keep track of the free blocks on the disk.

To create a file, the free space list is searched for the required amount of space, and the space is then allocated to the new file. The newly allocated space is removed from the free space list. Similarly, when a file is deleted its space is added to the free space list.

Free Space management

- The amount of free Space on the disk is not Concerned with the amount of physical memory.

- Free Space Manager is responsible for keeping track of unallocated blocks, allocating them, and adding the Space freed by deletion of files to the free list.

- No disk repartitioning is required while adding more space to free Space.

- free - space is not reinitialized at the time of System boot.

I/O Systems

A computer System contains many I/o devices & Usually, most of the computer time is spent in performing I/O.

Since a Variety of I/o devices are attached to the System providing a device-independent interface is a major Challenge among OS designers.

I/O Hardware

Computer Communicates with a Variety of I/O devices ranging from mouse, keyboard and disk to highly specialized devices, Such as fighter plane Steering.

A device is attached with the computer System via a connection point known as port (as Serial or parallel port) A bus is a group of wires that specifies a set of messages that can be sent over it.

A device controller (or adapter) is an electronic component that can control one or more identical devices depending on the type of device controller.

Serial port controller is simple and controls the Signal Sent to the Serial port. on the other hand, SCSI controller is complex and can control multiple devices.

There are two approaches to let the communication b/w processor and controller occur.

- In the first approach, Special I/o instructions are used, which Specify the read or write Signal, I/o port address and cpu register.

The I/o port address helps in the Selection of Correct device.

- The Second approach is memory mapped I/o. In this registers of device are mapped into the address space of the processor and standard data-transfer instructions are used to perform read/write operation with the device.

Polling

A complete interaction b/w a host and a Controller may be complex, but the basic abstract model of interaction can be understood by a simple example.

Suppose that a host wishes to interact with a Controller and write some data through an I/o port. It is quite possible that Controller is busy in performing some other task.

When a host is in this waiting state, we say the host is busy-waiting or polling. To start the interaction, the host continues to check the busy bit until the bit becomes clear.

When controller notices that the ready bit is set it starts reading the data-out register to get the byte and writes it to the device.

Interrupt - driven I/O

The above scheme for performing interaction b/w host and controller is not always feasible, since it requires busy-waiting for host.

When either controller or device is slow, this waiting time may be long. In that scenario, host must switch to another task.

However, if host switches to another task and stops checking the busy bit, then how would it come to know that the controller has become free.

One solution to this problem is that the host must check the busy bit periodically and determine the status of controller.

This solution is however is not feasible because in many cases host must service the device continoously otherwise the data may be lost

Another solution is to arrange the hardware with which a controller can inform the CPU that it has finished the work given to it.

This mechanism of informing the cpu about completion of a task is Called interrupt.

The interrupt mechanism eliminates the need of busy-waiting of processor and hence considered more efficient than the previous one.

Cpus have two interrupt - request lines:

(i) maskable and

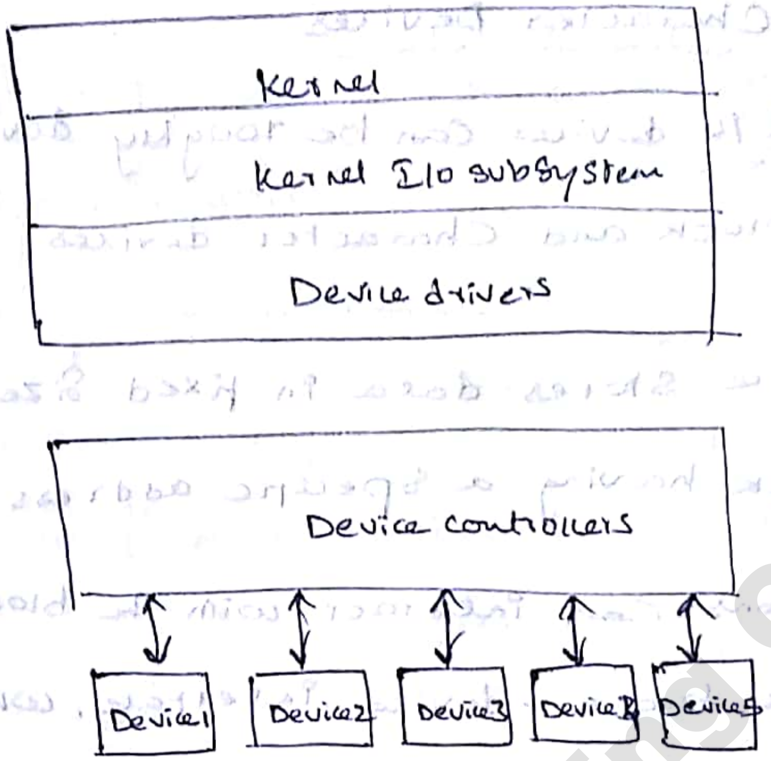
(ii) non-maskable interrupts

Maskable interrupts are used by device controllers and can be disabled by cpu wherever required but Non-maskable interrupts handle exceptions and Should not be disabled.

Application I/O interface

There is a variety of I/o devices that can be attached with the computer System and the operating System has to deal with all these devices.

However, it is almost impossible that operating System developer would write separate code to handle every distinct I/o device.



Layers in the Kernel I/o Structure

Clearly, this is not a feasible Solution. Instead I/o devices are grouped under a few general kinds. For each general kind, a standardized set of functions (called interface) is designed through which the device can be accessed.

The differences among the I/o devices are encapsulated into the kernel modules called device drivers. Not with this structure implemented, the I/o subsystem becomes independent of the hardware.

Thus, the device can be accessed through one of the standard interfaces and independent of the device itself.

Block and Character Devices

All I/O devices can be roughly divided into two classes: block and character devices.

Block device stores data in fixed size blocks with each block having a specific address.

Applications can interact with the block devices through the block-device interface, which supports the following basic system calls.

- read() : To read from the device
- write() : To write to the device
- seek() : To specify the next block to be accessed.

Character device is the one that accepts and produce a stream of characters. Unlike block devices, character devices are not addressable. The data transfer to/from them is performed in units of bytes.

Applications can interact with a character device through the character-stream interface, which supports the following basic system calls.

- get () : To read a character from the device
- put () : To write a character to the device.

— x —

UNIT V

CASE STUDY

Linux System - Basic Concepts ; System Administration Requirements for Linux System Administrator ; Setting up a LINUX Multifunction Server, Domain Name System, Setting up Local Network Services; Virtualization - Basic Concepts, Setting up xen, VMware on Linux Host and Adding Guest OS.

Linux

Linux is an operating System, just like windows and Mac OS x. AS an OS, Linux manages your h/w and provides Services your other software needs to run.

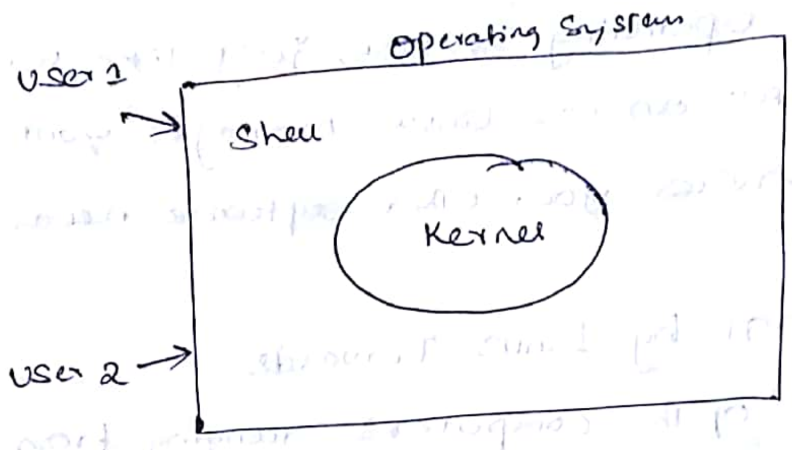
- ✓ Developed in 1991 by Linus Torvalds
- ✓ used in most of the Computers, ranging from Super Computers to embedded System
- ✓ Multi user
- ✓ Multi tasking
- ✓ Time Sharing
- ✓ Monolithic Kernel
- ✓ Latest stable version of Linux Kernel - 2.6.28, released on 24-Dec-2008

- Founded by Richard Stallman in 1983
- Organisation that started developing copylefted programs
- Project - GNU Project
  - GNU Not Unix
  - Recursive expansion

Linux Distributions

- Redhat
- Fedora

Operating System



Main components of Linux operating system

GUI
Gnome KDE

LAMP:
Apache
PHP
MySQL

Net:
sshd
inetd

gcc GNU Coreutils bash

GNU C Library Other Libraries

SCI device files Linux kernel Sockets
Processes file systems protocols
memory management drivers and modules

Computer H/W

- Debian
- Novell's SUSE Linux
- Ubuntu
- Mandrake
- Knoppix and more

GNU / Linux

- Only the kernel is called by the name Linux
- The rest are the tools developed under GNU project
- Hence the name GNU / Linux

Requirements for a Linux System Administrator

Setting up a Linux Multifunction Server

What Systems Administrators do

Systems Administration is an old responsibility gaining new found importance and acceptance as a profession. It has come into existence because of the increasing complexity of modern computer systems and networks and because of the economy's increasing reliance on computers.

It can be said the A System administrators have two basic reasons for being

- ensuring that the computing systems runs correctly and as efficiently as possible and
- ensuring that all users can and do use the computing system to carry out their required work

in the easiest and most efficient manner

What a Systems Administrator needs to do Come from a number of categories including:

* Users
* hardware and
* Support

Users

Some of the Characteristics of people that can Contribute to your job include:

- How many users are there?
- The level of the user's expertise
- what are the users trying to do
- Are they responsible or irresponsible
- who do the users know?

Hardware / Software

Some Consideration includes!

- How many, how big and how Complex?
- Is there a network
- Are the Computers heterogenous or homogenous

System administrator requirements?

One of the most important aims for a system administrator is to be pro-active rather than reactive.

Important components of administration an include

* documentation

Both for yourself, the users and management.

* Time management.

This is an essential ability for a Systems Administrator who must balance a small amount of time between a large number of simultaneous tasks.

* policy.

There must be policy on just about everything at a site. Having policies that have been accepted by management and hopefully the users is essential.

* Self-education.

Computing is always changing. A Systems Administrator must keep up with the pack.

* planning.

What are the aims for your site and yourself for the next 12 months?

- automation, and

Anything that can be should be automated. It makes your job easier and gives you more time to do other things

- financial planning and management.

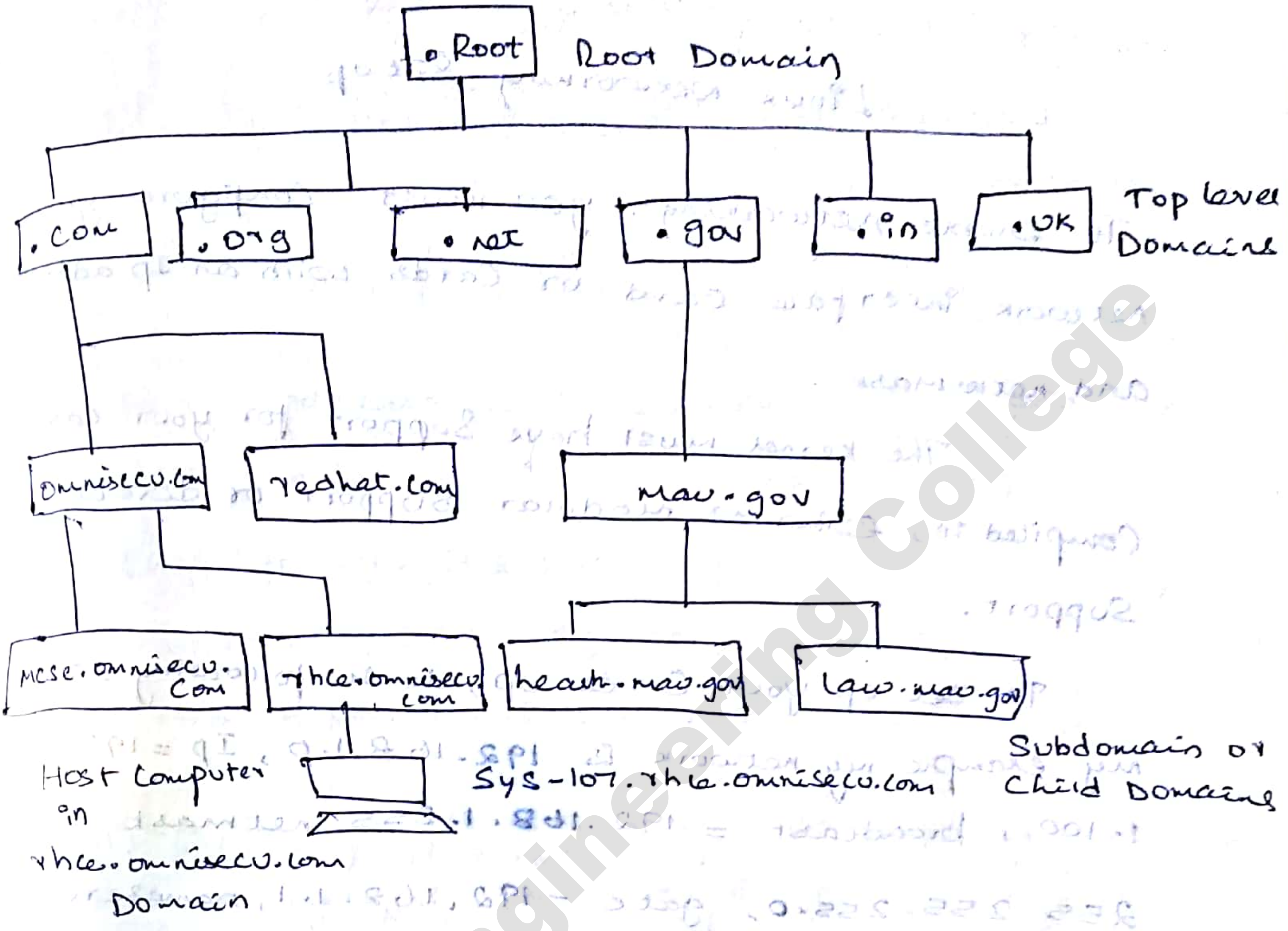
Domain Name System (DNS) in Linux

A DNS or name Server, is used to resolve an IP address to a hostname or vice versa. DNS naming system is a hierarchical and logical tree Stru-cture called the DNS namespace.

The DNS namespace has a unique root that can have any number of Subdomains. In turn, each Subdomain can have more subdomains.

The root in the internet namespace has many top-level domain names, one of which is com. The domain .com can have a Subdomain for example omnisecu.com, can have further subdomains like mcse.omnisecu.com and rhce.omnisecu.com.

organizations can also create private network and use their own private DNS namespaces that are not visible on the Internet.



Features of Domain Name System (DNS)

- Domain Name System (DNS) provides name resolution services to the client computers.
- DNS provides Ip address to hostname mapping
- Linux DNS Servers can operate in three modes and these modes are, catching only DNS server, primary DNS server and slave (secondary) DNS server.

- Replication of DNS database b/w Servers.

Linux Networking Setup

To enable networking, you must configure your network interface card or cards with an Ip address and netmask.

The kernel must have support for your cards compiled in, either as modular support or direct support.

To see up your Cards up, do the following. In my example my network is 192.168.1.0, Ip = 192.168.1.100, broadcast = 192.168.1.255, netmask = 255.255.255.0, gate - 192.168.1.1, nameserver: 192.168.1.10

1. Determine your machines Ip address from your network administrator.

2. your network mask. This determines which portion of the Ip address specifies the Subnetwork number and which portion specifies the host

Class C (most networks) 255.255.255.0

Class B 255.255.0.0

3. your network address which is your Ip address bit wise and with the network mask.

Ex: Ip : 192.168.1.100

Mask : 255.255.255.0

Net Addr :: 192.168.1.0

4. your broadcast address, used to broadcast packets to every machine on your subnet.

Ex: Ip : 192.168.1.100

Mask : 255.255.255.0

Net Addr : 192.168.1.255

5. your gateway address: The address of the machine that is your gateway to ~~be~~ the outside world.

Ip : 192.168.1.100

Gateway : 192.168.1.1

6. your nameserver address ; Translates host names into Ip addresses : 192.168.1.10.

Evolution of operating System

Serial processing

- Computers from 1940 to 1955 were able to perfor

Serial processing

- programs for the machine (relays, vaccum tubes) written in assembly language
- Console with display lights, switches, punch card reader / plugboard, printer
- Re-wiring or punch card reading necessary for filling the program memory
- program had complete control of the machine f the entire run time
- Manual reservation, user either finished early (wast time) or couldn't debug their problem
- long setup time

Job may involve running the compiler progra first and feeding in the output again

- IBM 1401 - October 5th, 1959
  - IBM's first affordable general purpose computer, ie for accounting
  - 1401 processing Unit, 1402 Card Read punch (250 cards / minute), printer

VMware Workstation

VMware Workstation includes the ability to designate multiple virtual machines as a team which can then be powered on, powered off, suspended or resumed as a single object, making it particularly useful for testing client-server environments.

VMWare Player

The VMware Player, a virtualization package of basically similar, but reduced, functionality, is also available, and is free of charge for non-commercial use, or for distribution or other use by written agreement.

VMware Player is a virtualization software package supplied free of charge by VMware, Inc. VMware Player can run existing virtual appliances and create its own virtual machines. It uses the same virtualization core as VMware Workstation, a similar program with more features, but not free of charge. VMware Player is available for personal non-commercial use, or for distribution or other use by written agreement.

VMware claims the Player offers better graphics, faster performance, and tighter integration for running Windows XP under Windows Vista or Windows 7 than Microsoft's Windows XP Mode running on Windows Virtual PC, which is free of charge for all purposes.

VMware Tools

VMware Tools is a package with drivers and other software that can be installed in guest operating systems to increase their performance. It has several components, including the following drivers for the emulated hardware:

- VESA-compliant graphics for the guest machine to access high screen resolutions
- Network drivers for the vmxnet2 and vmxnet3 NIC

- Mouse integration, Drag-and-drop file support
- Clipboard sharing between host and guest
- Time synchronization capabilities (guest syncs with host machine's clock)
- Support for Unity, a feature that allows seamless integration of applications with the host desktop

Installing and Configuring VMWare

1. Download VMware Server 2. VMware management console on a remote Ubuntu desktop behind a firewall at a remote location. Run the following command:

   ```
   $gksu vmware-server-console
   ```

2. Install the VMware Server 2.0.2 rpm as shown below.

   ```
        # rpm -ivh VMware-server-2.0.2-203138.i386.rpm
   Preparing...
        1:VMware-server
   ########################################### [100%]
   ```

   The installation of VMware Server 2.0.2 for Linux completed successfully.

   You can decide to remove this software from your system at any time by invoking the following command:

   ```
   rpm -e VMware-server
   ```

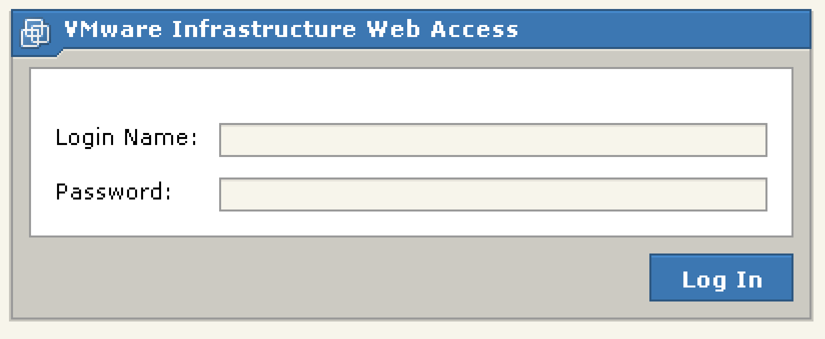
   Before running VMware Server for the first time, you need to configure it for your running kernel by invoking the following command:

   ```
   /usr/bin/vmware-config.pl
   ```

3. Configure VMware Server 2 using *vmware-config.pl*. Execute the vmware-config.pl as shown below. Accept default values for everything. Partial output of the vmware-config.pl is shown below.

   ```
   # /usr/bin/vmware-config.pl
   ```

4. Go to VMware Infrastructure Webaccess. Go to **https://{host-os-ip}:8333/ui** to access the VMware Infrastructure web access console.



VMware Web Access Login

Installing a VMware Guest OS

1. **Start VMware Workstation**

   Windows host: Double-click the VMware Workstation icon on your desktop or use the Start menu (Start > Programs > VMware > VMware Workstation).

   Linux host: In a terminal window, enter the command

   ```
   vmware &
   ```

2. **Start the New Virtual Machine Wizard**

   When you start VMware Workstation, you can open an existing virtual machine or create a new one. Choose File > New > Virtual Machine to begin creating your virtual machine.

3. Select the method you want to use for configuring your virtual machine

   If you select ***Typical***, the wizard prompts you to specify or accept defaults for the following choices:

   - The guest operating system
   - The virtual machine name and the location of the virtual machine's files
   - The network connection type
   - Whether to allocate all the space for a virtual disk at the time you create it
   - Whether to split a virtual disk into 2GB file

If you select ***Custom***, the wizard prompts you to specify or accept defaults for the following choices:

- Make a legacy virtual machine that is compatible with Workstation 4.x, GSX Server 3.x, ESX Server 2.x and VMware ACE 1.x.
- Use an IDE virtual disk for a guest operating system that would otherwise have a SCSI virtual disk created by default
- Use a physical disk rather than a virtual disk and Set memory options that are different from the defaults

If you selected **Custom** as your configuration path, you may adjust the memory settings or accept the defaults, then click Next to continue.

6. Configure the networking capabilities of the virtual machine.

   If you selected *Typical* as your configuration path, click Finish and the wizard sets up the files needed for your virtual machine.

   If you selected *Custom* as your configuration path, continue with the steps below to configure a disk for your virtual machine.

7. Select whether to create an IDE or SCSI disk and specify the capacity of the virtual disk.

8. Click Finish. The wizard sets up the files needed for your virtual machine.

Setting up a XEN Workstation XEN Workstation

Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.

The University of Cambridge Computer Laboratory developed the first versions of Xen. The Xen community develops and maintains Xen as free and open-source software, subject to the requirements of the GNU General Public License (GPL), version 2. Xen is currently available for the IA-32, x86-64 and ARM instruction sets.

XenServer runs directly on server hardware without requiring an underlying operating system, which results in an efficient and scalable system. XenServer works by abstracting elements from the physical machine (such as hard drives, resources and ports) and allocating

XEN Environment

Responsibilities of the hypervisor include memory management and CPU scheduling of all virtual machines, and for launching the most privileged domain - the only virtual machine which by default has direct access to hardware. From the dom0 the hypervisor can be managed and unprivileged domains can be launched.

Benefits of Using XenServer

1. **Using XenServer reduces costs by:**
   - Consolidating multiple VMs onto physical servers
   - Reducing the number of separate disk images that need to be managed
   - Allowing for easy integration with existing networking and storage infrastructures
2. **Using XenServer increases flexibility by:**
   - Allowing you to schedule zero downtime maintenance by using XenMotion to live migrate VMs between XenServer hosts
   - Increasing availability of VMs by using High Availability to configure policies that restart VMs on another XenServer host if one fails

- Increasing portability of VM images, as one VM image will work on a range of deployment infrastructures

Administering XenServer

- There are two methods by which to administer XenServer: XenCenter and the XenServer Command-Line Interface (CLI).
- **XenCenter** is a graphical, Windows-based user interface. XenCenter allows you to manage XenServer hosts, pools and shared storage, and to deploy, manage and monitor VMs from your Windows desktop machine.
- The XenCenter on-line Help is a useful resource for getting started with XenCenter and for context-sensitive assistance.

Installing and Configuring XenServer

1. Type the following command to get information about xen server package

   ```
   # yum info xen
   ```

2. Run the system-config-securitylevel program or edit /etc/selinux/config to looks as follows:

   ```
   SELINUX=Disabled
   SELINUXTYPE=targeted
   ```

   If you changed the SELINUX value from enforcing, you'll need to reboot Fedora before proceeding.

3. This command will install the Xen hypervisor, a Xen-modified Fedora kernel called *domain 0*, and various utilities:

   ```
   # yum install kernel-xen0
   ```

4. To make the Xen kernel the default, change this line:

   ```
   default=1
   ```

   to

   ```
   default=0
   ```

5. Now you can reboot. Xen should start automatically, but let's check:

   ```
   # /usr/sbin/xm list
   Name        ID    Mem(MiB)    VCPUs     State     Time(s)
   Domain-0    0     880         1         r-----    20.5
   ```

   The output should show that Domain-0 is running. Domain 0 controls all the guest operating systems that run on the processor, similarly to how the kernel controls processes in an operating system.

Installing a Xen Guest OS from the Command-line

1. **Preparing the System for virt-install**

   Fedora Linux does not install VNC by default. To verify whether VNC is installed, run the following command from a Terminal Window:

If rpm reports that VNC is not installed, it may be installed from root as follows:

```
yum    install   vnc
```

2. **Running virt-install to Build the Xen Guest System**

   virt-install must be run as root and, once invoked, will ask a number of questions before creating the guest system. The question are as follows:

   i. *What is the name of your virtual machine and install location?*
   ii. *How much RAM should be allocated (in megabytes)?*
   iii. *What would you like to use as the disk (path)?*
   iv. *Would you like to enable graphics support? (yes or no)*

   The following transcript shows a typical virt-install session:

```
# virt-install
```

3. Once the guest system has been created, the vncviewer screen will appear containing the operating system installer:

Installing a Xen Guest OS (Fedora Core 5)

1. Fedora Core 5 has a Xen guest installation script that simplifies the process, although it installs only FC5 guests. The script expects to access the FC5 install tree via FTP, the Web, or NFS; for some reason, you can't specify a directory or file.

```
# mkdir /var/www/html/dvd
# mount -t iso9660 /dev/dvd /var/www/html/dvd
# apachectl start
```

   Now we'll run the installation script and answer its questions:

```
# xenguest-install.py
```

2. Xen does not start the guest operating system automatically. You need to type this command on the host:
3. To prove that both servers are running, try these commands:

```
# xm list
# xentop
```

4. To start Xen domains automatically, use these commands:

```
# /sbin/chkconfig --level 345 xendomains on
# /sbin/service xendomains start
```

5. To Edit A Xen Guest Configuration File, Which Is A Text File (Actually, A Python Script) In The /Etc/Xen Directory.

```
      # man xmdomain.cfg
And edit as follows,
# Automatically generated Xen config file
name = "guest1"
memory = "256"
```

```
disk = [ 'file:/xenguest,xvda,w' ]
vif = [ 'mac=00:16:3e:63:c7:76' ]
uuid = "bc2c1684-c057-99ea-962b-de44a038bbda"
bootloader="/usr/bin/pygrub"
on_reboot = 'restart'
on_crash = 'restart'
```

6. Once you have a guest configuration file, create the Xen guest with this command:

```
# xm create -c guest_name
```

where

guest_name can be a full pathname or a relative filename (in which case Xen places it in /etc/xen/guest_name).

Xen will create the guest domain and try to boot it from the given file or device. The **-c** option attaches a console to the domain when it starts, so you can answer the installation questions that appear **.**

iPhone OS becomes iOS

Prior to the release of the iPad in 2010, the operating system running on the iPhone was generally referred to as iPhone OS. Given that the operating system used for the iPad is essentially the same as that on the iPhone it didn‟t make much sense to name it iPad OS. Instead, Apple decided to adopt a more generic and non-device specific name for the operating system. Given Apple‟s predilection for names prefixed with the letter „i‟ (iTunes, iBookstore, iMac etc) the logical choice was, of course, iOS. Unfortunately, iOS is also the name used by Cisco for the operating system on its routers (Apple, it seems, also has a predilection for ignoring trademarks). When performing an internet search for iOS, therefore, be prepared to see large numbers of results for Cisco‟s iOS which have absolutely nothing to do with Apple‟s iOS.

<u>An Overview of the iOS 6 Architecture</u>

iOS consists of a number of different software layers, each of which provides programming frameworks for the development of applications that run on top of the underlying hardware.

These operating system layers can be presented diagrammatically as illustrated in Figure

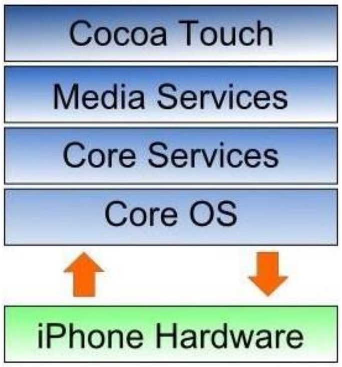


Figure: iOS 6 Architecture

Some diagrams designed to graphically depict the iOS software stack show an additional box positioned above the Cocoa Touch layer to indicate the applications running on the device. In the above diagram we have not done so since this would suggest that the only interface available to the app is Cocoa Touch. In practice, an app can directly call down any of the layers of the stack to perform tasks on the physical device.

That said, however, each operating system layer provides an increasing level of abstraction away from the complexity of working with the hardware. 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. In general, the higher level of layer you program to, the less effort and fewer lines of code you will have to write to achieve your objective. And as any veteran programmer will tell you, the less code you have to write the less opportunity you have to introduce bugs.

The Cocoa Touch Layer

The Cocoa Touch layer sits at the top of the iOS stack and contains the frameworks that are most commonly used by iPhone application developers. Cocoa Touch is primarily written in Objective-C, is based on the standard Mac OS X Cocoa API (as found on Apple desktop and laptop computers) and has been extended and modified to meet the needs of the iPhone hardware.

The Cocoa Touch layer provides the following frameworks for iPhone app development:

UIKit Framework (UIKit.framework)

The UIKit framework is a vast and feature rich Objective-C based programming interface. It is, without question, the framework with which you will spend most of your time working. Entire books could, and probably will, be written about the UIKit framework alone. Some of the key features of UIKit are as follows:

- User interface creation and management (text fields, buttons, labels, colors, fonts etc)
- Application lifecycle management
- Application event handling (e.g. touch screen user interaction)
- Multitasking
- Wireless Printing
- Data protection via encryption
- Cut, copy, and paste functionality

- Web and text content presentation and management
- Data handling
- Inter-application integration
- Push notification in conjunction with Push Notification Service
- Local notifications (a mechanism whereby an application running in the background can gain the user‟s attention)
- Accessibility
- Accelerometer, battery, proximity sensor, camera and photo library interaction
- Touch screen gesture recognition
- File sharing (the ability to make application files stored on the device available via iTunes)
- Blue tooth based peer to peer connectivity between devices
- Connection to external displays

Map Kit Framework (MapKit.framework)

If you have spent any appreciable time with an iPhone then the chances are you have needed to use the Maps application more than once, either to get a map of a specific area or to generate driving directions to get you to your intended destination. The Map Kit framework provides a programming interface which enables you to build map based capabilities into your own applications. This allows you to, amongst other things, display scrollable maps for any location, display the map corresponding to the current geographical location of the device and annotate the map in a variety of ways.

Push Notification Service

The Push Notification Service allows applications to notify users of an event even when the application is not currently running on the device. Since the introduction of this service it has most commonly been used by news based applications. Typically when there is breaking news the service will generate a message on the device with the news headline and provide the user the option to load the corresponding news app to read more details. This alert is typically accompanied by an audio alert and vibration of the device. This feature should be used sparingly to avoid annoying the user with frequent interruptions.

Message UI Framework (MessageUI.framework)

The Message UI framework provides everything you need to allow users to compose and send email messages from within your application. In fact, the framework even provides the user interface elements through which the user enters the email addressing information and message content. Alternatively, this information may be pre-defined within your application and then displayed for the user to edit and approve prior to sending.

Address Book UI Framework (AddressUI.framework)

Given that a key function of the iPhone is as a communications device and digital assistant it should not come as too much of a surprise that an entire framework is dedicated to the integration of the address book data into your own applications. The primary purpose of the framework is to enable you to access, display, edit and enter contact information from the iPhone address book from within your own application.

Game Kit Framework (GameKit.framework)

The Game Kit framework provides peer-to-peer connectivity and voice communication between multiple devices and users allowing those running the same app to interact. When this feature was first introduced it was anticipated by Apple that it would primarily be used in multi-player games (hence the choice of name) but the possible applications for this feature clearly extend far beyond games development.

iAd Framework (iAd.framework)

The purpose of the iAd Framework is to allow developers to include banner advertising within their applications. All advertisements are served by Apple‟s own ad service.

Event Kit UI Framework (EventKit.framework)

The Event Kit UI framework was introduced in iOS 4 and is provided to allow the calendar and reminder events to be accessed and edited from within an application.

Accounts Framework (Accounts.framework)

iOS 5 introduced the concept of system accounts. These essentially allow the account information for other services to be stored on the iOS device and accessed from within application code. Currently system accounts are limited to Twitter accounts, though other services such as Facebook will likely appear in future iOS releases. The purpose of the Accounts Framework is to provide an API allowing applications to access and manage these system accounts.

Social Framework (Social.framework)

The Social Framework allows Twitter, Facebook and Sina Weibo integration to be added to applications. The framework operates in conjunction the Accounts Framework to gain access to the user‟s social network account information.

The iOS Media Layer

The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities. As with the other layers comprising the iOS stack, the Media layer comprises a number of frameworks which may be utilized when developing iPhone apps. In this section we will look at each one in turn.

Core Video Framework (CoreVideo.framework)

The Core Video Framework provides buffering support for the Core Media framework. Whilst this may be utilized by application developers it is typically not necessary to use this framework.

Core Text Framework (CoreText.framework)

The iOS Core Text framework is a C-based API designed to ease the handling of advanced text layout and font rendering requirements.

Image I/O Framework (ImageIO.framework)

The Image I/O framework, the purpose of which is to facilitate the importing and exporting of image data and image metadata, was introduced in iOS 4. The framework supports a wide range of image formats including PNG, JPEG, TIFF and GIF.

Assets Library Framework (AssetsLibrary.framework)

The Assets Library provides a mechanism for locating and retrieving video and photo files located on the iPhone device. In addition to accessing existing images and videos, this framework also allows new photos and videos to be saved to the standard device photo album.

Core Graphics Framework (CoreGraphics.framework)

The iOS Core Graphics Framework (otherwise known as the Quartz 2D API) provides a lightweight two dimensional rendering engine. Features of this framework include PDF document creation and presentation, vector based drawing, transparent layers, path based drawing, anti-aliased rendering, color manipulation and management, image rendering and gradients. Those familiar with the Quartz 2D API running on MacOS X will be pleased to learn that the implementation of this API is the same on iOS.

Core Image Framework (CoreImage.framework)

A new framework introduced with iOS 5 providing a set of video and image filtering and manipulation capabilities for application developers.

Quartz Core Framework (QuartzCore.framework)

The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone. It provides the foundation for the majority of the visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.

OpenGL ES framework (OpenGLES.framework)

For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL. Originally developed by the now defunct Silicon Graphics, Inc (SGI) during the 1990s in the form of GL, the open version of this technology (OpenGL) is now under the care of a non-profit consortium comprising a number of major companies including Apple, Inc., Intel, Motorola and ARM Holdings.

OpenGL for Embedded Systems (ES) is a lightweight version of the full OpenGL specification designed specifically for smaller devices such as the iPhone. iOS 3 or later supports both OpenGL ES 1.1 and 2.0 on certain iPhone models (such as the iPhone 3GS and iPhone 4). Earlier versions of iOS and older device models support only OpenGL ES version 1.1.

GLKit Framework (GLKit.framework)

The GLKit framework is an Objective-C based API designed to ease the task of creating OpenGL ES based applications.

NewsstandKit Framework (NewsstandKit.framework)

The Newsstand application is a new feature of iOS 5 and is intended as a central location for users to gain access to newspapers and magazines. The NewsstandKit framework allows for the development of applications that utilize this new service.

iOS Audio Support

iOS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, µ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.

AV Foundation framework (AVFoundation.framework)

An Objective-C based framework designed to allow the playback, recording and management of audio content.

Core Audio Frameworks (CoreAudio.framework, AudioToolbox.framework and AudioUnit.framework)

The frameworks that comprise Core Audio for iOS define supported audio types, playback and recording of audio files and streams and also provide access to the device‟s built-in audio processing units.

Open Audio Library (OpenAL)

OpenAL is a cross platform technology used to provide high-quality, 3D audio effects (also referred to as positional audio). Positional audio may be used in a variety of applications though is typically used to provide sound effects in games.

Media Player Framework (MediaPlayer.framework)

The iOS Media Player framework is able to play video in .mov, .mp4, .m4v, and .3gp formats at a variety of compression standards, resolutions and frame rates.

Core Midi Framework (CoreMIDI.framework)

Introduced in iOS 4, the Core MIDI framework provides an API for applications to interact with MIDI compliant devices such as synthesizers and keyboards via the iPhone‟s dock connector.

The iOS Core Services Layer

The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built and consists of the following frameworks.

Address Book Framework (AddressBook.framework)

The Address Book framework provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.

CFNetwork Framework (CFNetwork.framework)

The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets. This enables application code to be written that works with HTTP, FTP and Domain Name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

Core Data Framework (CoreData.framework)

This framework is provided to ease the creation of data modeling and storage in Model-View-Controller (MVC) based applications. Use of the Core Data framework significantly reduces the amount of code that needs to be written to perform common tasks when working with structured data within an application.

Core Foundation Framework (CoreFoundation.framework)

The Core Foundation framework is a C-based Framework which provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication. Additional XML capabilities beyond those included with this framework are provided via the libXML2 library. Though this is a C-based interface, most of the capabilities of the Core Foundation framework are also available with Objective-C wrappers via the Foundation Framework.

Core Media Framework (CoreMedia.framework)

The Core Media framework is the lower level foundation upon which the AV Foundation layer is built. Whilst most audio and video tasks can, and indeed should, be performed using the higher level AV Foundation framework, access is also provided for situations where lower level control is required by the iOS application developer.

Core Telephony Framework (CoreTelephony.framework)

The iOS Core Telephony framework is provided to allow applications to interrogate the device for information about the current cell phone service provider and to receive notification of telephony related events.

EventKit Framework (EventKit.framework)

An API designed to provide applications with access to the calendar, reminders and alarms on the device.

Foundation Framework (Foundation.framework)

The Foundation framework is the standard Objective-C framework that will be familiar to those who have programmed in Objective-C on other platforms (most likely Mac OS X). Essentially, this consists of Objective-C wrappers around much of the C-based Core Foundation Framework.

Core Location Framework (CoreLocation.framework)

The Core Location framework allows you to obtain the current geographical location of the device (latitude, longitude and altitude) and compass readings from with your own applications. The method used by the device to provide coordinates will depend on the data available at the time the information is requested and the hardware support provided by the particular iPhone model on which the app is running (GPS and compass are only featured on recent models). This will either be based on GPS readings, Wi-Fi network data or cell tower triangulation (or some combination of the three).

Mobile Core Services Framework (MobileCoreServices.framework)

The iOS Mobile Core Services framework provides the foundation for Apple‟s Uniform Type Identifiers (UTI) mechanism, a system for specifying and identifying data types. A vast range of predefined identifiers have been defined by Apple including such diverse data types as text, RTF, HTML, JavaScript, PowerPoint .ppt files, PhotoShop images and MP3 files.

Store Kit Framework (StoreKit.framework)

The purpose of the Store Kit framework is to facilitate commerce transactions between your application and the Apple App Store. Prior to version 3.0 of iOS, it was only possible to charge a customer for an app at the point that they purchased it from the App Store. iOS 3.0 introduced the concept of the "in app purchase" whereby the user can be given the option to make additional payments from within the application. This might, for example, involve implementing a subscription model for an application, purchasing additional functionality or even buying a faster car for you to drive in a racing game. With the introduction of iOS 6, content associated with an in-app purchase can now be hosted on, and downloaded from, Apple‟s servers.

SQLite library

Allows for a lightweight, SQL based database to be created and manipulated from within your iPhone application.

System Configuration Framework (SystemConfiguration.framework)

The System Configuration framework allows applications to access the network configuration settings of the device to establish information about the “reachability” of the device (for example whether Wi-Fi or cell connectivity is active and whether and how traffic can be routed to a server).

Quick Look Framework (QuickLook.framework)

The Quick Look framework provides a useful mechanism for displaying previews of the contents of file types loaded onto the device (typically via an internet or network connection) for which the application does not already provide support. File format types supported by this framework include iWork, Microsoft Office document, Rich Text Format, Adobe PDF, Image files, public.text files and comma separated (CSV).

The iOS Core OS Layer

The Core OS Layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware. The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

Accelerate Framework (Accelerate.framework)

The Accelerate Framework provides a hardware optimized C-based API for performing complex and large number math, vector, digital signal processing (DSP) and image processing tasks and calculations.

External Accessory Framework (ExternalAccessory.framework)

Provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

Security Framework (Security.framework)

The iOS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, keychains, encryption, digests and Hash-based Message Authentication Code (HMAC).

System (LibSystem)

As we have previously mentioned, iOS is built upon a UNIX-like foundation. The System component of the Core OS Layer provides much the same functionality as any other UNIX like operating system. This layer includes the operating system kernel (based on the Mach kernel developed by Carnegie Mellon University) and device drivers. The kernel is the foundation on which the entire iOS platform is built and provides the low level interface to the underlying hardware. Amongst other things, the kernel is responsible for memory allocation, process lifecycle management, input/output, inter-process communication, thread management, low level networking, file system access and thread management.

As an app developer your access to the System interfaces is restricted for security and stability reasons. Those interfaces that are available to you are contained in a C-based library called LibSystem. As with all other layers of the iOS stack, these interfaces should be used only when you are absolutely certain there is no way to achieve the same objective using a framework located in a higher iOS layer.