(O:S)

OPERATING SYSTEM

Contents

1. Basic Introduction :- Types, Process, Diagram, System Call, etc.

** 2. Process Sheduling :- FCFS, SJF, Pre-emptive, Round Robin, etc.

3. Process Synchronization :- Semaphore, Peterson sol., etc.

* 4. Deadlock & Threads :- Banker's Algo, etc.

* 5. Memory Management :- Paging, Segmentation, Virtual Memory, fragmentation, Page Replacement algo., etc.

* 6. Disc Sheduling :- SCAN, CSCAN, FCFS, etc.

7. UNIX Commands :- (ls, cd, chmod, etc), Open System call, etc.

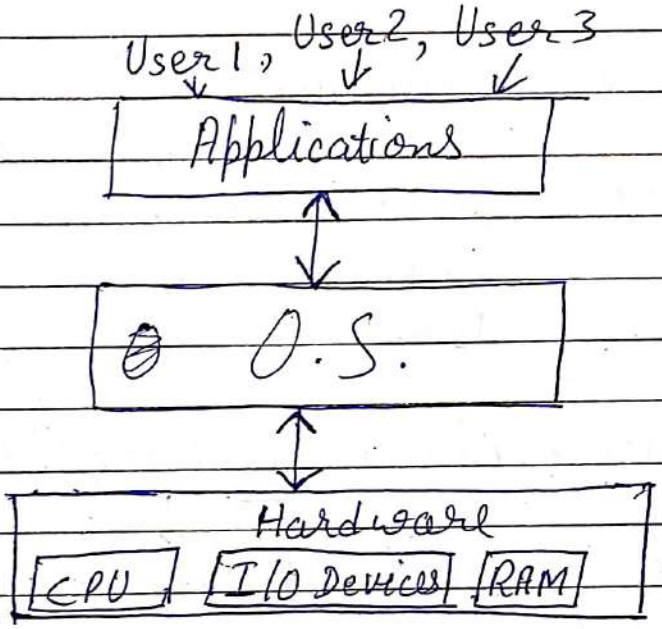
8. File Management & Security :- Sequential access, Random access, linked access, etc. (Defination in sequrity).

Gary Kildall → Father of O.S.

1. Introduction to O.S

Date
Page

- Operating System :- OS is a system software, It works as a Interface b/w User & the Hardware.



- If there is NO O.S., then we have to write the Program for every task we want to perform.
- Primary Goal of O.S. is to provide Convinience.
- Throughput :- No. of tasks executed per unit time.
- Functionality :-

1. Resource Management :- O.S. tries to divide the Resource to equally to all users, Mostly use in Parallel Processing where users are more than one.

2. Process Management :- How to execute Multiple Processes at a time.
* CPU Sheduling.

3. Storage Management :- Means the Data that we want to store in our system permanently, O.S. manages how we store that Data.
* File System.

4. Memory Management :- (RAM)
When we perform any task, first of all it will come in the RAM.
* (Multi Programming & Multitasking concept)

* Swapping :- When any task is done, the O.S. take that task out of RAM & Put New Task ~~in~~ ~~it~~ ~~to perform~~ process

5. Security :- (Windows uses KERBEROS security Protocol).
When we perform any task, we allocate some RAM to that task, when any Process ~~try~~ ~~To~~ calls or excecutes any Instruction outside the given segment, Then ~~to~~ that Process will be blocked Instantly.

Date
Page

* Sometimes we user directly use Hardware.

User → OS → Hardware
User → Hardware

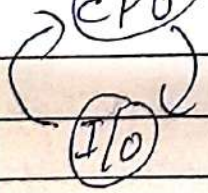
Example → When we Print any file using Printer (Hardware)

Types of O.S. :-

1. Batch
2. Multiprogrammed
3. Multitasking
4. Real time O.S.
5. Distributed
6. Clustered
7. Embedded.

1. Batch O.S. :-

Baches
B_1 B_2 B_3
2 3 4

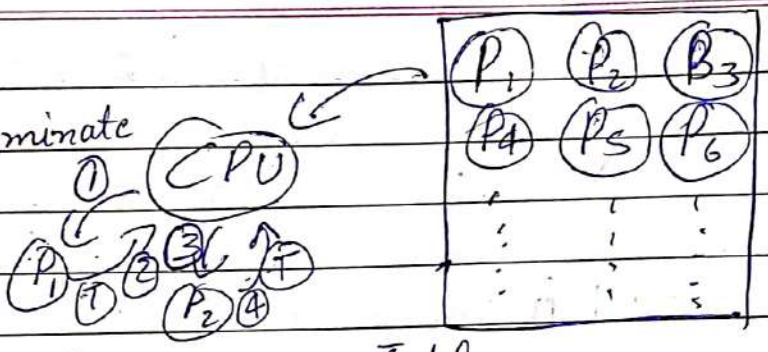


- CPU will stay Idle until B_1 completes it I/O (Input Output) Processes (This is an Disadvantage)

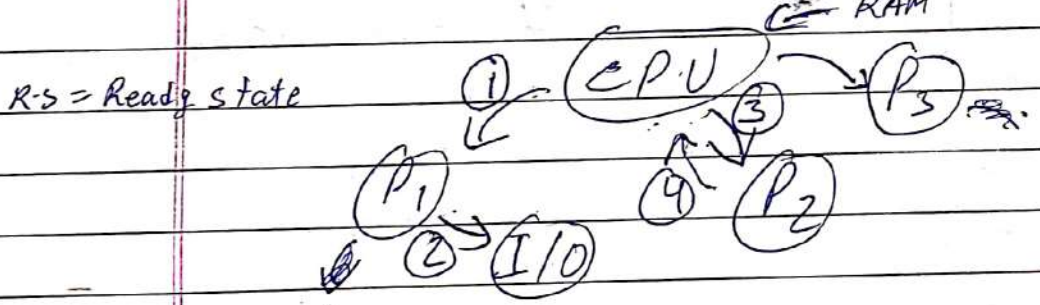
2. Multiprogrammed O.S. :- (Non-Preemptive).

- We bringing Multiple Process in the RAM. Multiprogramming OS allows more than one program to run simultaneously using a single CPU.

Non-Preemptive ⟷ Multiprogramming
Preemptive ⟷ Multitasking

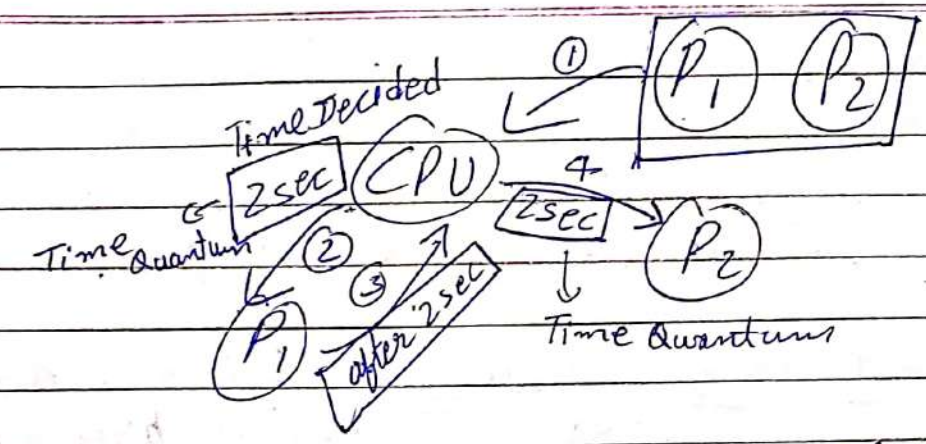


* - CPU Never go Idle.
* - CPU will ~~the given task~~ complete the given task/Process(P_1) then go to Next task unless & untill 'P_1' goes for some I/O Processes, In that case CPU skip(RST) P_1 & go to P_2.



* 3. Multitasking / Time Sharing O.S. :- Preemptive *

- A Multitasking O.S. allows Multiple processes/ tasks to be executed at the same time utilizing multiple CPUs. → (TQ Time Quant)
- We will decide the time in which Process to be done, If it done in that time, its good. Otherwise we will move on to the next task & shedule the previous task for future.
* - It is more responsive.



Not Imp

4. Real time O.S. :- ~~Hard~~ Time matters.
Hard, soft.

Not Imp

5. Distributed O.S :- When Machines are distributed geographically & connected with a network.

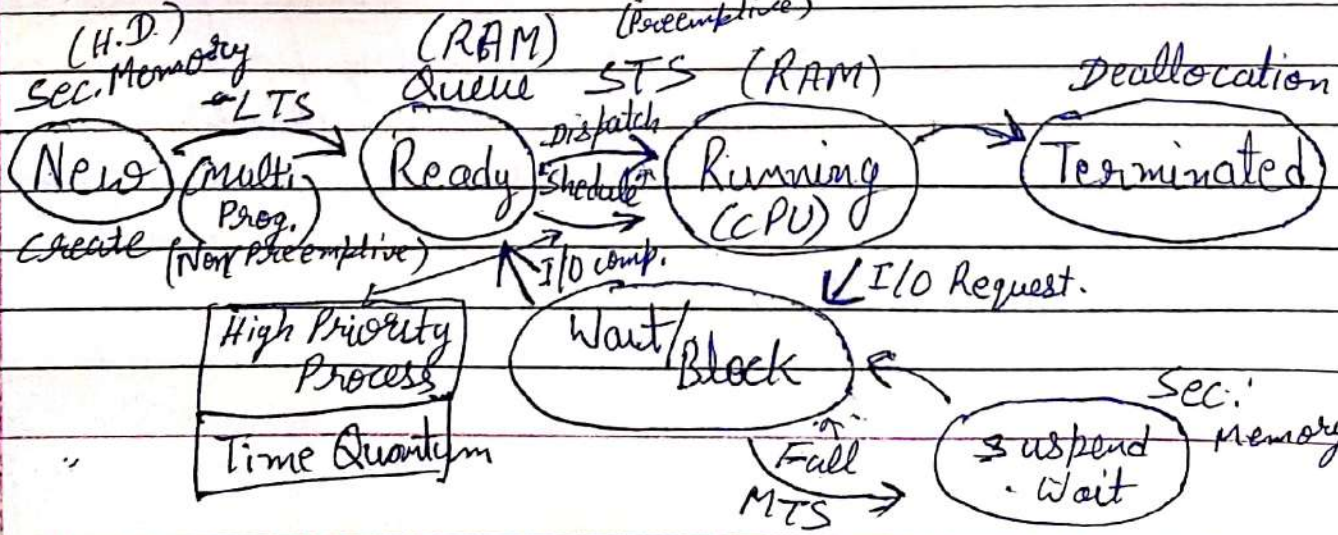
N. I.

6. Clustured O.S :- ~~Same as~~ When Machines are connected with a local network.

N. I.

7. Embedded O.S :- Work in single functionality Can't do changes in them.

— Process States :-



* - Long Term Sheduler (LTS) :- ~~Long~~ ~~te~~ Job Sheduler regulates the programs which are selected to system for processing.

* - Short Term Sheduler (STS) :- STS ensures which program is suitable or Important for processing It regulates the less DOM. (Degree of Multiprogramming

- D.O.M :- The maximum number of processes that a single processor system can accommodate efficiently.

* - Time Quantum :- The Period of time in which a process is allowed to run in a Preemptive Multitasking system.

- Mid Term Sheduler (MTS) :- MTS is Incharge of handling the swaped out processes. A running process may become suspended if it makes an I/O request. A ~~swapped~~ Suspended process can't make any progress towards completion.

— System Calls :- System call is a programmatic way through which user shifts from user mode to Kernal mode.

→ File Related :- Open(), Read(), write(), close(), create(), etc.

→ Device Related :- Read, write, Reposition, ioctl (I/O ctrl), fcntl.

→ Information :- get Pid, attributes, get system time & data.

→ Process control :- Load, execute, abort, fork, wait, allocate, etc

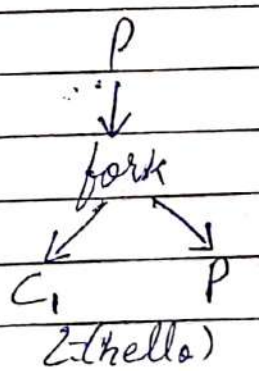
→ Communication :- Pipe(), create/delete conections, shmget()

* — Fork System call Fork() :- A fork is a process that generates a copy of itself.

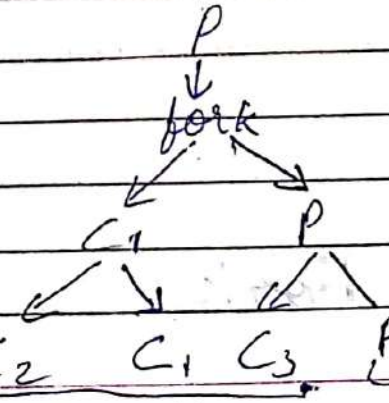
Parent Process (P)

```
main()
{
  fork();
  Printf("hello");
}
```

Child Process (C)



— If we use 2 fork();
fork();



4 = hello

* To check how many times output Print.

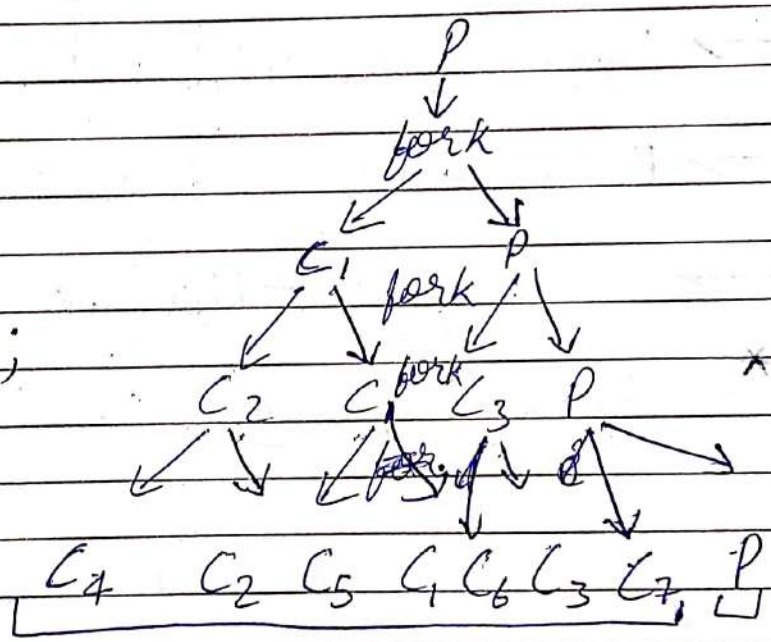
2^n | n = No. of fork

* To check No. of child Processes.

$2^n - 1$

- If we use 3 fors!

```
main()
{
  fork();
  fork();
  fork();
  printf("hello");
}
```



Child Processes

No. of 'hello' print = 2^n

$= 2^3 = 2 \times 2 \times 2$

$= 8$

No. of child processes = $2^n - 1$

$= 8 - 1$

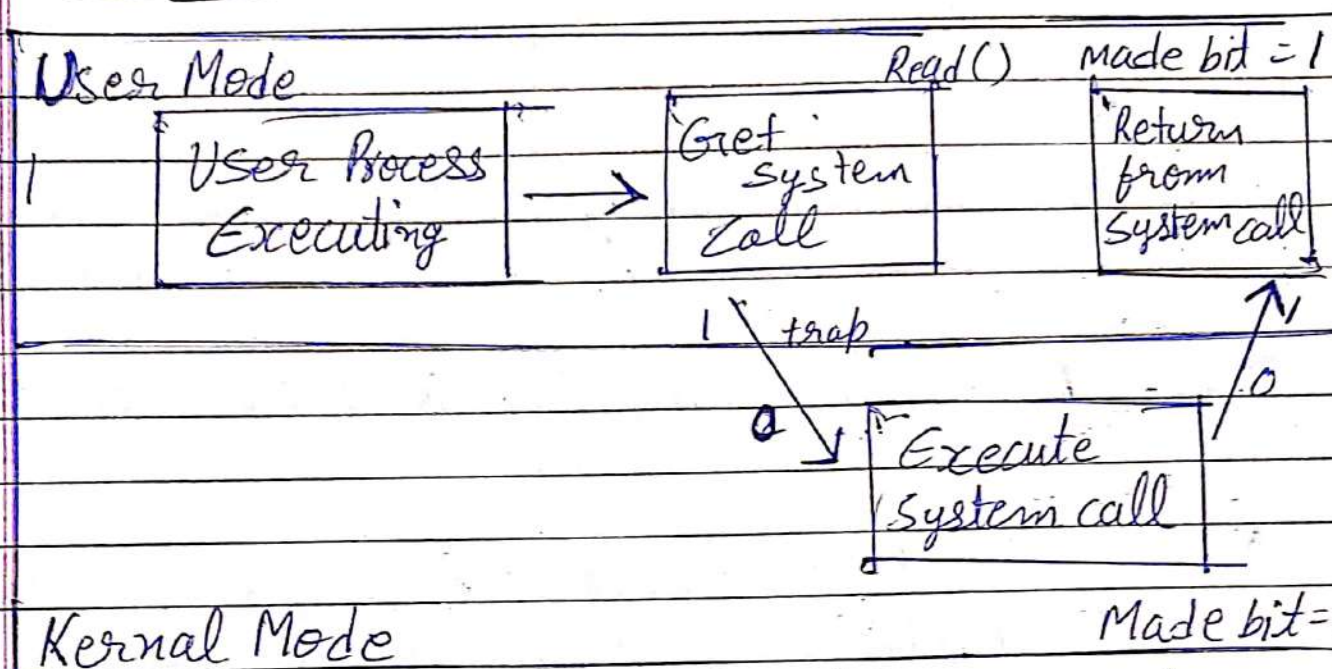
$= 7$

* Fork ⟶ 0 = Child

⟶ +1 (+ve) = Parent

rare case ⟶ -ve = No child created.

— User Mode vs. Kernal Mode :-



Process	Threads
1. System calls Involved in Process	There is NO system call Involved.
2. OS treats different processes differently	All user level threads treated as single task for OS.
3. Different processes have different copies of data, files etc.	Threads share same copy of code & data.
* 4. Content switching is slower.	Content switching is faster.
* 5. Blocking a process will not block another.	Blocking a thread will block entire process.
6. Independent	Interdependent.

— User level Threads vs. Kernal level threads

ULT	KLT
1. ULT are managed by User level library.	1. KLT are managed by OS system calls
2. ULT are typically faster.	2. KLT are slower.
* 3. Context switching is faster.	3. Context switching is slower.
* 4. If one ULT perform blocking operation then the entire process get blocked.	4. If one KLT blocked, no affect on others.

* Process > KLT > ULT

CT

CT = context switching Time

2. Process Sheduling

- Sheduling algorithm :- Sheduling algo. is a way or sheduling selecting a process from ready queue & put it on the CPU.

- Preemptive Sheduling :- Preemptive sheduling is a CPU sheduling technique that works by deviding Time slots (Time Quantum) of CPU to given process.
  * These Time slots also known as Time Quantum.
  * Works on concept of Multitasking O.S.
- Non-Preemptive Sheduling :- It is a CPU sheduling technique, the process takes the resource (CPU time) & holds it till the process gets terminated or pushed to the waiting state.
  * Works on concept of ~~Multitasking O.S.~~ Multiprogramming O.S.

- Different Times in CPU Sheduling :-

1. Arrival (AT) :- The time at which process enter the Ready Queue or State.
2. Burst time (BT) :- The Time required by a process to get execute on CPU. (Duration eg. 1s-3sec)

POT = Point of time

3. Completion Time :- The Time at which Process complete its execution (POT)

4. Turn around Time :- {Completion T. - Arrival T.}

5. Waiting Time :- {Turnaround T - Burst T.}

6. Response Time :- (POT)

* [The time at which a process gets CPU first time (POT) - (Arrival time)]

— FCFS : First Come First Serve.

- The process which come first in the queue gets the CPU first.
- The Process which request the CPU first, get the CPU allocation first.

* - Criteria : "Arrival Time"
"Mode : "Non Preemptive"

* In Non Preemptive Case :

$$WT = RT$$

Waiting Time = Response Time

Q-

Process No.	Arrival Time (Given)	Burst Time (Given)	Completion Time	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

Gantt Chart: | P_1 | P_2 | IDLE CPU | P_3 | P_4 |

0 2 4 5 8 12

Time ⟶

- Time taken to comlete all Processes = 12

- **SJF : Shortest Job First**

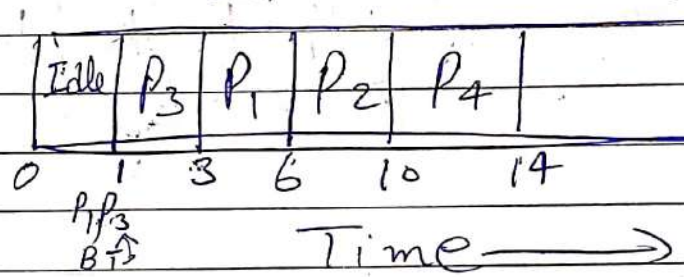
- The Process with the least/shortest Burst Time, gets the CPU allocation first.

*- Criteria : "Burst Time"
Mode : "Non-Preemptive"

Q—

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4	4	4	14	10	6	6

Grantt Chart



→ Total time taken = 14

— SRTF :- Shortest Remaining Time First

- In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, The short Term scheduler schedules the process with the least remaining burst time among the list of available processes & the running process.

*— Criteria :- "Burst time"
Mode : "Preemptive"

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	5	9	9	4	0
P_2	1	3	4	3	0	0
P_3	2	4	13	11	7	7
P_4	4	1	5	1	0	0

Gantt chart

P_1	P_2	P_2	P_2	P_4	P_1	P_3	P_1	P_1	P_3
0	1	2	3	4	5	6	7	8	9	13

Total time taken = 13

* **Round Robin** :- Round Robin is a CPU Scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to Implement, and starvation free as all processes get fair share of CPU.

* Criteria : "Time Quantum"

Mode : "Preemptive"

Q-

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	5	12	12	7	0
P_2	1	4	11	10	6	1
P_3	2	2	6	4	2	2
P_4	4	1	9	5	4	4

Given TQ = 2

Ready Queue | P_1 | P_2 | P_3 | P_1 | P_4 | P_2 | P_1 |

Running Queue | P_1 | P_2 | P_3 | P_1 | P_4 | P_2 | P_1 |

(Gantt Chart) 0 2 4 6 8 9 11 12

1 2 3 4 5 6

Time →

C.S = 6

* Context Switches :- How many times Previous Process saved & New Process (Next) loaded.

* Priority Sheduling :- It is Non-Preemptive

~~Criter~~ But we do question on Preemptive

* Criteria : "Priority"
Mode : "Preemptive"

Priority	Arrived Time	Burst Time	Completion Time	TAT	WT	Process No.
10	0	5	12	12	7	P_1
20	1	4	8	7	3	P_2
30	2	2	4	2	0	P_3
40	4	1	5	1	0	P_4

Gantt Chart :

P_1	P_2	P_3	P_3	P_4	P_2	P_1

0 1 2 3 4 5 8 12

* Given
Higher the Priority
. Higher the No

— Mix Burst Time :

(CPU & I/O both)
Mode : "Preemptive"
Ceiteria : "Priority based
Find CT of P_1, P_2, P_3, P_4
Given : Lower the No.,
Higher the Priority.

Process	AT	Priority	CPU	I/O	CPU
P_1	0	2	1	5	3
P_2	2	3	3	3	1
P_3	3	1	2	3	1
P_4	3	4	2	4	1

$P_4 \rightarrow$

P_1	CPU Idle	P_2	P_3	P_3	P_2	P_1	P_1	P_3	P_1	P_2	P_4	////	P_2	////	P_4

0 1 2 3 4 5 6 7 8 9 10 11 13 15 16 17 (18)

CT

$P_1 \rightarrow$ I/O

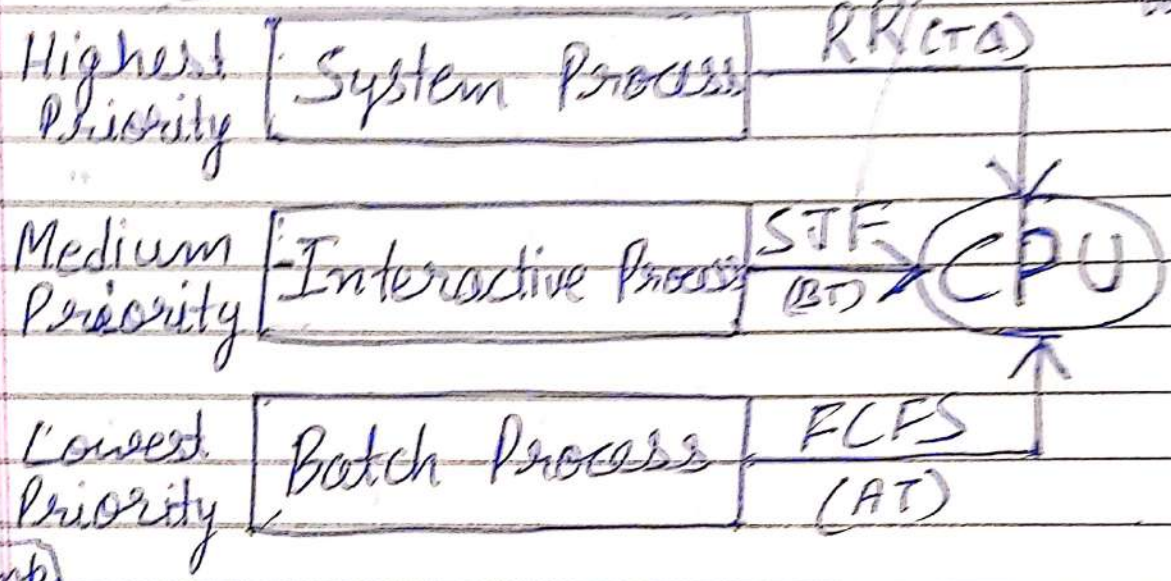
$P_3 \rightarrow$ I/O

$P_2 \rightarrow$ I/O

PCB : Process control Board

(Not Much Important)

— Multilevel Queue Scheduling :-



(Not Imp)

— Multilevel Feedback Queue :-

3. Process Synchronization

- Process Synchronization :- It is a way to coordinate processes that use shared data. It occurs in an operating system among cooperating processes.

1.- Cooperating Processes :- These are the processes that share Resources.

* Share :- Variable, Memory, code, Resources (CPU, Printer, Scanner)

2.- (NI) Independent Processes :- Independent Processes means, execution of one is not putting any effect on other.

* sleep(1); => Pause the Process for 1 sec.

Unsynchronized Parallel Processes

	P_1 Start with	P_2 Start
1	int x = shared; (x=5)	int y = shared; (y=5)
2	x++; → (x=6)	y--; (y=4)
3	sleep(1);	sleep(1);
4.	Shared = x; [x = 6] executed	shared = y; [y = 4] executed

* Given

int Shared = 5;

Uniprocessor System.

shared = 5 / 6 / 4

Final value = 4

- Race Condition :- It is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time.

- Producer Consumer Problem :- In the PCP, there is one Producer & there is one consumer that is consuming the products produced by the producer. The producers & consumers share the same memory buffer that is of fixed size.
The job of the producer is to generate data, put it into the buffer, & again start generating data. While the job of the consumer is to consume the data from the buffer.

- Printer Spooler Problem :- (loss of data)

- Critical Section Problem :-

- Critical section :- It is a place where shared variables, Resources are placed.

- Race condition occurs if two Parallel processes executes critical section at the same time.

* Synchronization Mechanism: Whichever method you take to achieve synchronization should fullfil 'four' (4) conditions.

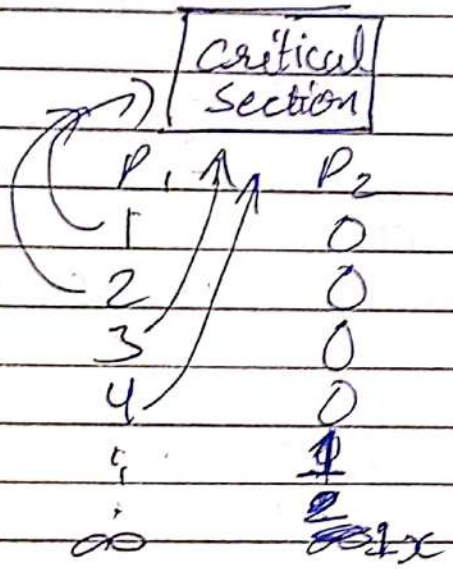
*
1. Mutual Exclusion. } Primary Rules
2. Progress
3. Bounded wait. } Secondary Rules
4. No assumption related to Hardware or Speed.

*
1. Mutual Exclusion: Mutex is a program object that prevents simultaneous access to a shared resource. (Critical section)
- ~~This concept~~ This concept is used in concurrent programming with a critical section, a piece of code in which processes or threads access a shared resources.

*
2. Progress: If NO process is executing in its critical section and some processes wish to enter their critical sections. Then only those processes that are not executing in their remainder section can participate in deciding which will enter its critical section next, and this selection can't be postponed.

While = True → Loop me Fasega
While = False → can go to 'Critical section'

* 3. Bound Wait :- The number of time a process is bypassed by another process.



4. No Assumption related to H/W & speed.

— LOCK Variable :- Lock variable is a solution for busy waiting that can be easily applied by more than two processes.
— In the lock variable mechanism, we use Lock variable.

* — Critical section solution using "Lock"

```
do {
    aquire Lock
    CS (critical sec)
    Release Lock
}
```

* Execute in user Mode
* Multiprocess solution
* No Mutual Exclusion Guarantee.

```
1. While (Lock == 1);   } Entry
2. Lock = 1             } Code
3. Critical section

4. Lock = 0.
```

Lock = 0 → CS vacant
Lock = 1 → CS full

* - Critical section sol. using "Test_and_set" Instruction

First it will test the value of Lock with 'while loop' & then set it in a single line.

```
While (test_and_set(&lock));
[CS]
Lock = false;

Boolean test_and_set (boolean *target)
    {
      boolean r = *target
      *target = TRUE;
       return r;
    }
```

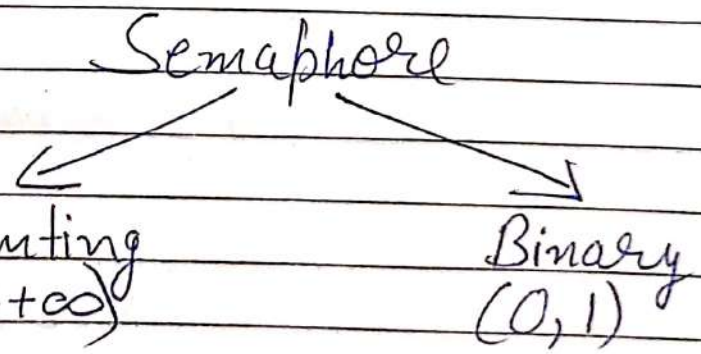
* — Turn Variable :- (Strict Alternation Approach)

It is the software mechanism Implemented at user mode. It is a busy waiting solution which can be Implemented only for two ~~purposes~~ processes.

— The turn variable is actually a Lock.

* 2 Process solution
* Run in User Mode
* Mutual Exclusion Verified
* Progress Not verified.
* Bounded wait satisfied.

** — Semaphore :- Semaphore is a Integer (int) variable, which is used in Mutual exclusion manner by various, concurrent cooperative Processes in order to achieve synchronization.



Entry code
CS
Exit code

- Operations in Semaphore:-

P(), Down, wait → Entry code
V(), Up, signal, Post, Release → Exit code

(P, wait) → Entry section | Exit Section

Code:

```
Down (Semaphore S)
{
    S.value = S.value - 1;
    if (S.value < 0)
    {
        Put Process(PCB) in
        suspended list
        sleep();
    }
    else
    {
        return;
    }
}
```

```
Up (Semaphore S)
{
    S.value = S.value + 1
    if (S.value <= 0)
    {
        Select a process
        from suspended list
        wake up();
    }
}
```

Q- S = 10, operations = 6 P() & 4 V()

10 - 6 = 4,
4 + 4 = 8

Final answer = 8

S = 0 — block

S = 1 — CS

- **Binary Semaphore** :- (0, 1)

```
Down(semaphore S)
{
  if (S value == 1)
  {
    S value = 0;
  }
  else
  {
    Block this process
    and place in suspend
    list, sleep();
  }
}
```

```
up(semaphore S)
{
  if (suspend list empty)
  {
    S value = 1;
  }
  else
  {
    Select a process
    from suspend list
    & , wake up();
  }
}
```

- V(); Never stops any process from entering Critical section (CS)

Q- Each process P_i {i = 1 to 9} Execute the following code.

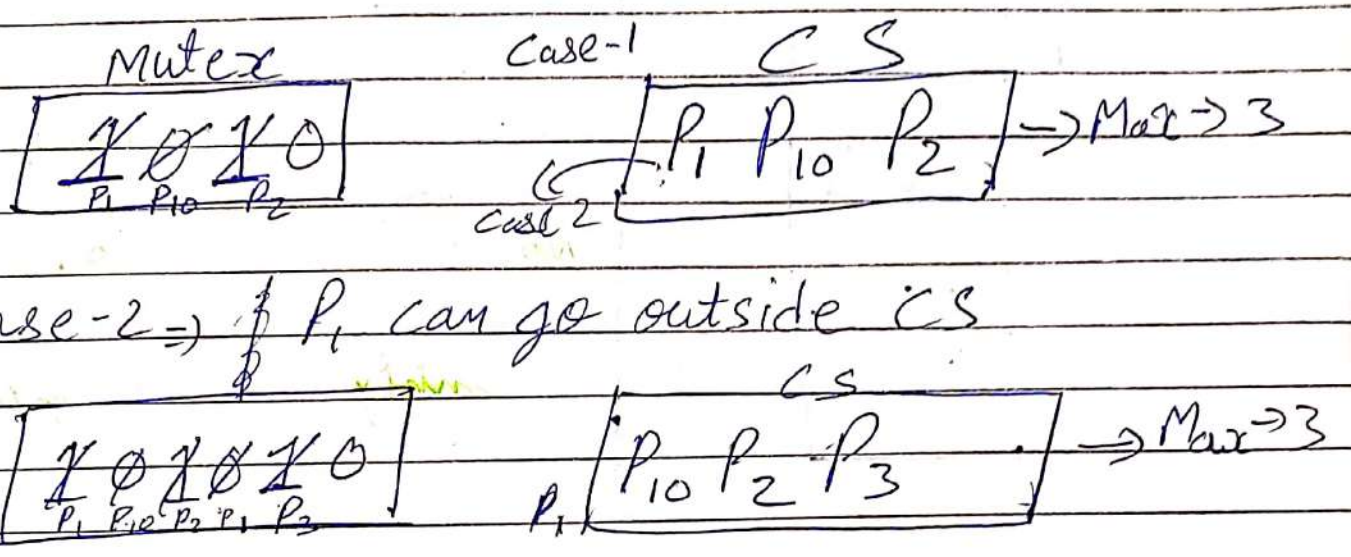
```
P(mutex) ← entry
[CS]
V(mutex) ← exit
```

Process P_{10} execute the following code

```
V(mutex) ← entry
[CS]
P(mutex) ← exit
```

- What is the maximum no. of Processes present in CS at any part of time?

sol —



→ Max No. of Processes at any time is 3

— Sol. of Producer consumer Problem (semaphore)

Produce_Item (item P);	Consumer
entry 1. down (empty);	1. down (full);
2. down (s);	2. down (s);
CS [Buffer[In] = itemp; In = (In+1) mod x;]	[item c = buffer[out]; out = (out+1) mod x;]
exit 3. up (s);	4. up (s);
4. up (full);	5. up (empty);

default semaphore = 1 *

N = 8

0	a
1	b
2	c
3	d
4	
5	
6	
7	

a, b, c Items already there

In 3 : Out 0

Producer Put his new Item on No. 3 because it is empty.

consumer Take the Item from No. 0 because it is full.

- Reader-Writer Problem :- This problem occurs when a Reader & a Writer use same database at the same time.

R-W → Problem
W-R → Problem
W-W → "
R-R → No Problem

```
int rc = 0;
Semaphore Mutex = 1;
Semaphore db = 1;
for Reader →
Void Reader (void)
{
  While (true)
}
```

rc = read count *

Reader exit code

```
down (mutex);
rc = rc - 1;
if (rc == 0) then
up (db);
```

down (mutex);

rc = rc+1

if (rc == 1) then
down db;

up(mutex)

DB

up(mutex)

Process data { }

for writer ↴

void writer (void)

{

while (true)

}

down (db)

DB

up(db); {

}

* db, mutex, rc → Variables
local

* will learn later

→ Dining Philosophers Problem :-

4. DEADLOCK & Threads

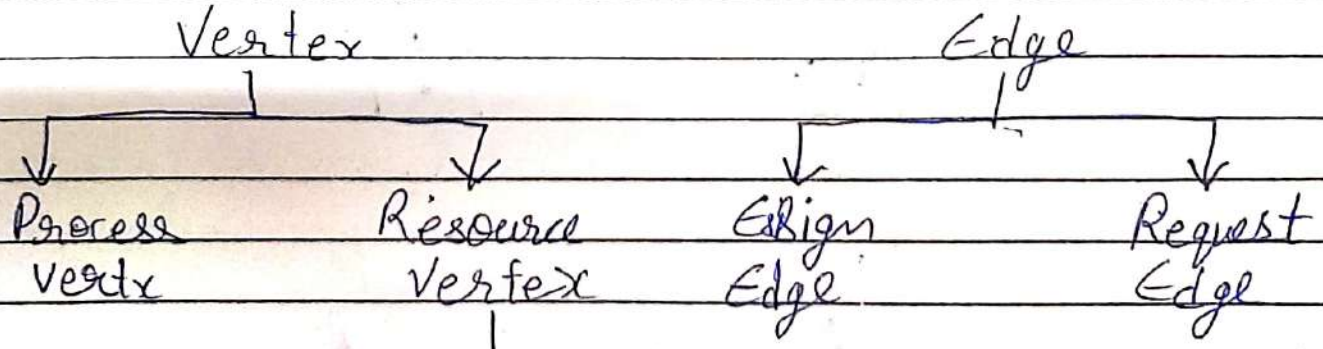
**
- **Deadlock** :- If two or more than two Processes are waiting on happening of some event, which never happens, then we say these Processes are in Deadlock & that state is called deadlock.

- Conditions for Deadlock :- (Necessary)

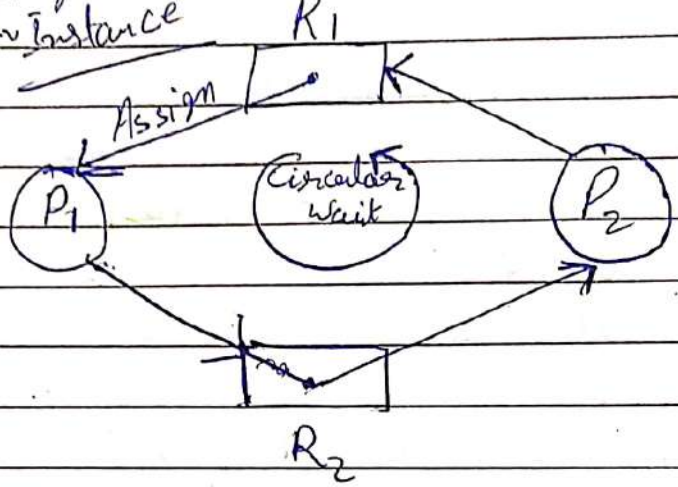
1. Mutual Exclusion. 2. No Preemption
3. Hold & Wait 4. Circular wait.

*
- Resource Allocation Graph :- RAG is the most efficient & convenient way to represent the state of the system.

Means, In our system, how the resources are allocated to the processes & how the processes have been assigned to multiple processes



-) Single Instance

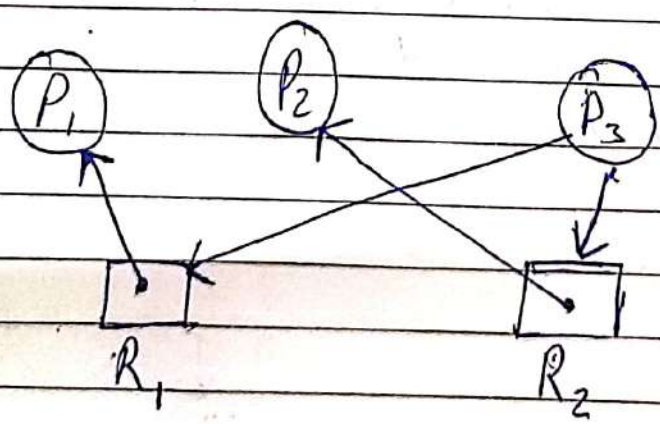


sol

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

→ Availability $(R_1, R_2) = (0, 0)$ → Deadlock ✓

=>



No circular wait

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	1	1

→ Availability (0,0) → Not dead
→ P_1 is not Requesting anything so, it will get executed.
→ New Availability (1,0)
→ P_2 will get executed, same as P_1

→ New Availability (1,1)
→ Now P_3 Requests (1,1) are same as availability, so its gets fulfilled/executed.

* → All Processes are executed. So, there is NO deadlock. (starvation)

*
→ Waiting

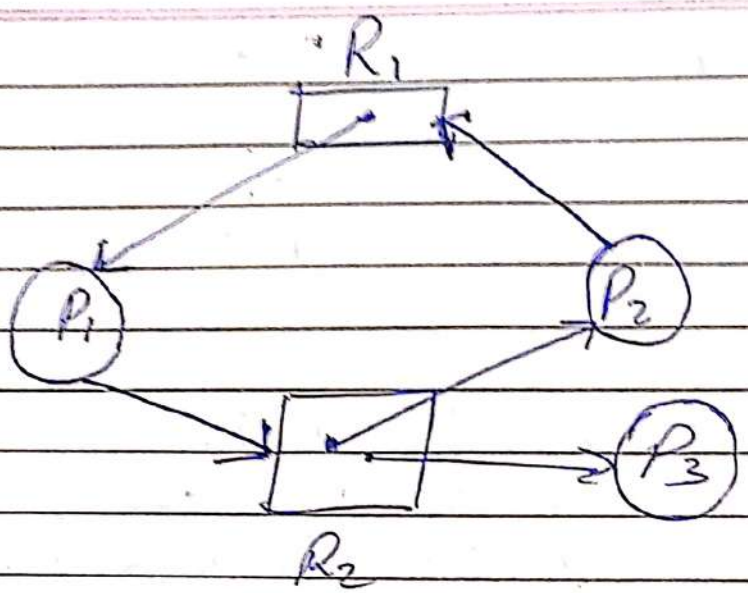
Finite time (starvation) | Infinite Time (Deadlock)

→ Starvation :- When we know, that Process will be executed after a finite Time.

* → In single instance case, If RAG has circular wait (cycle) then there will be always a Deadlock.

→ Multi-Instance RAG :-

1.→



	Allocation		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Availability (0, 0)

P_3 executed,

Availability (0, 1)

$R_2 \rightarrow P_1$

P_1 executed,

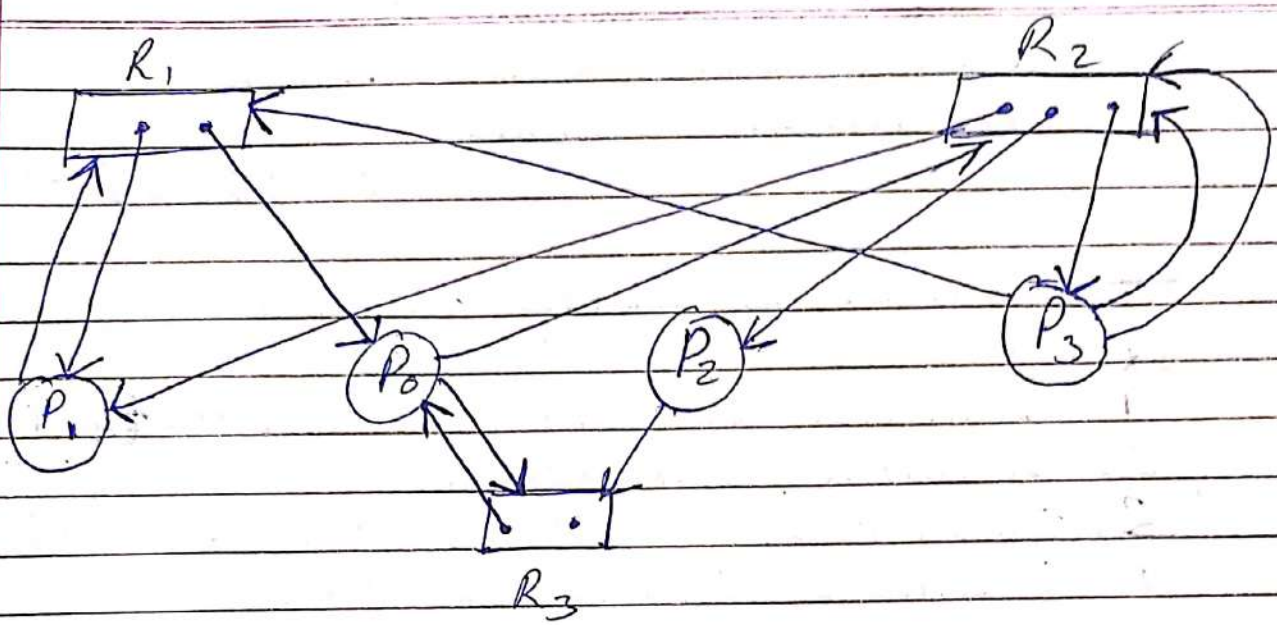
Avail. (1, 1)

P_2 — executed

Not a deadlock.

*→ Whether there is a circular wait in Multi Instance RAG, But Deadlock is not compulsory.

2. →



→

	Allocate			Request		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	1	2	0

$R_1\ R_2\ R_3$

Avail. = (0, 0, 1)

P_2 request (0 0 1) so it will executed.

new avail. (0 1 0 1)

P_0 request are (0 1 1) so it will be executed.

new avail (1 1 2)

P_1 need (1 0 0) so it will be executed.

new avail (2 2 2)

P_3 need (1 2 0) so it will be exe.

→ Final availability (2 3 2) ✓

* It's NOT a Deadlock
Execution Sequence, $P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

Various to handle deadlock :-
(eg. windows OS)

** 1. Deadlock Ignorance (Ostrich Method)
2. Deadlok Prevention.
** 3. Deadlock avoidance (Banker's Algo)
4. Deadlock detection & recovery.

1. Deadlock Ignorance :- (Ostrich Method)

→ It is most widely used Mechanism.
→ In this method, Operating System simply assumes, Deadlock never occurs.
→ Most widely used Method (eg. Windows)

2. Deadlock Prevention :- DP algorithm used in concurrent Programming when multiple Processes must acquire more than one shared resource.
→ This algo organises resources usage by each process to ensure that atleast one process is always able to get all resources it made.

* → Deadlock Prevention says either remove all four necessary conditions for Deadlock or remove one from them.

3. Deadlock Detection & Recovery :- First detect if there is any Deadlock present the recover it

4. Deadlock Avoidance :- (Banker's Algo.)

Banker's algorithm is a resource allocation & deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities before deciding whether allocation should be allowed to continue.

→ It is a deadlock avoidance algorithm.
→ It is used for deadlock detection.

* Q-

Process	Allocation			Max Need			Available			Remaining		
Exe. seq.	A	B	C	A	B	C	A	B	C	A	B	C
4 ← P_1	0	1	0	7	5	3	3	3	2	7	4	3
6 ← P_2	2	0	0	3	2	2	5	3	2	1	2	2
5 ← P_3	3	0	2	9	0	2	7	4	3	6	0	0
2 ← P_4	2	1	1	4	2	2	7	4	5	2	1	1
3 ← P_5	0	0	2	5	3	3	7	5	5	5	3	1
							10					
Total	7	2	5				10	5	7			

Given:- Total · A = 10, B = 5, C = 7

∴ Formula :- Available = Total - Allocated

Remaining Need = Max Need - Allocated

Sol:- Available = 10 5 7
- 7 2 5
3 3 2

Process Execution Sequence ⇒ $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ (Safe sequence)

* — Safe State ⇒ s-state, At least one process should be able to acquire its maximum possible set of resources.
→ A system is in a safe state it is in a safe sequence.

— Safe sequence :- Execution sequence of processes is called safe sequence, if there is NO deadlock situation.

*

— Min. No. of resources to avoid Deadlock

$$= [\Sigma \text{Min. Req.} + 1]$$

Min. Req. ⇒ Min. req. of resources by sum one Process.

e.g. if 2 resources req. by each process & then, 3 processes

Min = 1

3 Processes ⇒ (1 + 1 + 1) = 3

⇒ 3 + 1 = 4.

**

— [Resources + Processes > Total Demand]

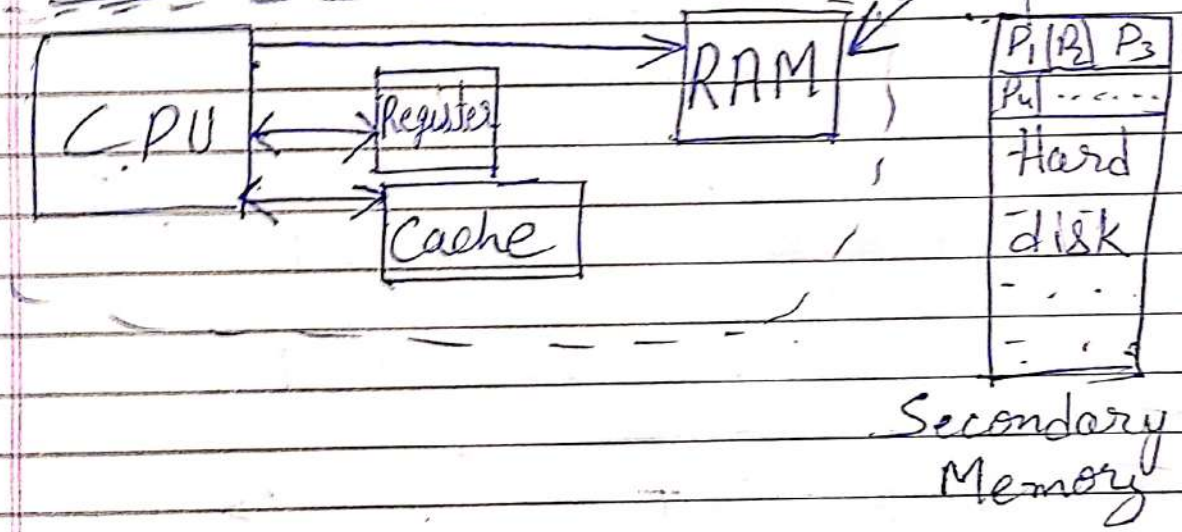
Deadlock Free.

[Total demand = Total Processes ⊕ X Min. need for execution.]

5. MEMORY MANAGEMENT

— Memory Management : Method of Managing primary Memory (RAM)

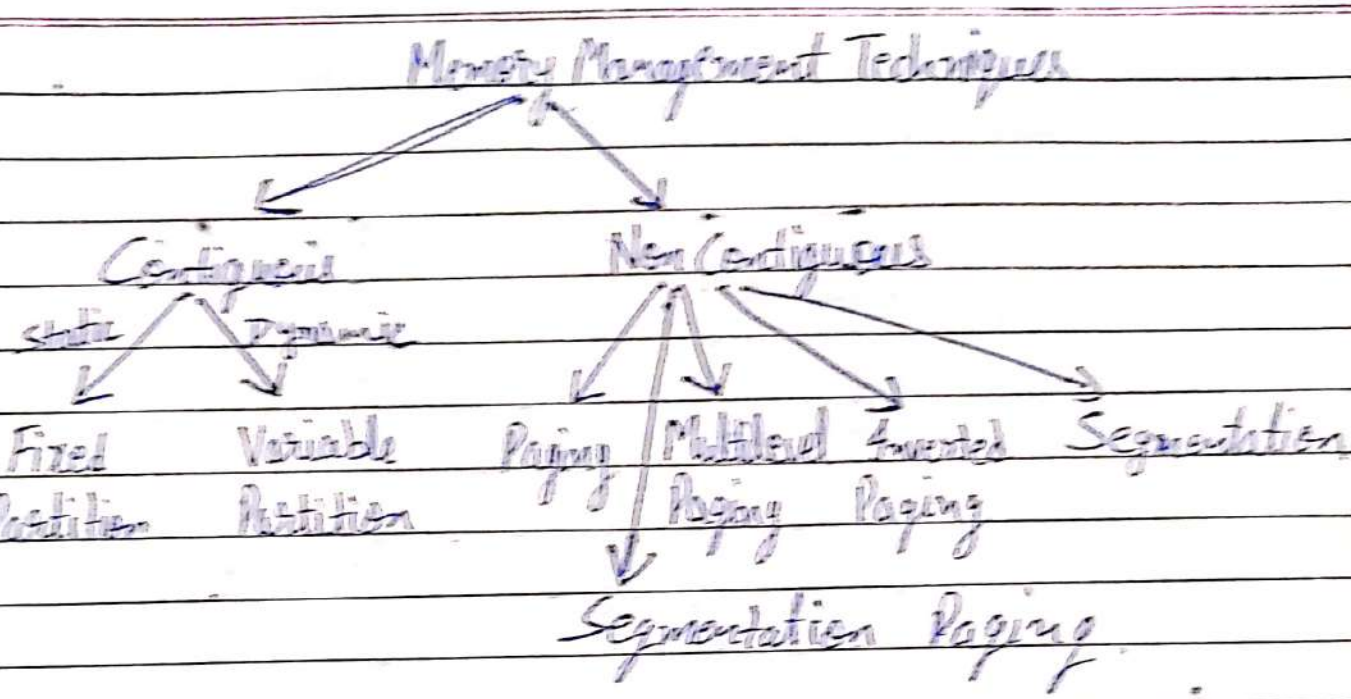
→ Goal : Efficient utilization of Memory



*. — Degree of Multiprogramming : → It keeps more & more number of processes in the RAM, means bring as many number of Processes as you can & try keep in the RAM so that efficiency of the system is very high.

→ When utilization of CPU is very high the performance of system will automatically Increase.

* — Memory Management Techniques :-



- Contiguous :- Dividing the memory into the fixed sized partition.

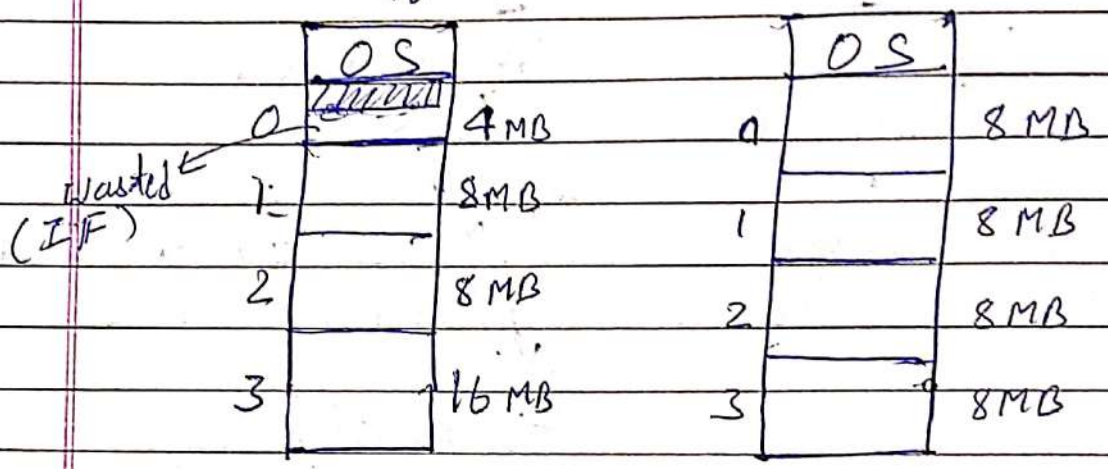
- Non-Contiguous :- The available free memory space is distributed here & there which means that all the free memory space is not in one place.

1. Fixed Partitioning :- (Static Partitioning)

In contiguous Memory allocation whenever the processes are coming into the RAM, how we are allocating them the space, that is one of the method is Fixed partitioning.

→ In the fixed partitioning, the concept is number partitions ~~of~~ are ~~fi~~ fixed.
→ The number of partitions are fixed from start but size of ~~eat~~ each partition can

be same, means can also be fixed or can be different also.



* — Internal Fragmentation :- When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block.
→ Due to this, the free space of memory block is unused which causes Internal fragmentation

— Limitations :- of (Fixed Partitioning)

1. Internal Fragmentation
2. Limit in Process size
3. Limitat on degree of Multiprogramming.
4. External Fragmentation.

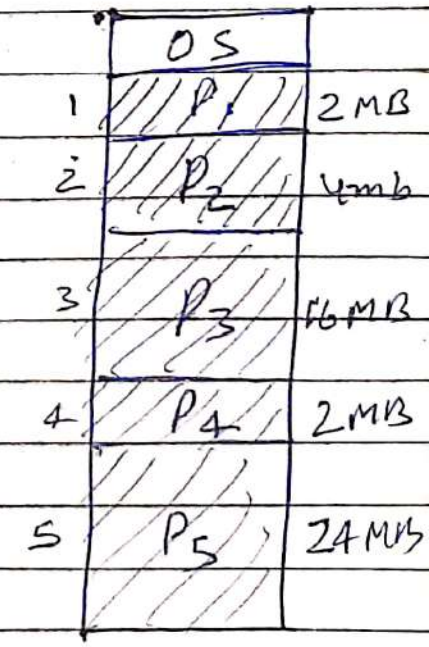
* — External Fragmentation :- EF happens when there is a sufficient quantity of area within the

memory to satisfy the memory request of a method.

* Whenever there is Internal fragmentation in the memory, there is always be External Fragmentation existing.

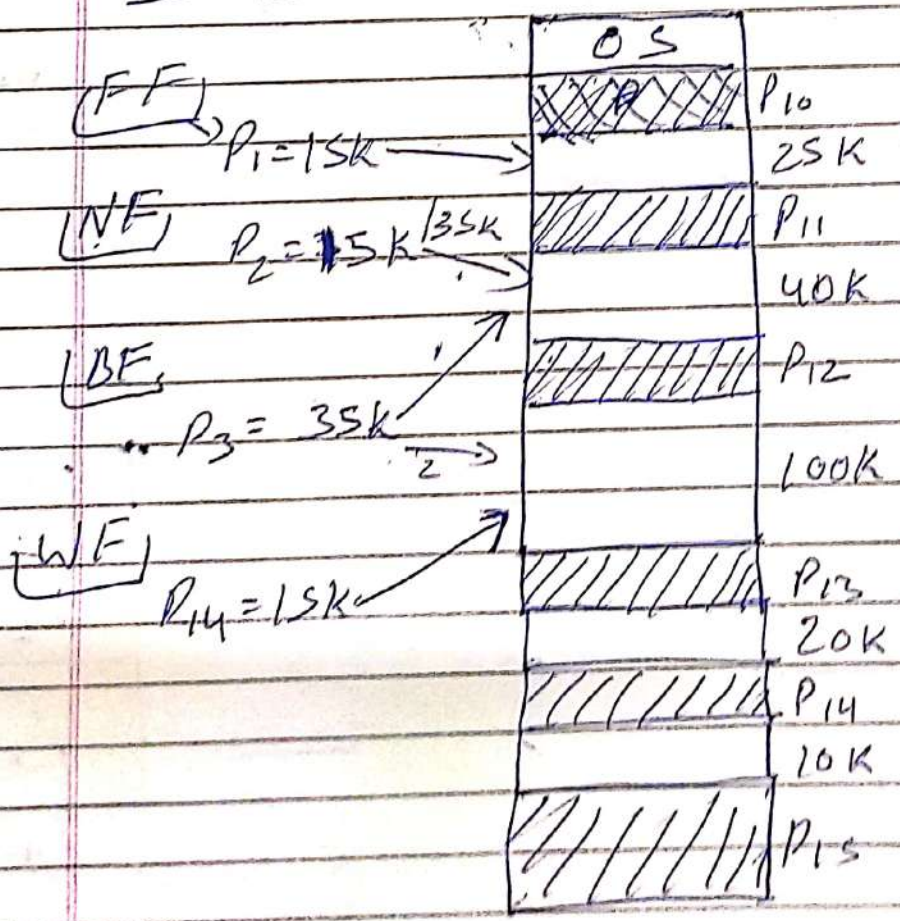
2. Variable Partitioning :- In variable partitioning whenever the processes are comming into RAM, only then we are allocating space to the processes.
- We keep RAM empty in starting, when ~~so~~ processes come in the RAM then at Run time the capacity they need, according to that space I will allocate them. But In fixed partitioning the capacity was fixed from the start.

*
1. No Internal Fragmentation
2. No Limit on no. of Processes.
3. No Limitations on the Process size.
*4. External Fragmentation.
5. Allocation/Deallocation Complex.

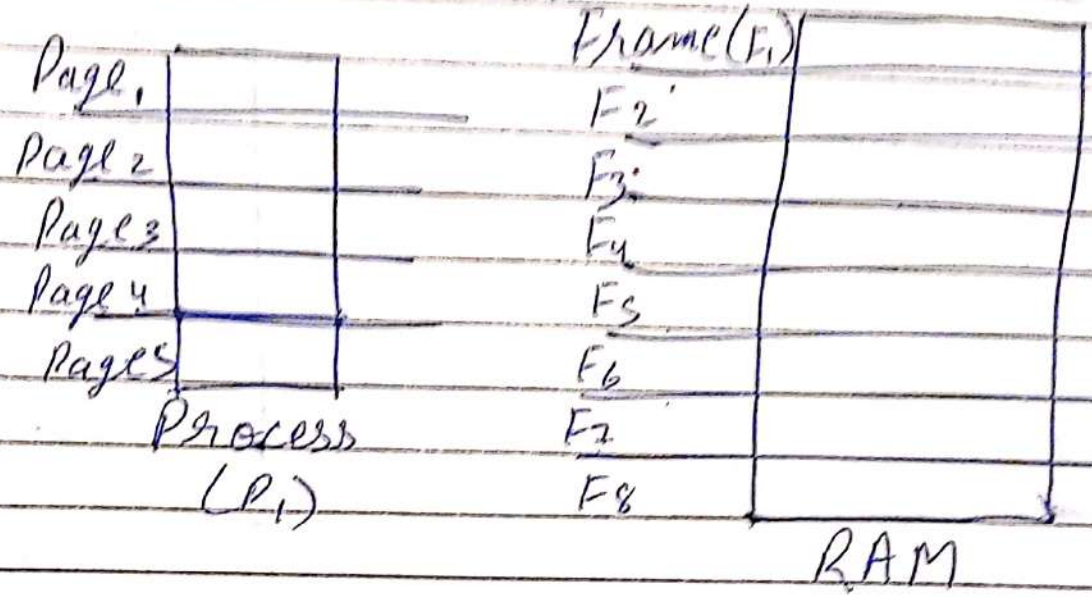


*

- First Fit :- Allocate the first hole that is big enough.
- Next Fit :- Same as First Fit, but always start search from last allocated hole.
- Best Fit :- Allocate the smallest hole that is Big enough.
- Worst Fit :- Allocate the largest hole.



Non Contiguous Memory Allocation:



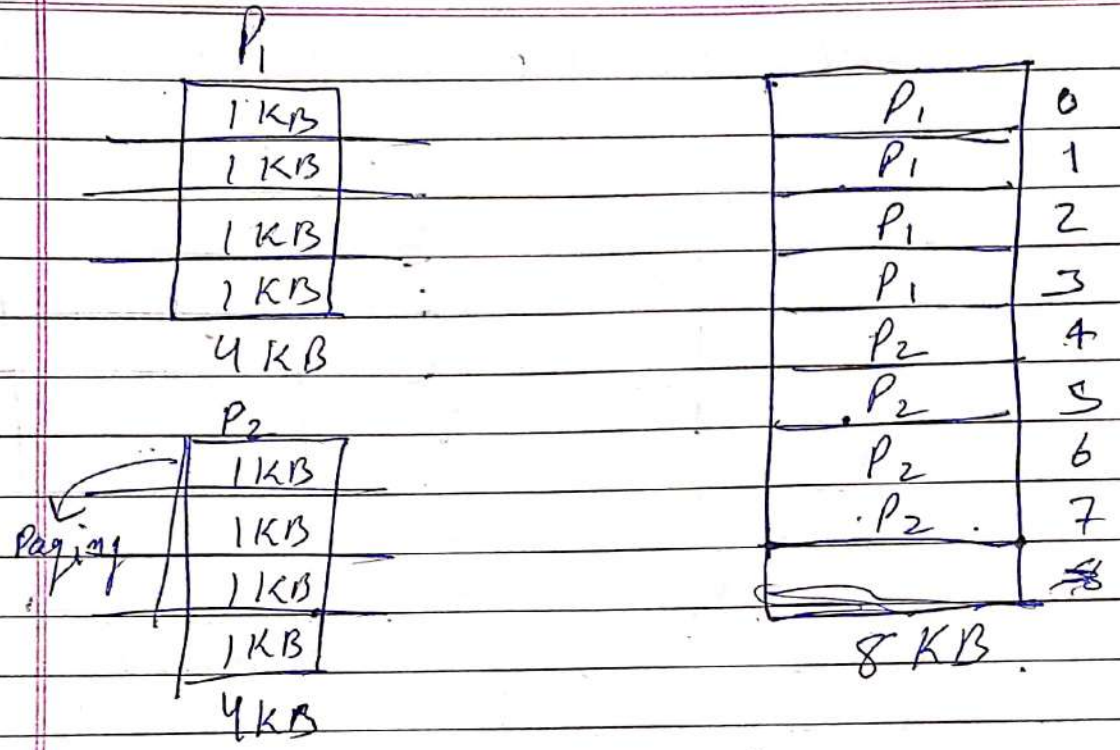
- **Page:-** Page is a fixed length contiguous block of virtual memory, described by a single entry in the Page table.

Frame → It is the smallest unit of data memory management in a virtual memory OS.

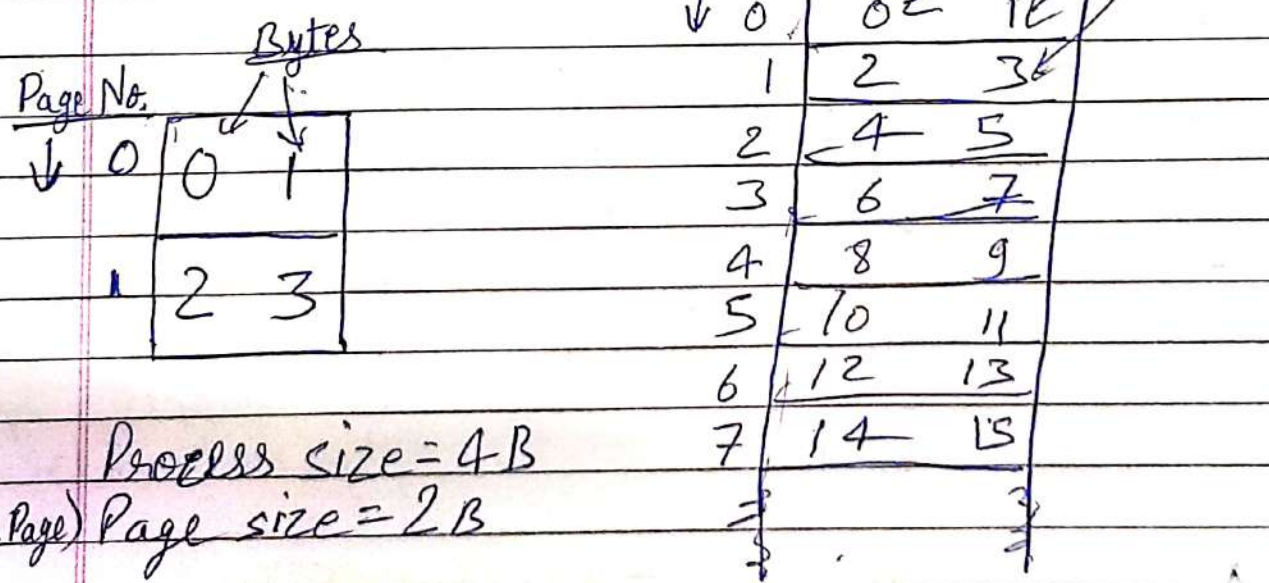
- **Frame:-** It is the smallest unit of data for memory management in a virtual memory OS.

- **Page Table:-** It is the data structure used by a virtual memory system in a computer OS to store the mapping b/w virtual addresses and physical addresses

* **Page Size = Frame Size**



* **Paging** :- In Paging, we split a process into equally sized pages and insert it into the frames of main memory.



Process size = 4B

(1 Page) Page size = 2B

No. of Pages/Process $= \frac{4B}{2B} = 2B$

RAM size = 16B

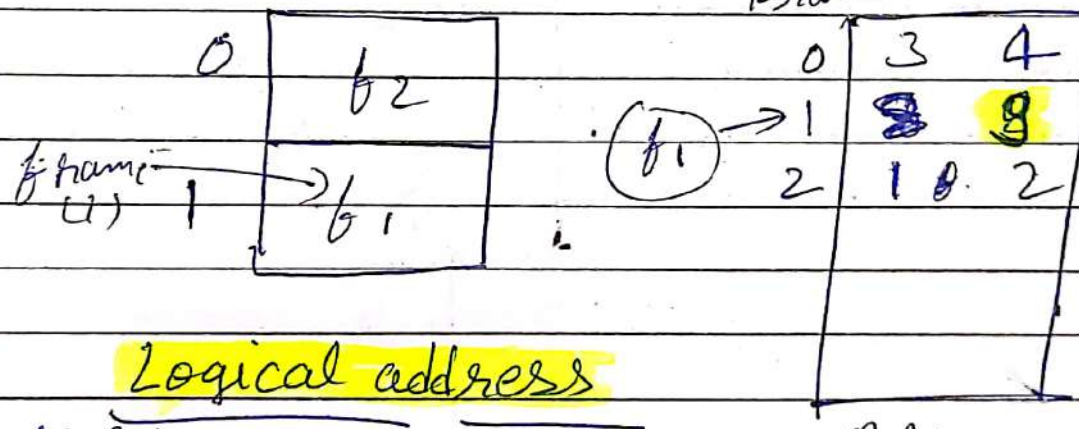
Frame size = 2B

(1 Frame)

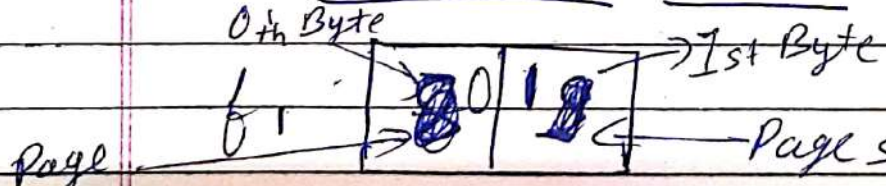
No. of B Frames :- $\frac{16\,B}{2\,B}$ = 8 B Frames

* Mapping :- The Translation between the logical address space and the physical memory.
→ It translates b/w logical & physical address.
→ It aids in memory protection. (q.v.)
→ It enables better management of memory resources.
→ Memory Management Unit (MMU) helps in Mapping

Page table of Process (P_1)

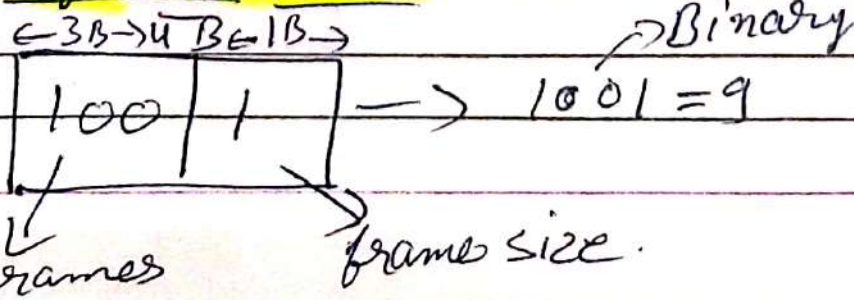


Logical address



Page No. We need 1st Byte in frame (f1) is 8

Physical address



* 1 TB = 2^{40} Bytes
* 1 GB = 2^{30} Bytes
* 1 MB = 2^{20} Bytes
* 1 KB = 2^{10} Bytes
* 1 word = 1 Byte

Q — Given :) Logical address space/size (LAS)
Physical " " " (PAS)

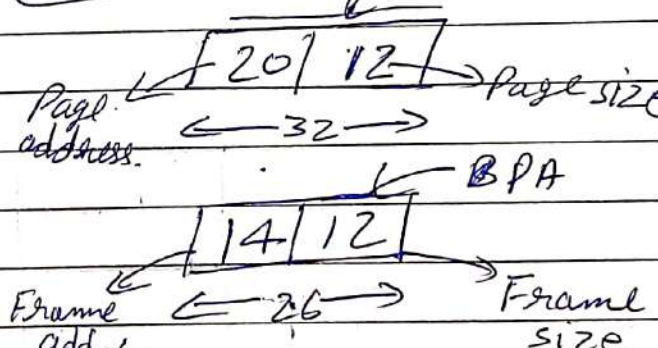
- LAS = 4 GB
- PAS = 64 MB
- Page Size = 4 KB

Find :-

- No. of Pages = 2^{20}
- No of Frames = 2^{14}
- No. of entries in Page table = 2^{20}
- Size of Page table = $2^{20} \times 14$

Sol :- Memory is byte addressable

LA = 4 GB = $2^2 \times 2^{30}$ Byte
= $2^2 \times 2^{30}$.
LA = 2^{32}



Sol :- Page Size = 4 KB = $2^2 \times 2^{10}$
= 2^{12}.

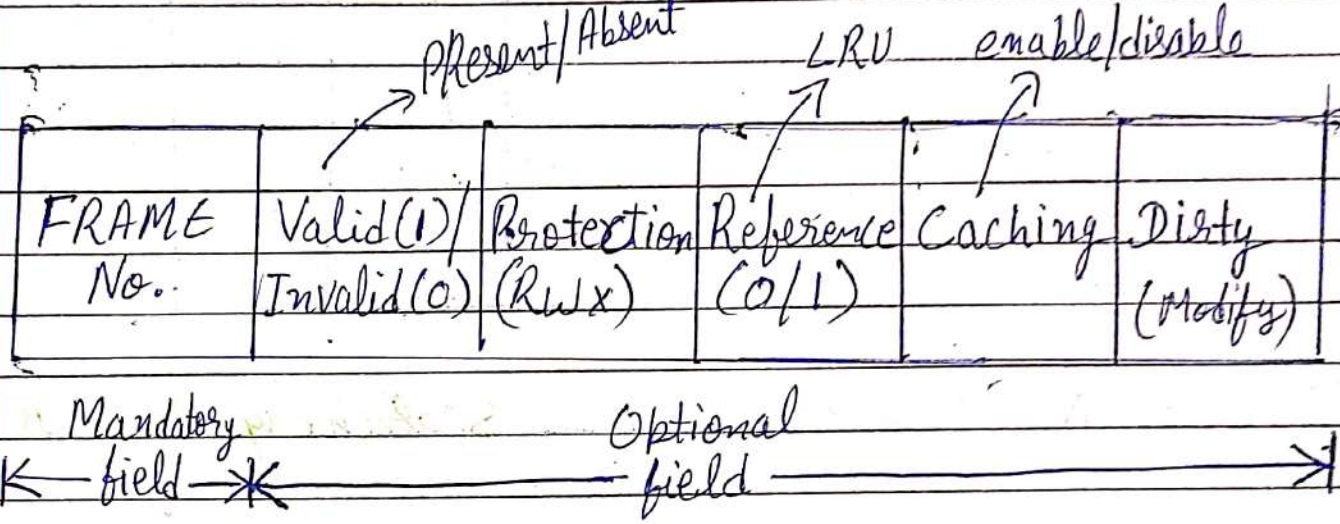
PA = 32 - 12 = 20

No. of
PA = $2^6 \times 2^{20}$
= 2^{26}

Frame address = 26 - 12
= 14

* No. of entries in Page Table = No. of Pages in Process

— Page Table Entry :-



* (RWX) → Read Write execute
* (LRU) → Page Replacement Algo.

— 2-Level Paging :- Multilevel Paging is a paging scheme which consist of two or more levels of Page tables in a hierarchical manner.

— Inverted Paging :- It is the global page table which is maintained by the Operating System for all the processes

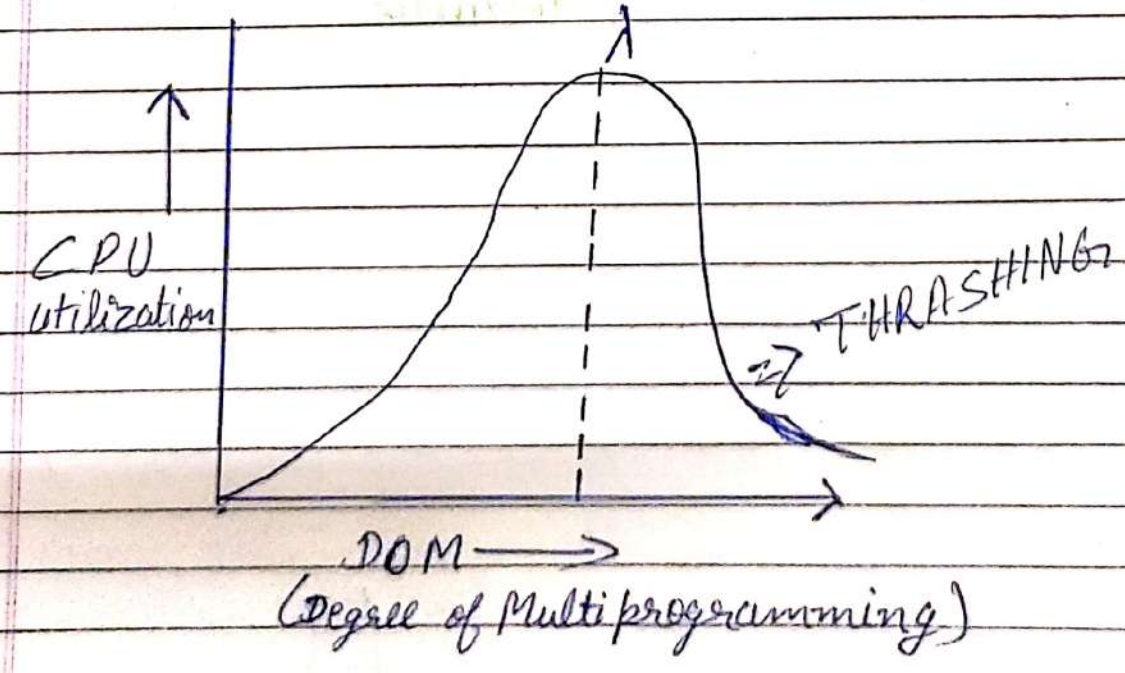
→ Each Process has its on Page Table.
→ Page table will be in RAM.
→ No. of Frames is equal to No. of entries.
→ Mandatory fields are: frame No, Process ID.

CPU → [P | d]

S.No.	Page No.	Process ID
0	p_0	P_1
1	p_1	P_2
2	p_2	P_1
3	p_1	P_3
4	p_3	P_2
5	p_2	P_3
6	p_3	P_1

→ Not famous becaus Seatching time is too High.

* — Thrashing :— Thrashing is a state in which the CPU performs "productive" work less, and "swapping" more.



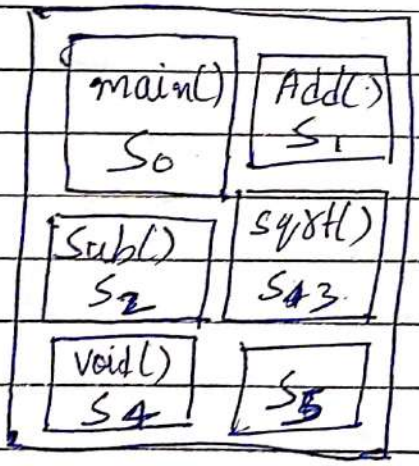
— Page Fault :- It is an Interruption that occurs when a software program attempts to access a memory block which is not currently stored in the system's RAM.

→ Points to Remove Thrashing :-

1. Increase the size of RAM.
2. Long Term Sheduler (lower)

*

— Segmentation :- It is a memory management technique in which the memory is divided into the variable size parts.



Base Address ←

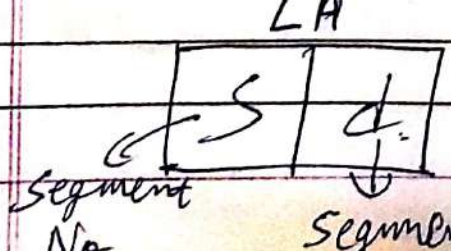
S.No.	BA	Size
0	3300	200
1	1800	400
2	2700	600
3	2300	400
4	2200	100
5	1500	300

→ Segment size

Segment Table

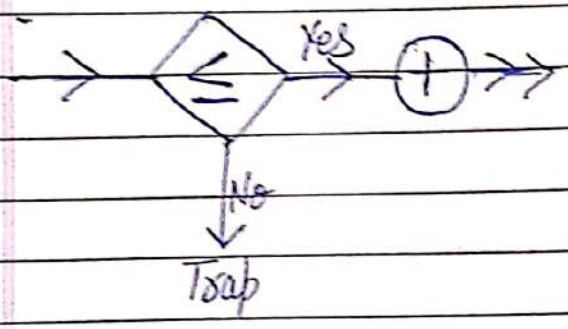
→ Segments (S0, S1, ...) can be of various size

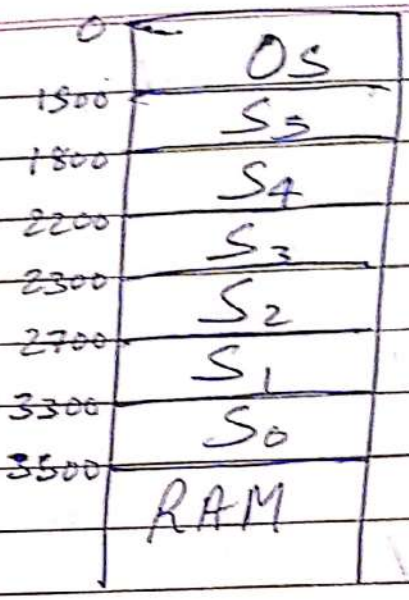
→ MMU (Memory Management Unit) will help to convert LA in to PA (Physical address)



d ≤ size

Size = Size in Table.





— Overlay :- The process of transferring a block of program or other data into internal memory, replacing what is already stored.

* — Virtual Memory :- A feature of an O.S. that enables a computer to be able to compensate shortages of physical memory by transferring pages of data from ~~xxx~~ RAM to disk storage.

→ We can also execute a process if its size is bigger than the RAM size.

* → Virtual Memory comes in use when RAM shortages of storage.

* → We only put those Pages of a Process in RAM which are used in execution of that Process.

* → No. and size of pages does not matters.

EMAT → Effective Memory Access Time

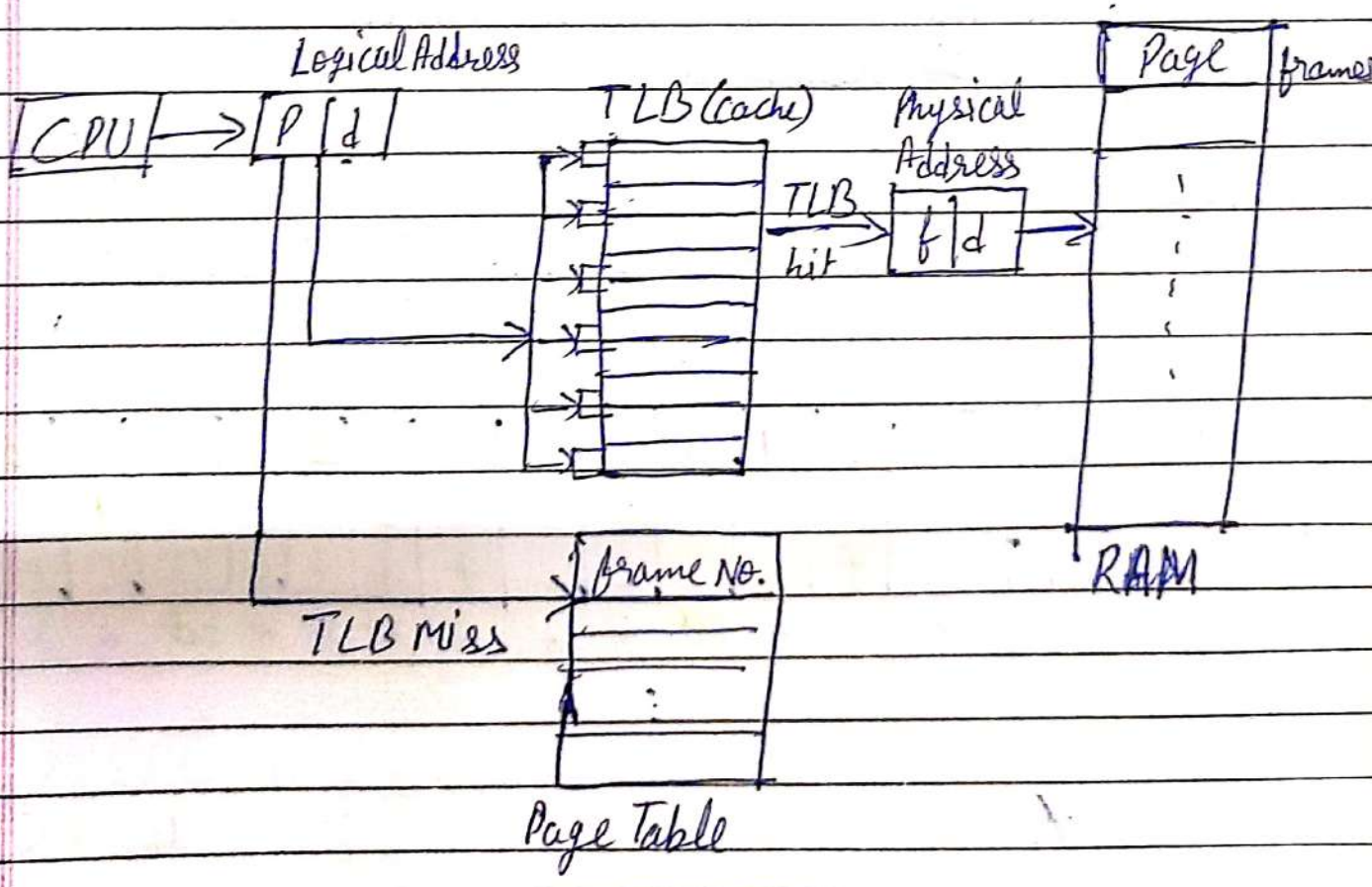
p → page fault.

$$EMAT = p(\text{page fault service time}) + 1-p(\text{RAM access Time})$$

(milisec) (nano sec)

— Translation Lookaside Buffer :- TLB is a Memory cache that is used to reduce the time taken to access a user memory location.

CPU → (PT) —frame→ (RAM)



EMAT = (TLB+x) + Miss (TLB+x) → No page fault

* — Page Replacement algo :- Bringing a page to the main memory if main memory is full, then Remove the old page from there by scrapping out & make space & place the new page in that place.

*
1. FIFO
2. Optimal Page Replacement.
3. Least Recently Used (LRU)

1. FIFO :- First In First Out :-

When a page needs to be Replaced, page in the front of the queue is selected for removal.

Reference String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

f3			1	1	1	1	0	0	0	3	3	3	3	2	2
f2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	*	*	*	*	hit	*	*	*	*	*	*	hit	*	*	hit

* = Page fault
hit = Page hit / Page found

hit Ratio = $\frac{3}{15} \times 100 = 20$

* **Hit Ratio = No. of hits / No. of References**

* — **Balady's anomaly** :- It is the phenomenon in which Increasing the number of page frames results in an Increase in the number of page faults for certain memory access patterns.

No. of page frames ∝ No. of page faults

2. **Optimal Page Replacement** :- Replace the page which is not used in longest dimension of time in future.

Ref. String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f4				2	2	2	2	2	2	2	2	2	2	2	2	2	2
f3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1
f2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3
	*	*	*	*	hit	*	hit	*	hit	hit	hit	hit	hit	*	hit	hit	hit

hit = 10
fault = 7

3. Least Recently Used :- Replace the least Recently used page in past.

Ref. St. → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

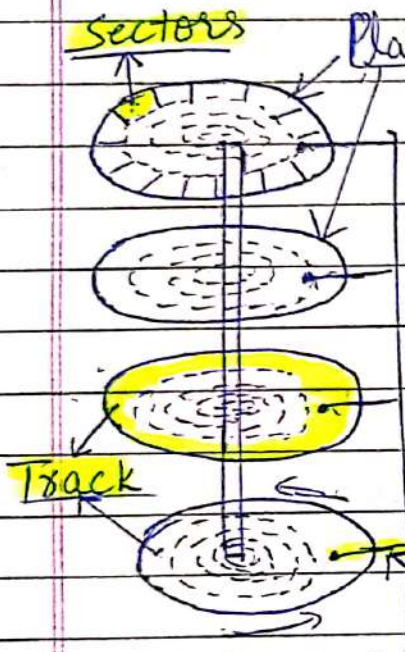
f4				2	2	2	2	2	2	2	2	2	2	2	2
f3			1	1	1	1	1	4	4	4	4	4	4	4	1
f2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
f1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3
	*	*	*	*	hit	*	hit	*	hit	hit	hit	hit	hit	*	

4. :- Most Recently Used :- Replace the most Recently used Page in the Past

6. DISK SHEDULING

**

— Disk Architecture :- In Hard disk architecture, Magnetic disk is a storage device that is used to write, rewrite and access data.



(up/down)
Platters → Surface → Track → Sectors → Data

e.g. 8 x 2 x 256 x 512 x 512 kb

$DS = 2^3 \times 2^1 \times 2^8 \times 2^9 \times 2^9 =$

Disk size $= 2^{30}$ kb
$= 2^{30} \times 2^{10}$ B
$= 2^{40} =$ 1 TB

Disk Size = P x S x T x S x D

* (most Numerical)

— Disk Access Time :- The Total Time required by the computer to process a read/write request and then retrieve the required data from the disk storage.

1. Seek Time :- The time taken by R/W head to reach desired track.

2. Rotation Time :- Time taken for one full Rotation (360°).

3. Rotational Latency : Time taken to reach to desired sector.
→ It is half of Rotation time

4. Transfer Time : Data to be Transfer / Transfer Rate

* Transfer Rate (Data Rate) = No. of Heads x Capacity of one Track x No. of Rotation in one sec

* No. of Head = No. of surface.

* Capacity of one Track = Sectors in one Track x Data in the sectors

* Disk Access Time = ST + RT + TT + QT + CT (OPT)

CT = controller Time (It's optional)
QT = Queue Time (optional).

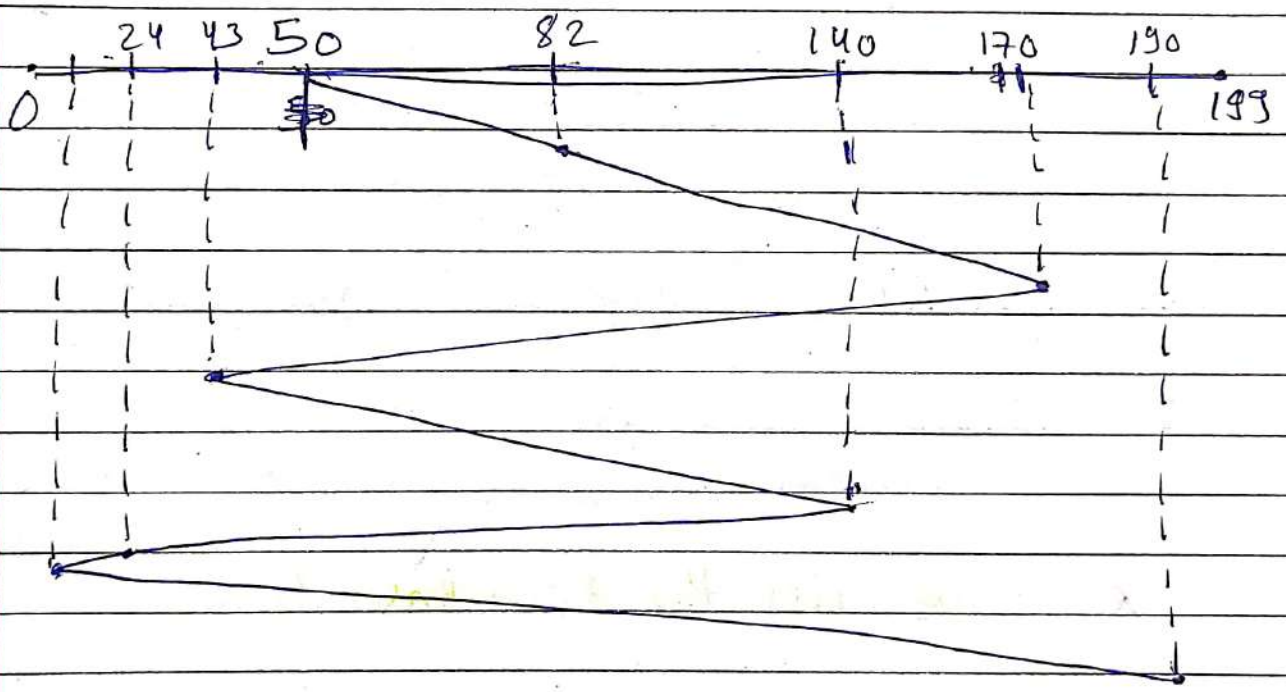
— Disk Sheduling algo. : The goal of this algorithm is to minimize seek time.

1. FCFS (First come First serve)
2. SSTF (shortest seek time First)
3. SCAN
4. Look
5. CScan (Circular scan)
6. CLook (Circular look)

1. FCFS : First come First serve.

Q- A disk contains 200 tracks (0-199). Request queue contains track No. 82, 170, 43, 140, 24, 16, 190. Current position of R/W head = 50. Calculate total No. of tracks movement by R/W head.

Sol.

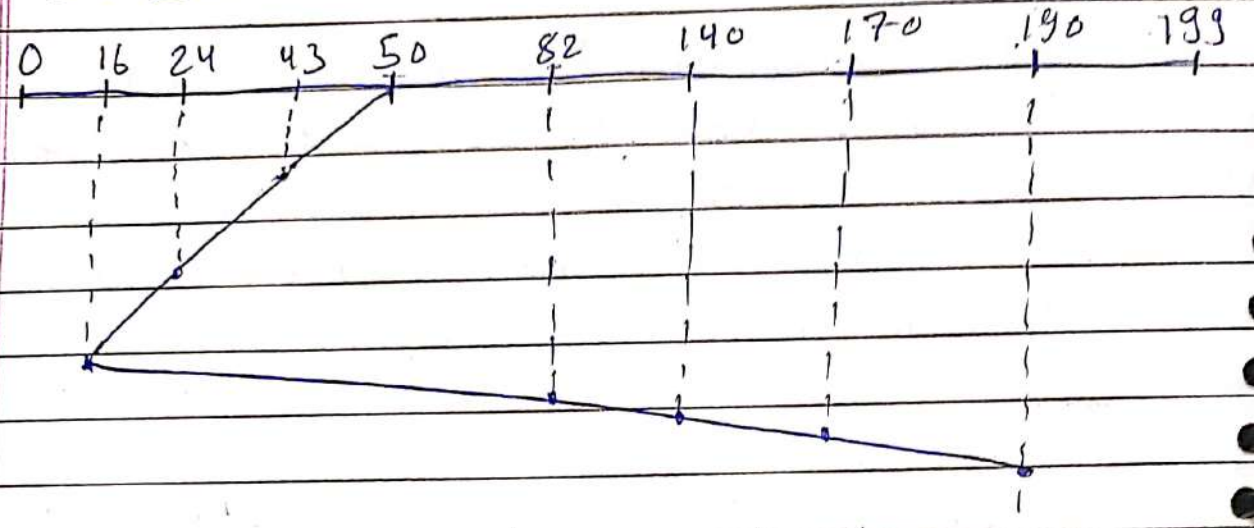


$T = (82-50) + (170-82) + (170-43) + (140-43) + (140-24) + (190-16) = 642$

A

Total No. of Tracks = 642

2. SSTF: (shortest seek time first)

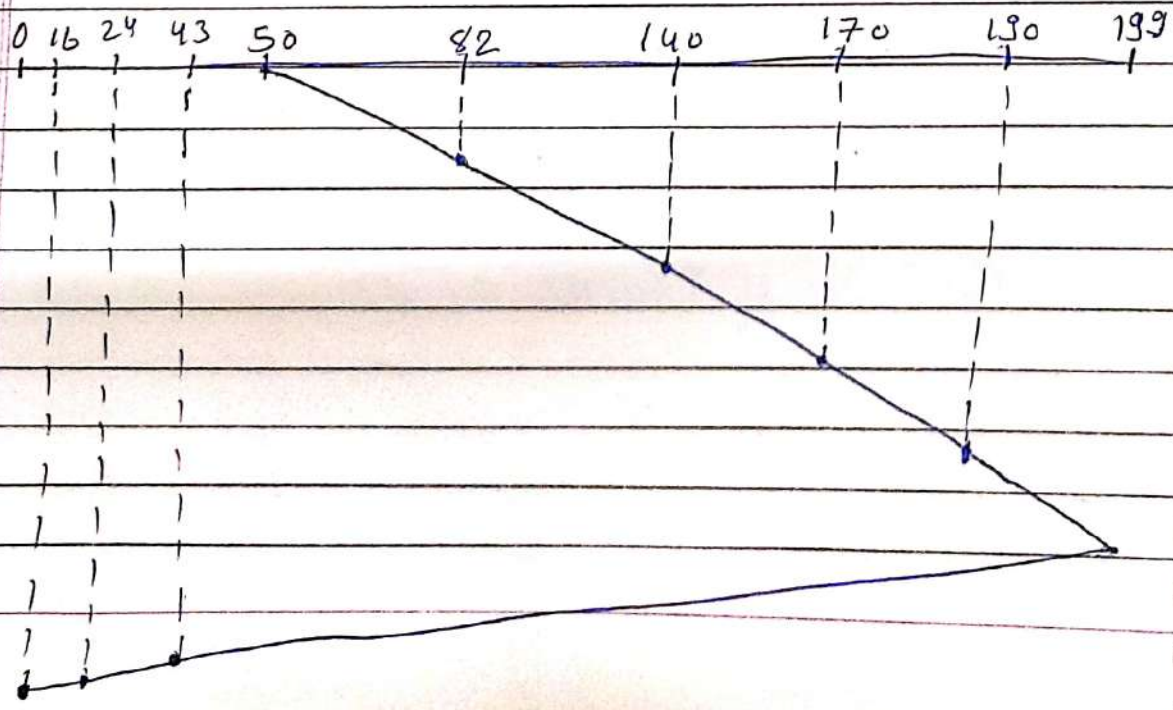


~~FF~~ = Total Tracks = (50-16) + (190-16)
= 208

3. SCAN: (elevator algorithm) (dir. Needed)

Q- Previous question,
Given => lager direction (direction to move in)

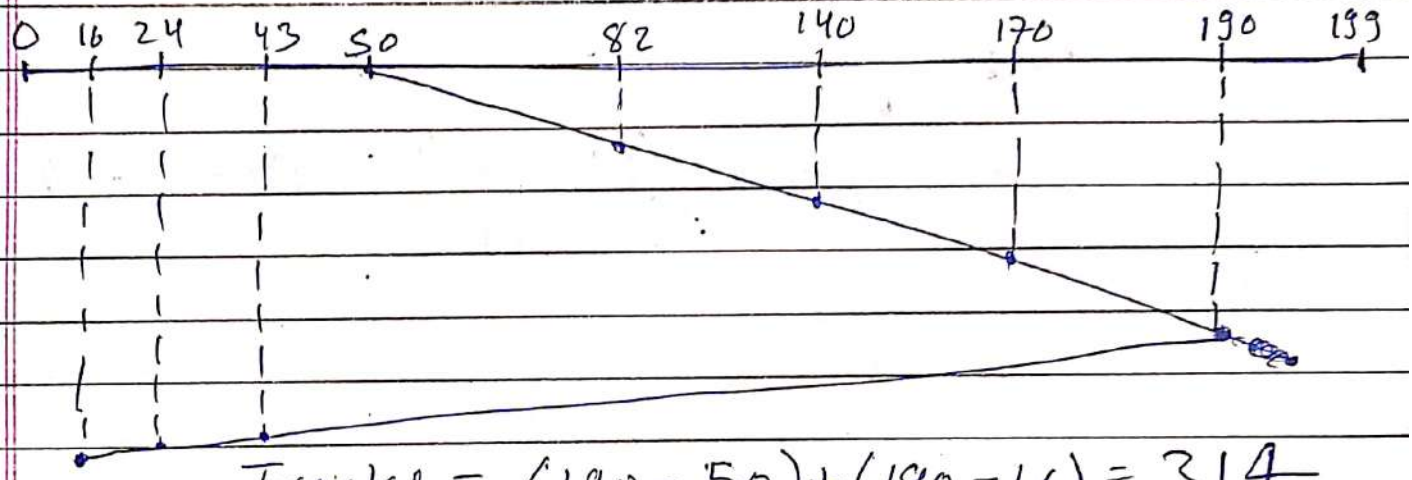
* Move till the end last Point.



Total Tracks = (199 - 50) + (199 - 16)
= (332)

4. LOOK :- (direction Needed)

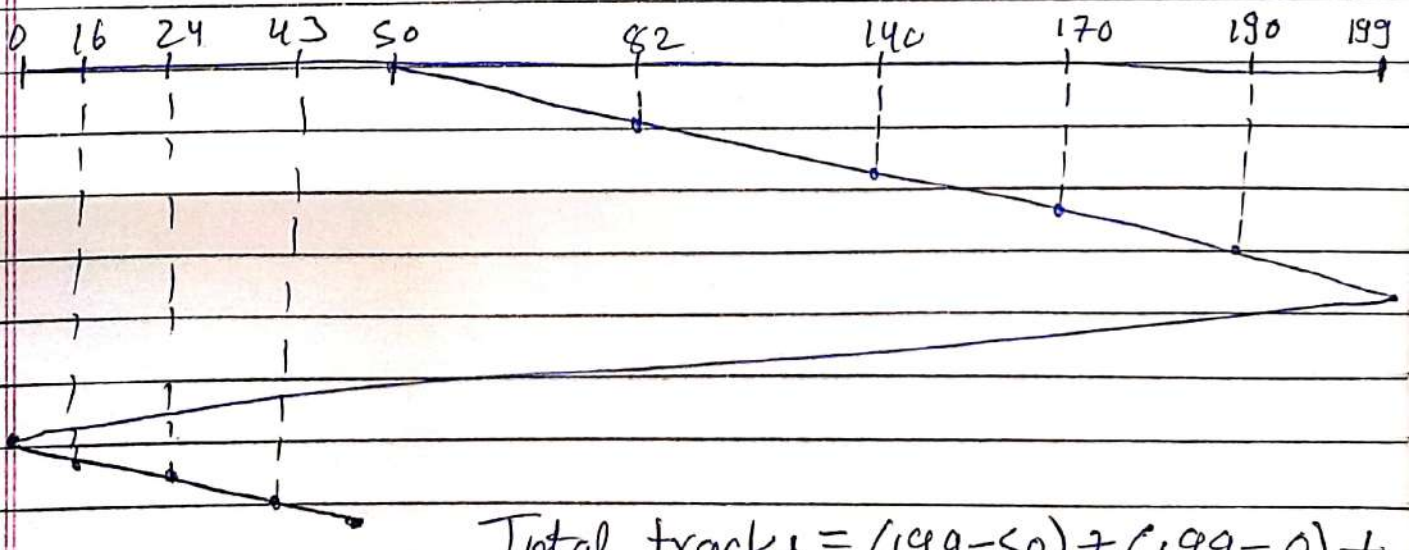
→ In LOOK, don't Move to the last point.



Tracks = (190 - 50) + (190 - 16) = 314

5. CSCAN :- (direction Needed)

* Move to the both ends (0 & 199).

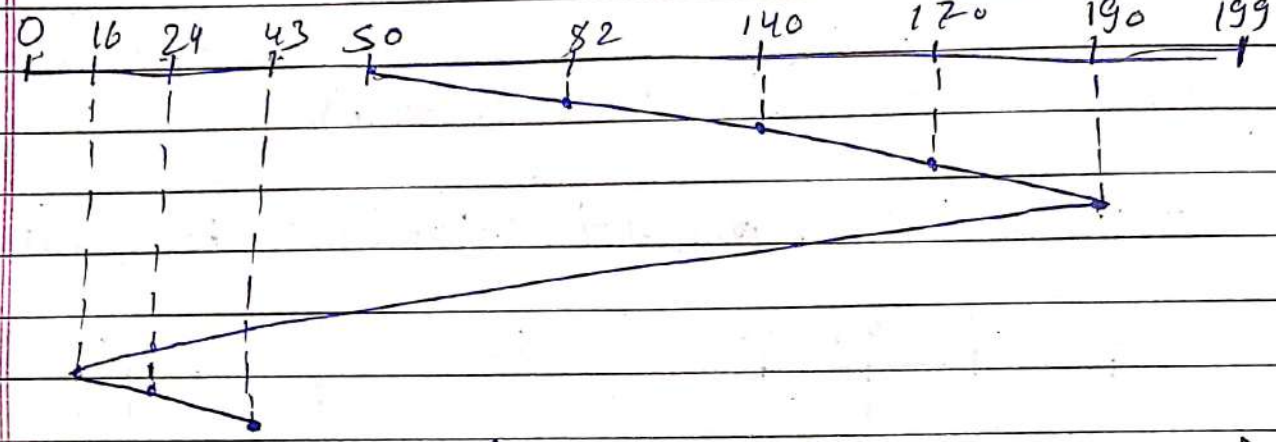


Total tracks = (199 - 50) + (199 - 0) + (43 - 0)

~~Total~~

= (391)

6. CLOOK : (directions needed)



Total Tracks = (190 - 50) + (190 - 16) + (43 - 16)

= 341

7. File System

— File system :- File system is a set of data structures, Interfaces, abstractions, and APIs that work together to manage any type of file on any type of storage device, in a consitent manner.

— Operations on File :-

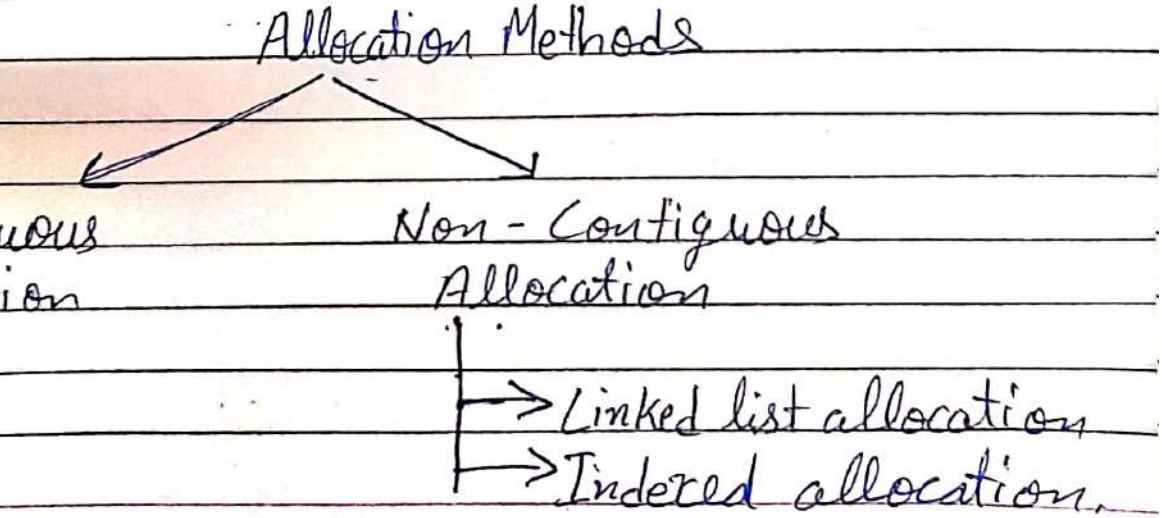
* File :- File is the collection of data / Information

1. Creating
2. Reading
3. Writing
4. Deleting
5. Truncating
6. Repositioning.

— File Attributes :- (meta-data)

1. Name
2. Extension (type)
3. Identifier
4. Location
5. Size
6. Modified date / created date
7. Protection / Permission
8. Encryption / Compression

—



1. Efficient Disk Utilization
2. Access faster.

— Contiguous Allocation :- Allocate files in continuous manner.

* Advantages :-

1. Easy to Implement
2. Excellent Read Performance.

* Disadvantages :-

1. Disk will become fragmented
2. Difficult to grow file.

— Non-Contiguous Allocation :-

1. Linked list Allocation :- Each file is considered as the linked list of disk blocks.

- Linked List :- A sequence of data structures, which are connected together via links.

- Advantages :-

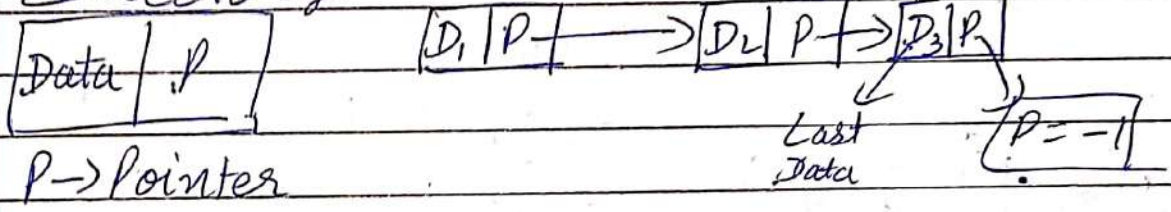
1. No external fragmentation
2. File size can Increase.

- Disadvantages :-

1. Large Seek Time
2. Random Access / Direct Access difficult.
3. Overhead of Pointers.

→ We use Pointers.

Directory



P → Pointer

→ Last Pointer value is '-1'.

2. Indexed Allocation :- It stores all the disk Pointers in one of the blocks called as Indexed block

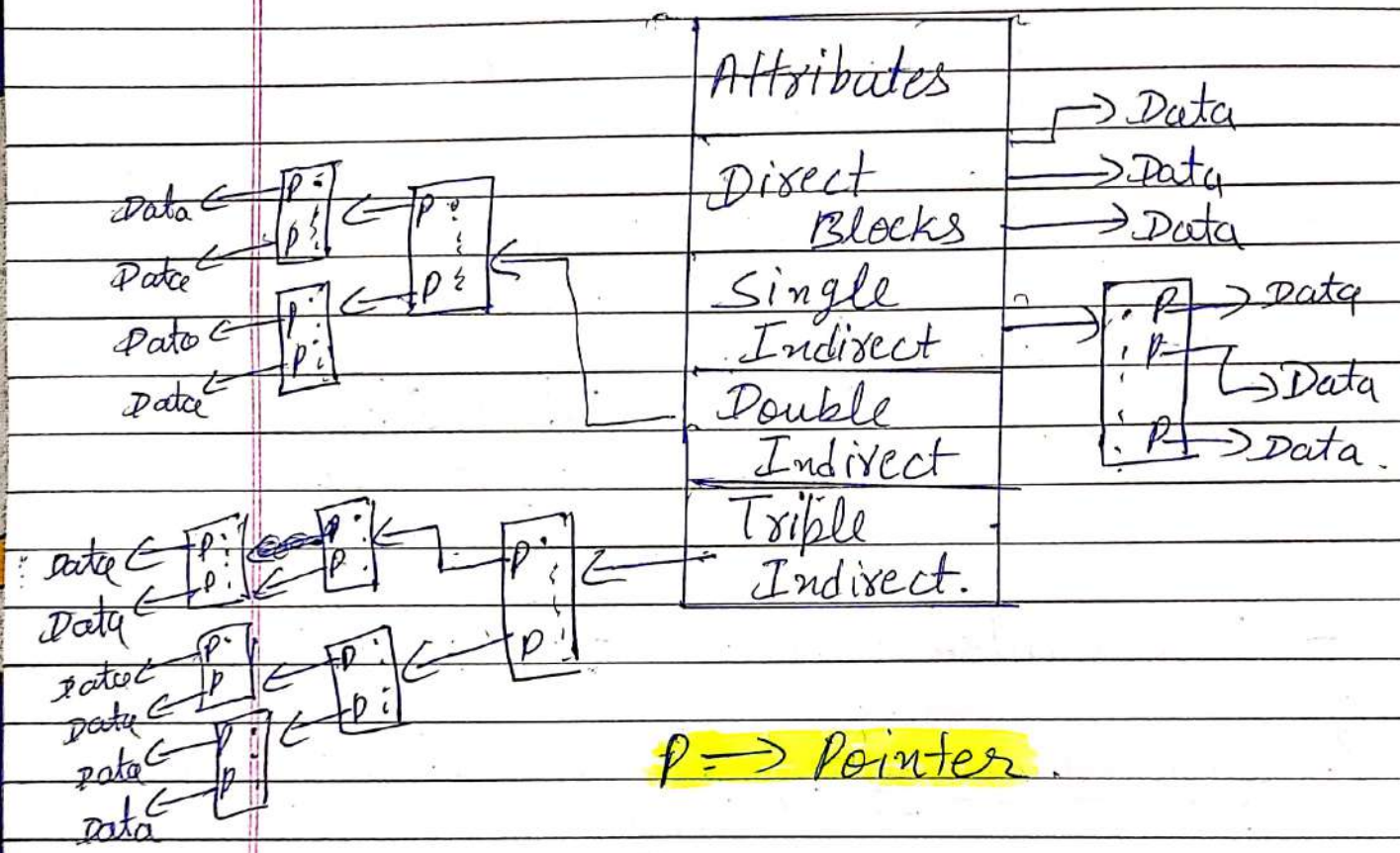
- Advantages :-

1. Support Direct Access
2. No External Fragmentation.

- Disadvantages :-

1. Pointer Overhead
2. Multilevel Index.

Unix Inode Structure :-



P => Pointer.

8. Protection & Security

(Not so Important)

Date
Page

— Goals of Protection :-

1. In one Protection Model, computer consists of a collection of objects, hardware or software.
2. Each object has a unique name and can be accessed through a well defined set of operations.
3. Protection Problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

** * CIA = confidentiality Integrity Availability.

— Security violation Catagories :-

* 1. Breach of Confidentiality :- Unauthorized reading of data.

* 2. Breach of Intigrity :- Unauthorized modification of data

* 3. Breach of Availability :- Unauthorized destruction of data.

4. Theft of Service :- Unauthorized use of resources.

* 5. Denial of Service (DOS) :- Prevention of Legitimate use. (Pinging)

— Principles of Protection :-

1. Program, users and systems should be given just enough Privileges to perform their task
2. Limits damage if entity has a bug, gets abused.
3. can be static.
4. Dynamic (Changed by Process as needed).

— Domain Structure :-

— Domain :- set of access Rights

D1.

<O3, (R, W)>
<O1, (R, W)>
<O2, (X)>

D2.

<O2, (Write)>
<O4, Print>
<O1, (X)>
<O3, (X)>

* — Access Matrix :- It is a ~~view~~ view protection as a matrix.

Virus :- A malicious program loaded onto a user's computer without the user's knowledge & performs malicious actions.

Worm :- It is a malware that reproduces itself and spreads over network conections.

Malware :- It is an Intrusive software that is designed to damage and destroy computers & computer systems.

Worm vs. Virus :- A worm is similar to a virus by its design, and is considered to be a sub-class of a virus. Worms spread from computer to computer, but unlike a virus, it has the capability to travel without any help from a person.