Operating System

→ In C, the program execution (./a.out file) is done by OS compiler is just addition to OS.

→ But in JAVA Execution is done by JVM and hence JAVA is platform independent.

→ User point of view:

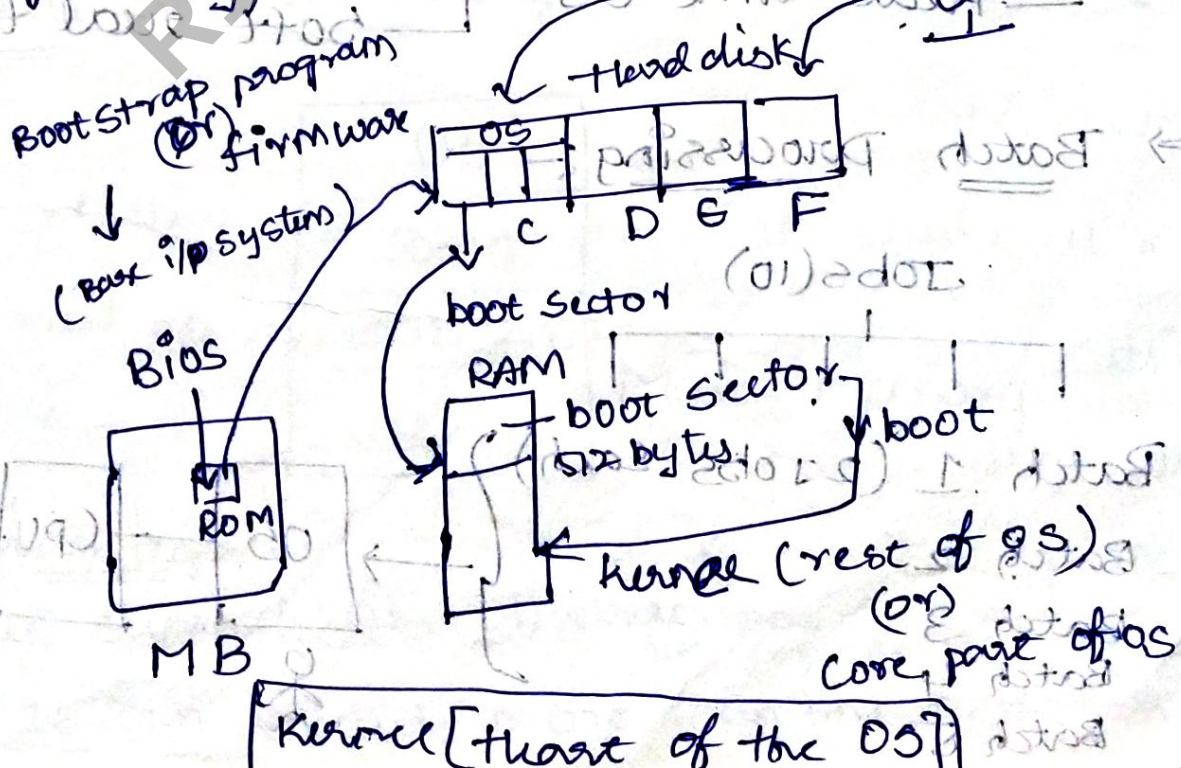
Operating System:-

→ It is a system software that acts as a interphase Between user and computer.

→ System point of view:-

→ The OS can be defined as resource allocater (or) resource manager.

→ A resource can be (a hardware component or it can be software component (Application Programing).



OS → functioning

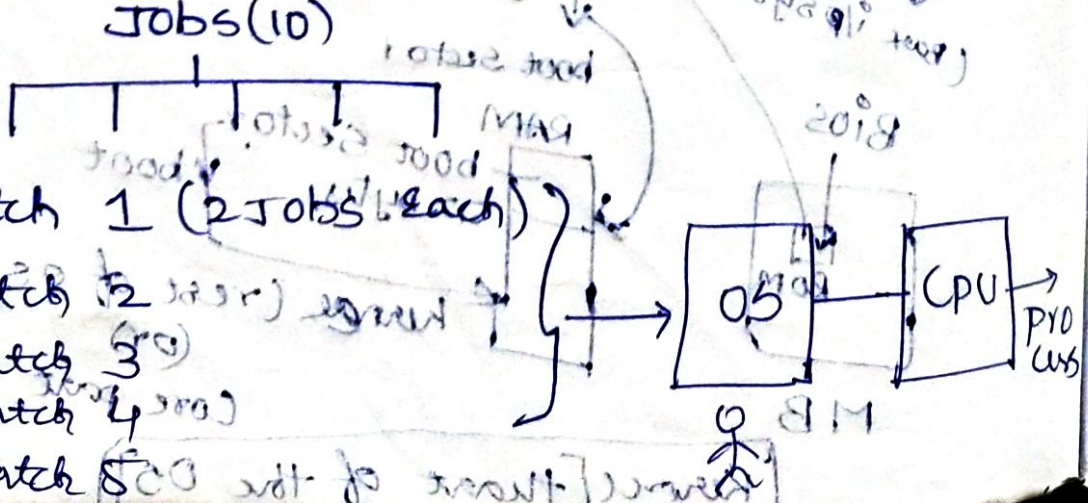
Objective or Goal or core functionality of

- convenience → Easy to use, operate, understand, implement
- Efficiency
- ability to evolve
- protection and security

Types of Operating System:

- Batch processing
- Multi programming
- Multi tasking
- Multi processing
- Distributed
- Network
- Real-time OS
  - Hard real time
  - Soft real time

⇒ Batch Processing



Batch processing :-

→ Jobs are devided into batches. The jobs in each batch will have a similarity.

Ex :- Payroll System,
IBM's MVS OS.

DisAdvantages :-

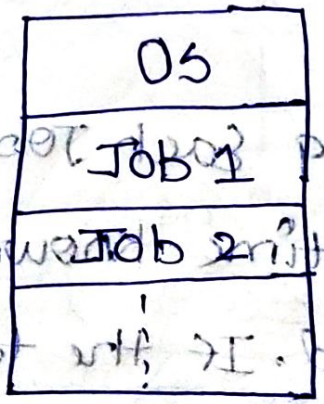
→ No user interaction in Batch processing with CPU.

→ only one Job will be Executed once.

→ If a batch has a problem next batch is will not be Executed.

→ In order to Execute stop that job & we can continue with other jobs; but in some cases order must be followed in order to complete work, which follows sequential steps, Entire work will be stopped.

Multi programming :-



Advantages :-

→ It supports user interaction.

→ It can Execute more than one program.

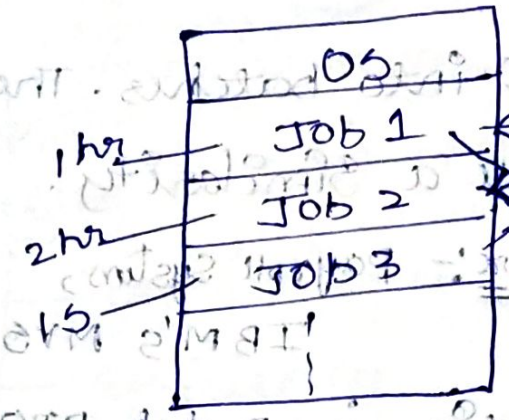


Fig-(1)

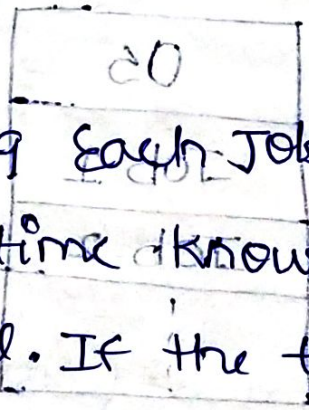
→ For eg :-

if i/o operation of Job 1 is performing we can assign CPU to Job 2 instead of keeping CPU idle.

Disadvantages :-

→ Here, Job 3 can be executed (or) processed time is less but he has to wait for more time until Job 1 and Job 2 gets processed. (Starvation happens i.e. last person should be waiting for longer duration)

Multi-tasking :-



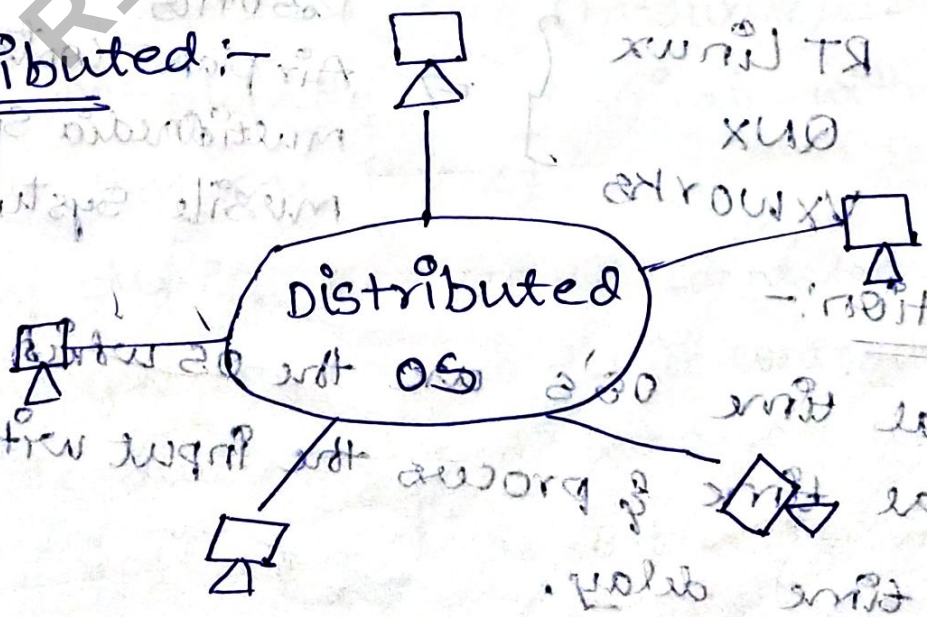
→ In multi tasking each Job will be Allocated certain time known as time slice for getting Executed. If the task is not completed in given time slice it has to wait till it get its time slice.

→ If we consider above eg: job 3 has to wait only for 120 seconds.
→ So starvation time and CPU idle time is decreased & CPU is always Busy.
→ User interaction is allowed.

Multi-Processing :-

→ It Enables the concept of concurrent execution of more number of processes at a time.
→ It will have more number of processors
For Eg:- It performs Job 1 and Job 2 and so on at a time.
→ Processes are performed in parallel at a time.

Distributed :-



→ Entire OS will not be in client machine it will be working over the network.

→ The client will have module (or) layer of OS which can communicate with network.

Eg:- OS's of Indian Railways

Network:

→ The Network OS generally resides in the Server Computer, that serves the number of clients.

→ There are 2 types of Real-time OS:

- Real time OS
  - Hard real-time
  - soft real-time

Ex:- RT Linux, QNX, VXWorks } → Robotics, Air plane control system, Multimedia System, Missile System

Definition:-

→ Real time OS's are the OS which work in real time & process the input without any time delay.

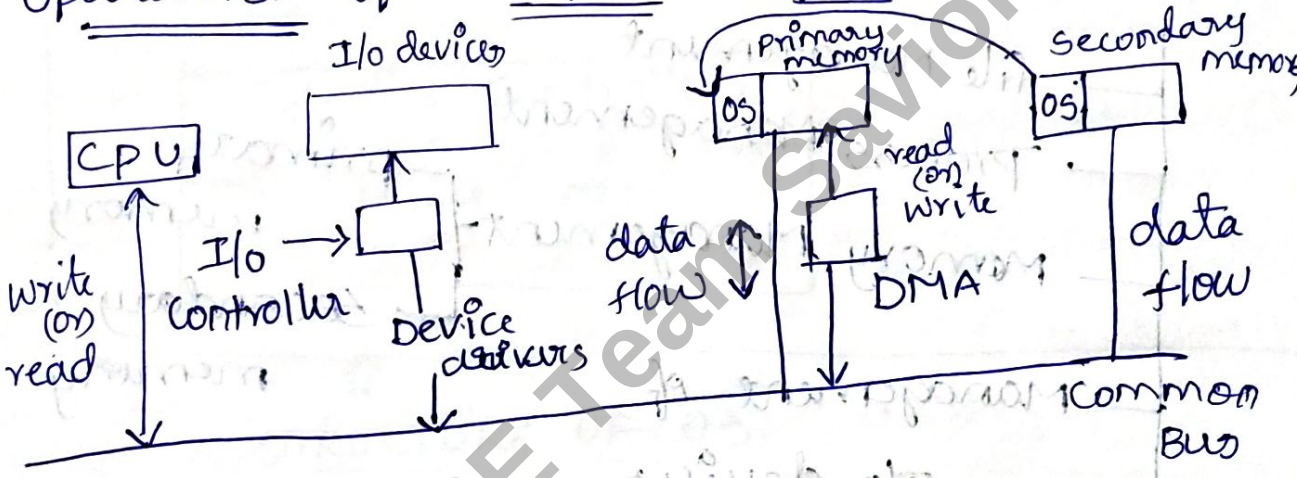
→ If within a deadline if system could not deliver output it creates a disastrous consequences.

→ Soft real time :-

→ It is the OS in which the time delay may be accepted to certain extent.

Ex :- Banking System

Operation of computer system :-



→ Device drivers is a software that acts as an interface between device controllers & OS.

(hardware)

→ OS cannot directly communicate with hardware directly that can connect.

→ Every device that can be connected to a computer generally have a device controller, it is inside the device, it is a hardware component which has its own local buffer (its own memory).

→ The specialised software which is designed to perform a specific operation are called drivers.

OS → device driver → device controller
↓
output / IOs

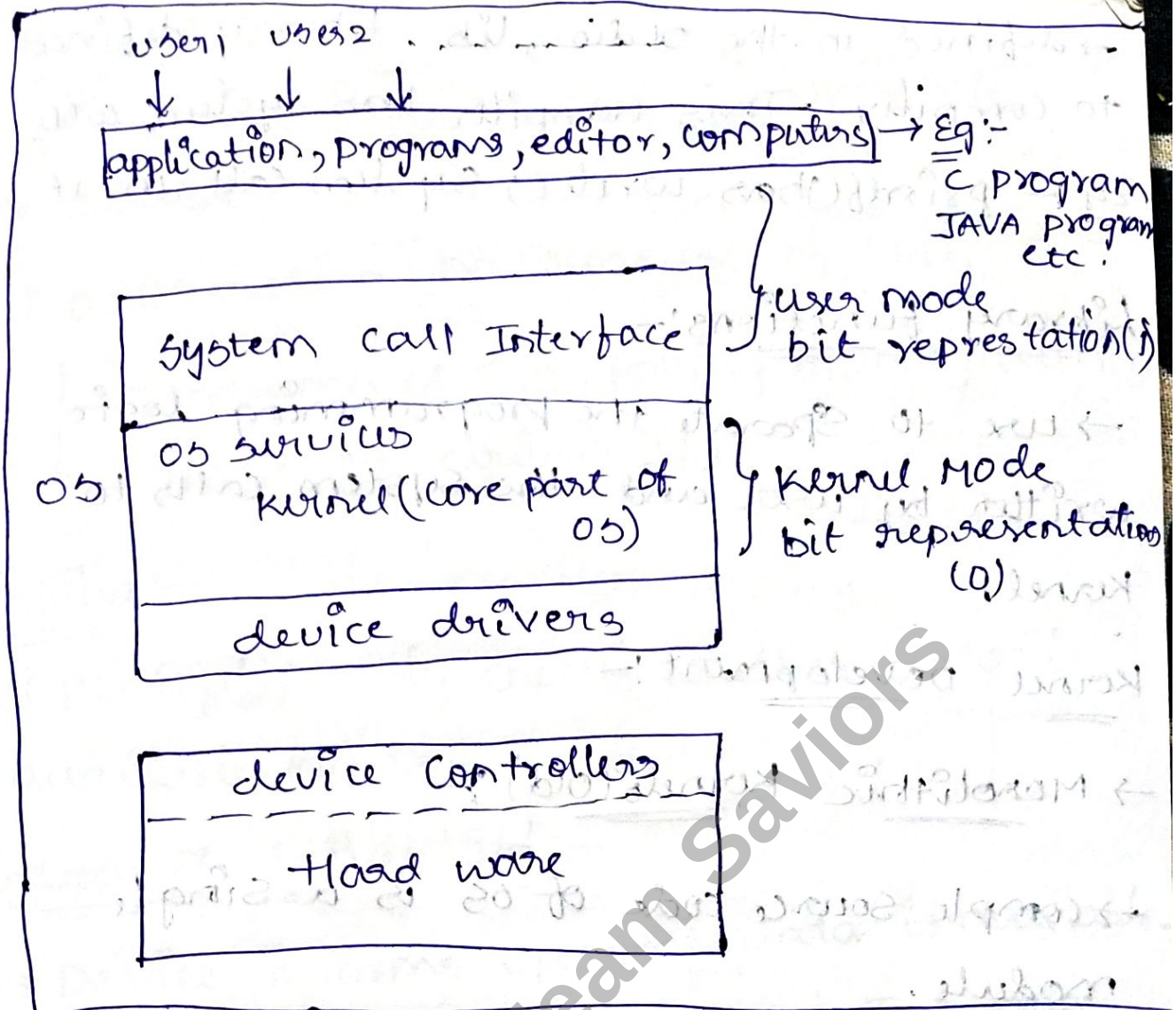
OS Services:-

- file Management
- process Management
- Memory Management
  - primary memory
  - secondary memory
- Management of I/O devices
- Networking
- Protection & Security

1) interrupts
2) system calls
3) Kernel (mode of operation)

Operating system structure

⇒ If any thing is loaded into primary memory then it becomes process.



STRUCTURE OF OS

→ Any action of the OS to the request of an user is done through the Kernel of the OS.

→ Only Kernel can access hardware.

API :- Application programming Level Interface

→ These are Interface with some system calls. These system calls are called in the compiler.

Eg:- write() is a system call

printf() by user calls the printf definitio

→ defined in the stdio, lib library defined to compiler. This compiler has system calls

Eg:- printf() has write() system call in it

Library functions:-

→ use to separate the programming logic written by user and the system calls to kernel;

Kernel development:-

→ Monolithic kernels (old) :-

→ comple source code of OS is a single module.

Ex:- windows 95, 98 | Linux

→ Micro kernels (new)

Ex:- Advanced windows, QNX

→ Each services of OS is a module.

→ Kernel only controls the module.

Interrupts:-

→ Interrupt informs CPU that data is available.

I/O operation (on request of CPU)

- Programmed I/O, polling or busy waiting
- Interrupt driven I/O

Polling:-

→ CPU continuously checks whether data is available (or) not.

Interrupt driven I/O:-

→ Device informs CPU that data is available

→ OS is event driven & CPU is Interrupt driven

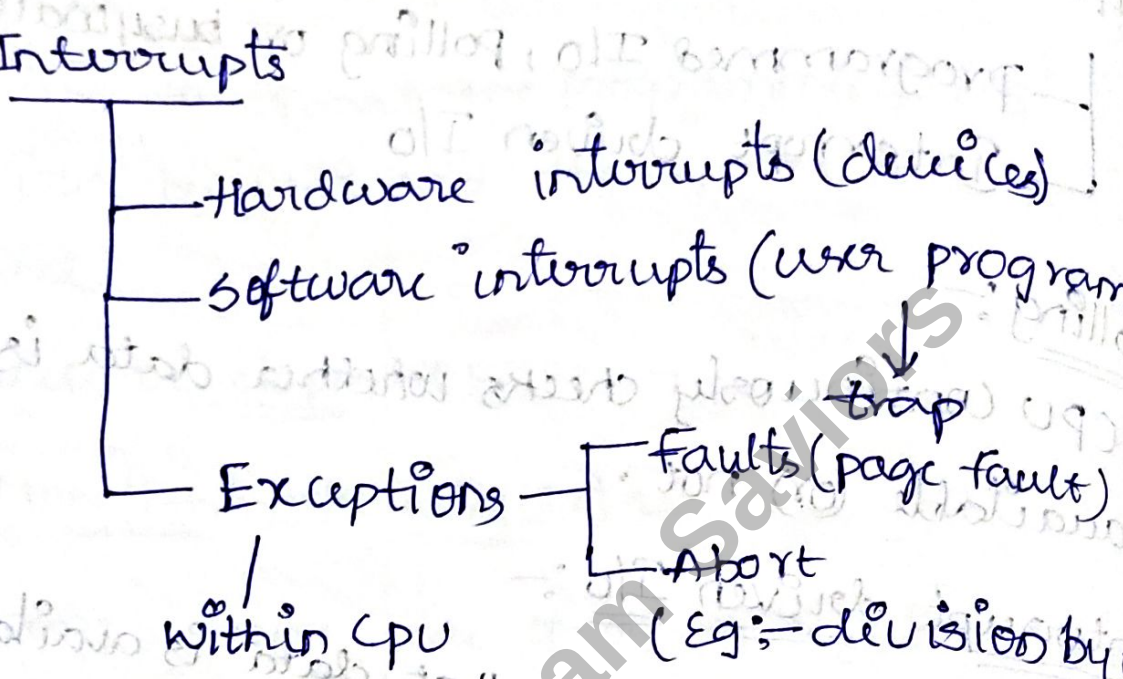
Interrupts:-

→ It is an event generated by the a resource (can be hardware (or) software) that interrupts the CPU to get the attention of CPU towards it.

→ When Interrupt happens the context of the current job scheduled on CPU is saved then CPU is switched to the Interrupt handler with can proces the interrupt

Service routine (part of OS) is invoked.

→ Later it resumes the old job (or) it may Execute new job based on the type of the interrupt (or) Job.



→ Exceptions are interrupts that occurs within CPU.

→ All the system calls are software interrupts.

Memory Mapped I/O:-

→ The device buffers are stored in RAM, so we require only one bus.

System call:-

→ It provides the functionality of OS to the user application.

→ It acts as a interface B/w application Program and Kernel.

→ It seems like a function call but when it is called the state of the system is changed from user mode to Kernel mode.

→ When system call is invoked a piece of code gets executed by interrupting the CPU, through interrupt service routine.

→ User function call and its definition may be relocated in the memory but system calls & it's definition is at the same location Even it is called multiple times & at different intervals of time

→ OS supports hundreds of system calls.

System calls in UNIX :-

→ Process Management
(fork(), wait(), exit())

→ file Management
(open(), read(), write(), close())

→ device Management
(read(), wait())

→ Information Maintanence
(getPid(), alarm(), sleep())

→ Communication
(pipe(), shmget())
→ security
(chmod())

OS services: [continued]

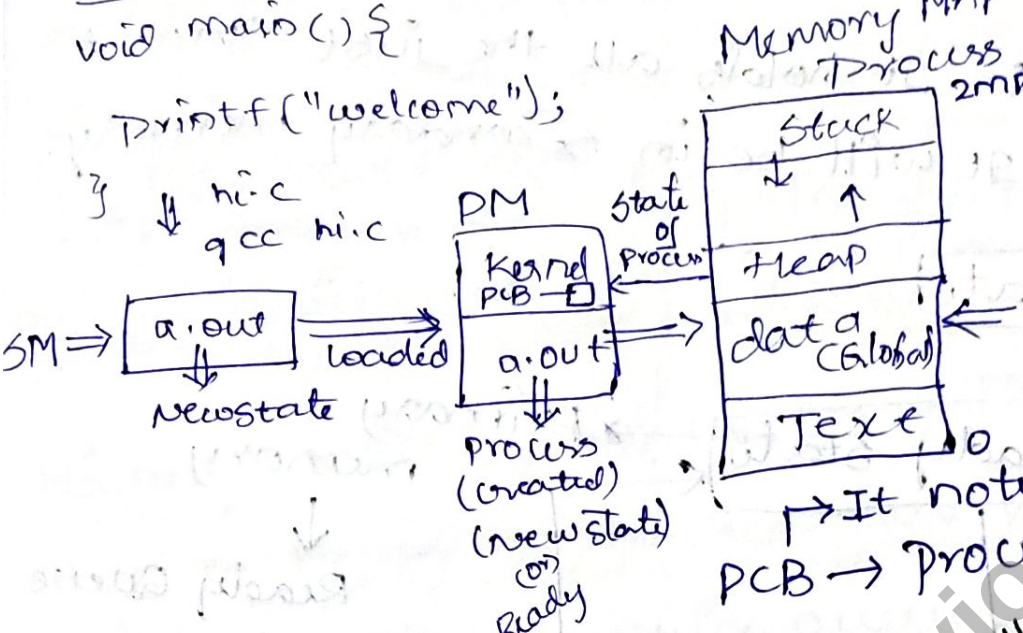
→ Accounting
→ error detection
→ resource allocation

* PROCESS MANAGEMENT :-

Program

```
void main(){
  Printf("welcome");
}
```

hi.c
↓ gcc hi.c



↳ It notes the state of block

PCB → Process Control Block

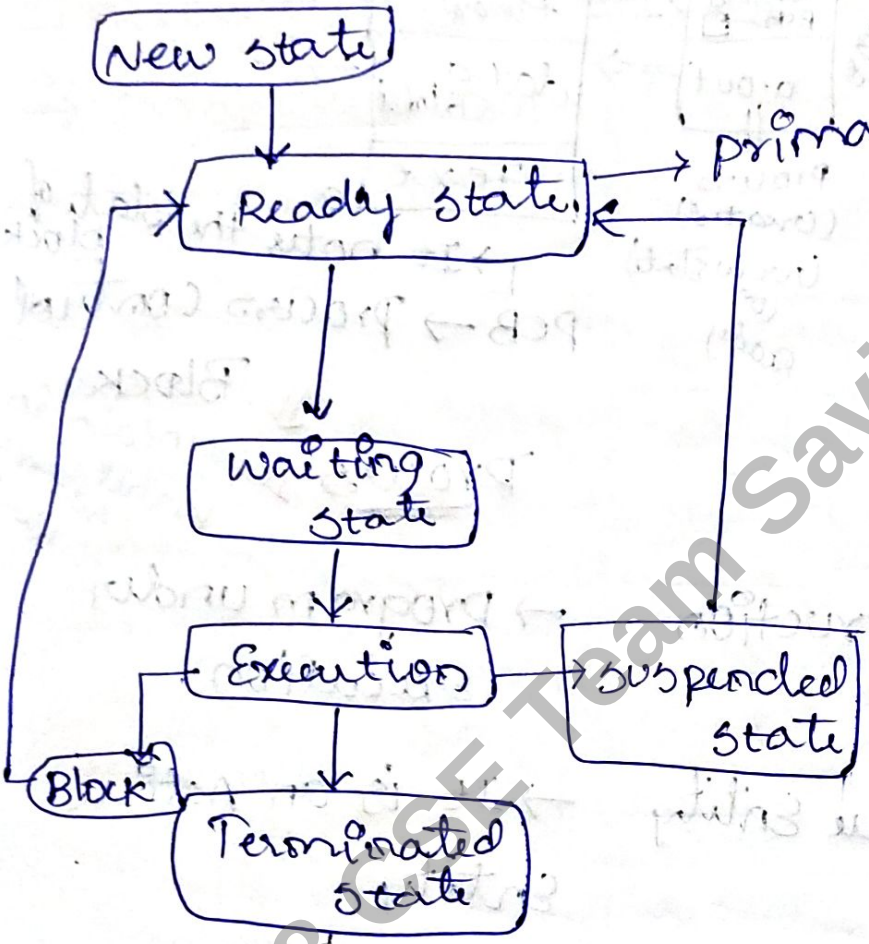
Process → It Maintain in Kernel Address block

Program	Process
→ set of Instruction	→ program under execution.
→ It is passive Entity	→ It is an Active Entity.
→ life span is longer	→ lifespan is shorter
→ It is on secondary storage.	→ It is in primary memory.
→ It won't use Any resource.	→ It uses many resources & OS.

Process state :-

New state — a.out → secondary memory

Job pool :- It holds all the just started process. It will be in secondary memory.



New State :-

→ The process just now Created.

Ready state :-

→ The processes which are readily to get Executed.

Waiting State :-

→ The processes which are in the ready queue & waiting for the attention of CPU.

Executing State :-

→ The process which is getting executed.

Blocked State :- The processes which are temporarily blocked & waiting for the I/o operation are said to be in the blocked state & they are maintained in the device queue.

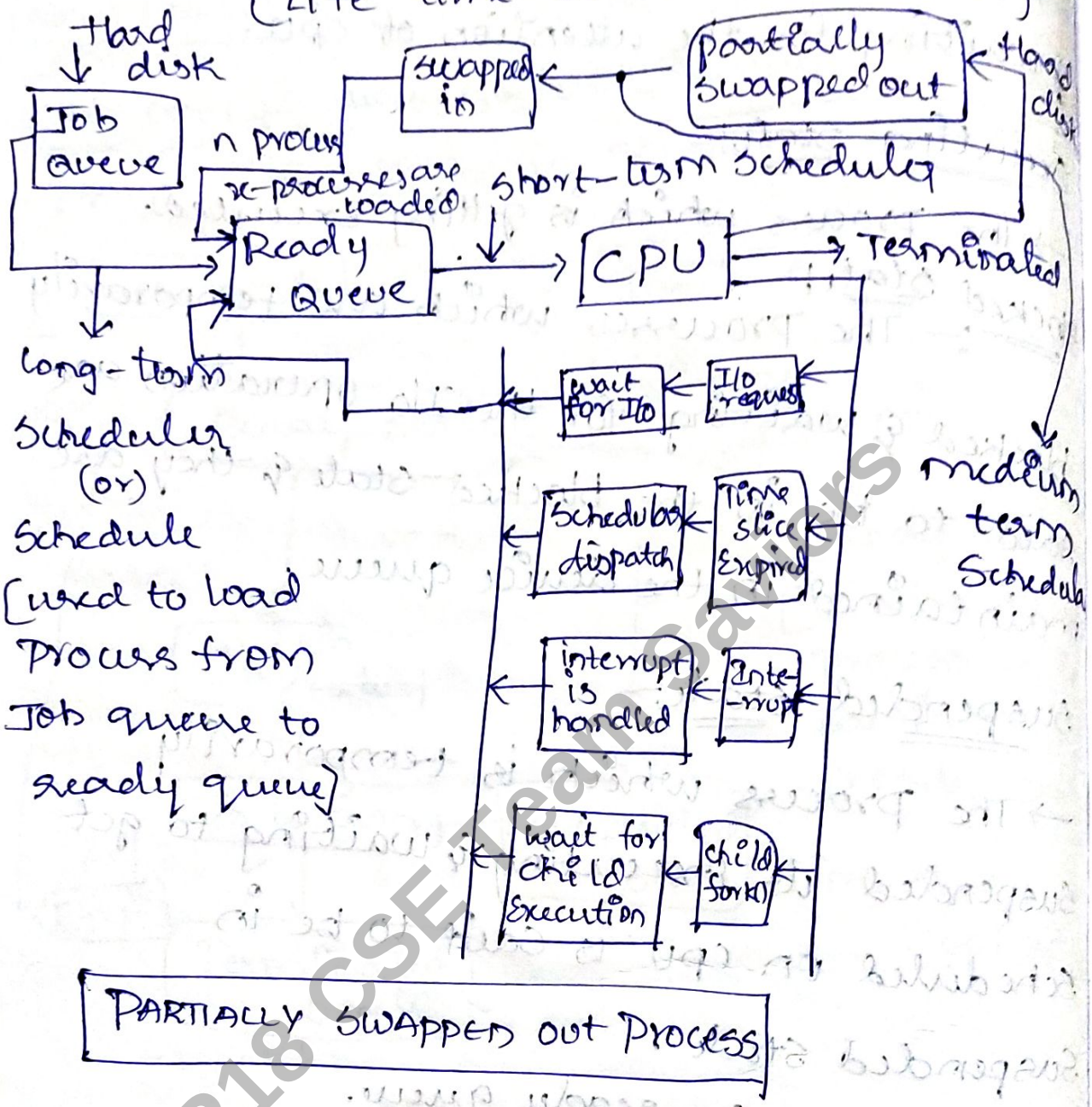
Suspended State :-

→ The process which is temporarily suspended its executing & waiting to get scheduled on CPU is said to be in Suspended state.

→ These are in ready queue.

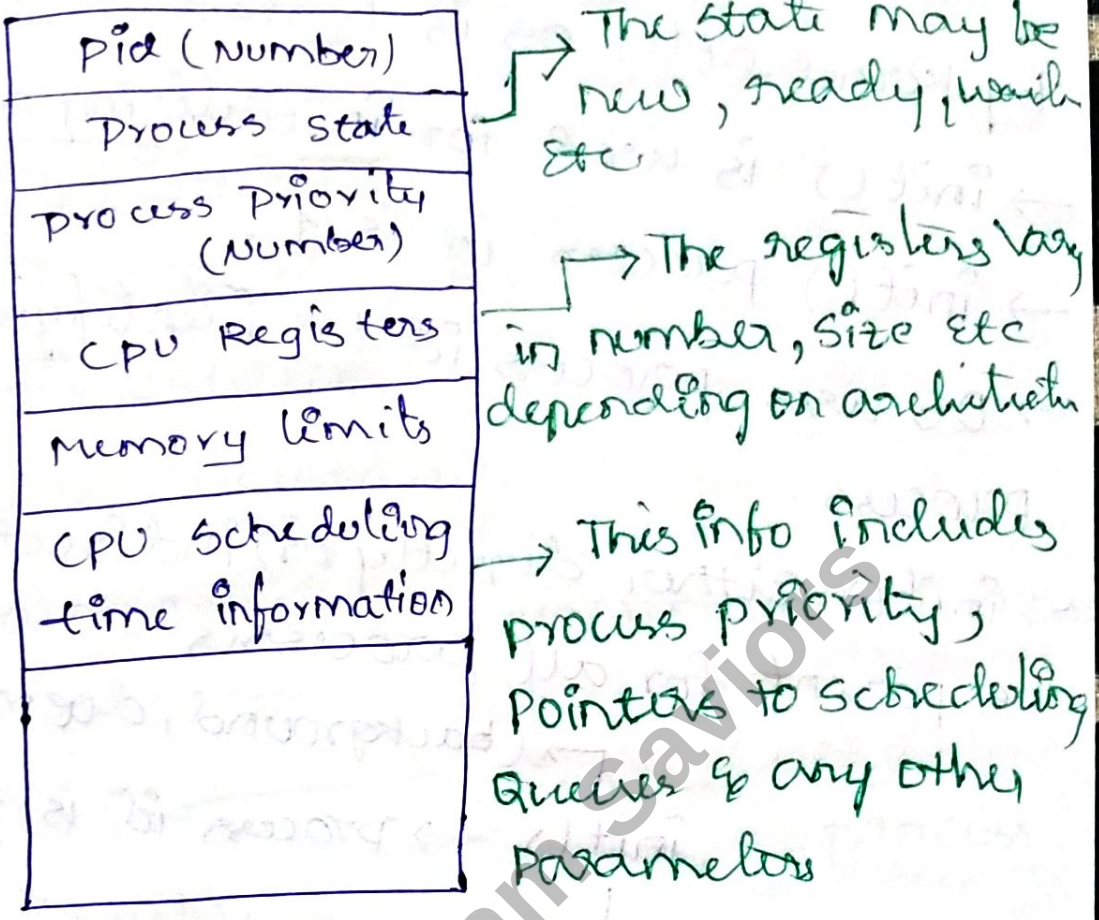
Queuing diagram of a process

(Life time states of a process)



→ The maintainence and management of a process is done through a special data structure allocated in kernel address space by the OS called as PCB (process Control Block).

PCB (Process Control Block):-



Process Management:-

- Process creation
- Process execution
- Process scheduling (FIFO, SJF, priority based, Round Robin, LRU)
  CPU scheduling
  - preemptive
  - non preemptive
- Process Synchronization
  (Software based, Hardware based, Semaphores, locks, Monitors)
- Inter Process Communication
  - message passing
  - shared memory
  - pipes (named pipes, unnamed pipes)
- Process terminating

→ init() :-

It is the first process getting executed by kernel after OS is booted.

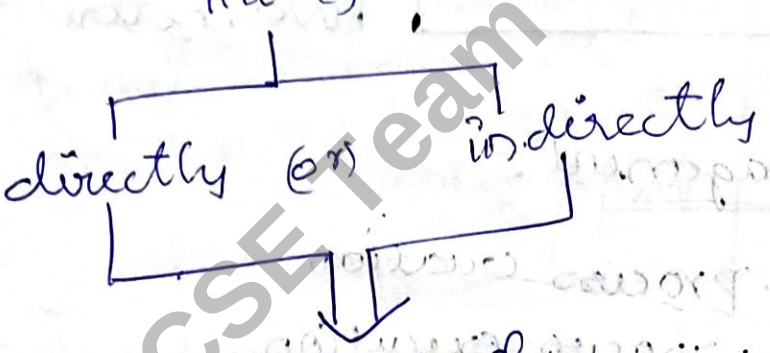
→ init() is used for initialising drivers.

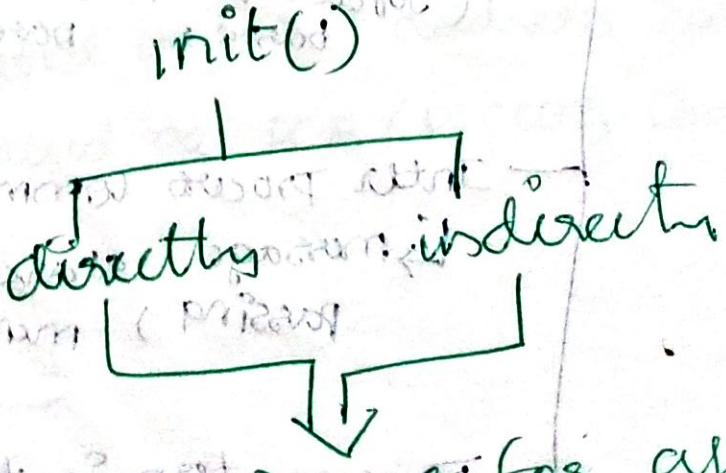
→ init() process id is 1.

→ OS uses process id for identifying a process.

→ init() either directly (or) indirectly will be parent for all processes

→ (background, daemon process)



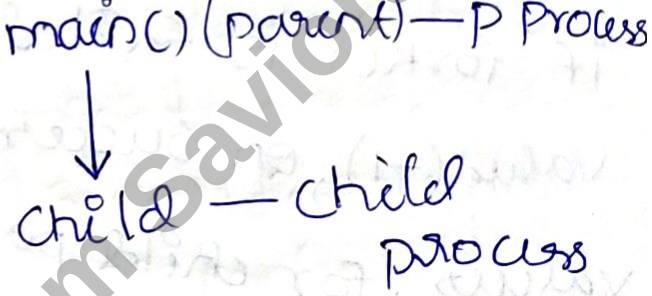


Creation of process : fork()

→ fork() is used to create new process.

Program :-

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  fork();
Printf("Hi");

 return 0;
}
```



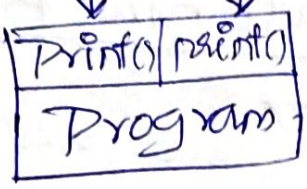
Output :-

Hi

Hi

→ fork() is a system used to create a child process.

→ The child and parent processes both share the same memory same from the point of creation of child process.



→ shares same memory space from point of creation.

→ parent will be executed & concurrently child will be executed.

→ fork() will return an integer value, if fork() is failed it returns negative value (-1), on success it returns two values, for child process it returns zero, for parent it returns the process id of child.

Program :-

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
printf("The process is %d, Kernel id is %d",
getpid(), getppid());
return 0;
}
```

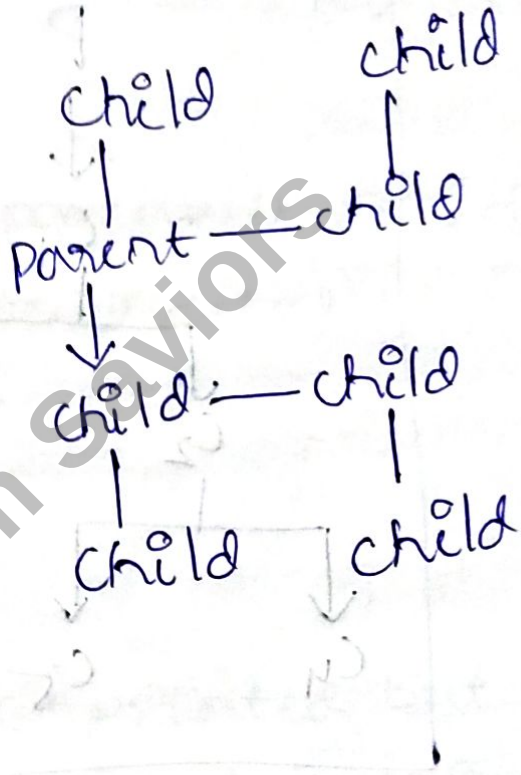
The process is 21525
Kernel id is 15455

getpid() → It may vary at different times

getppid() → constant

Program

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
  fork();
  fork();
  fork();
  printf("hi");
  return 0;
}
```



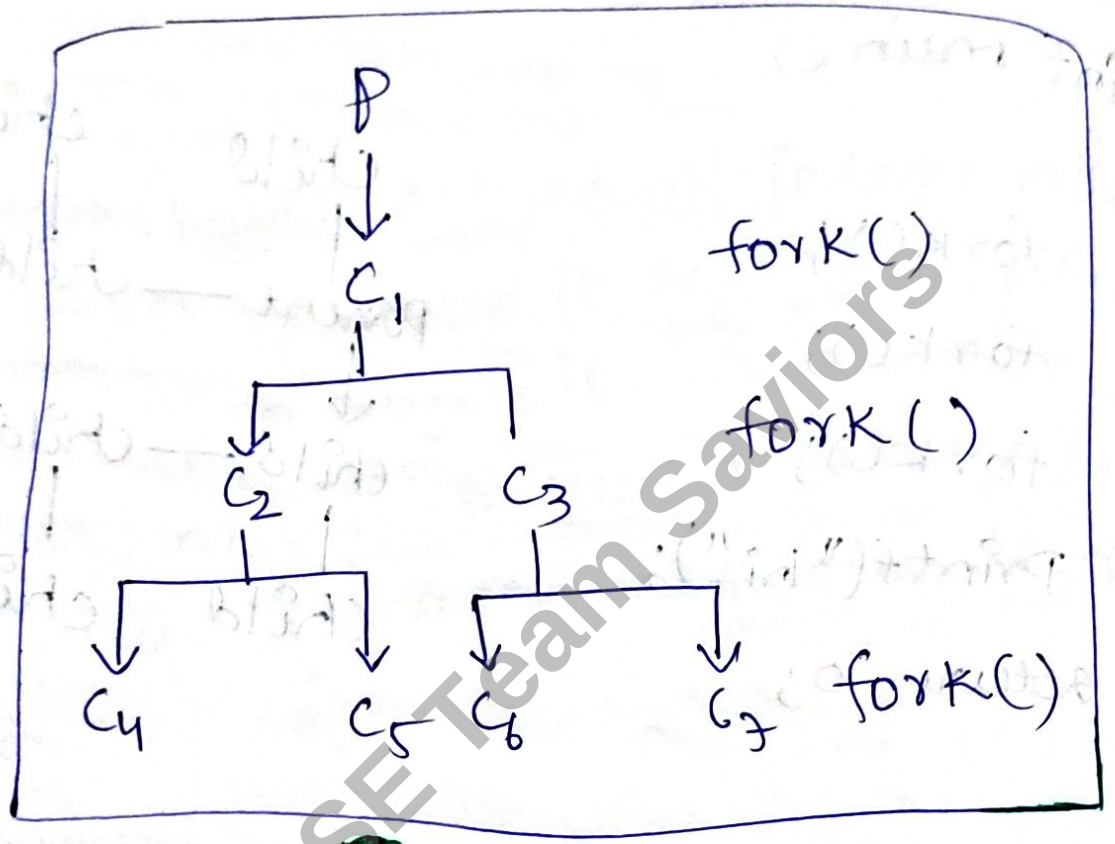
Output:

hi
hi
hi
hi
hi
hi
hi
hi

⇒ for (i=0; i<n; i++)

fork();

∴ There will be 2^n processes and $2^n - 1$ child processes



Program

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
  pid_t pid;
```

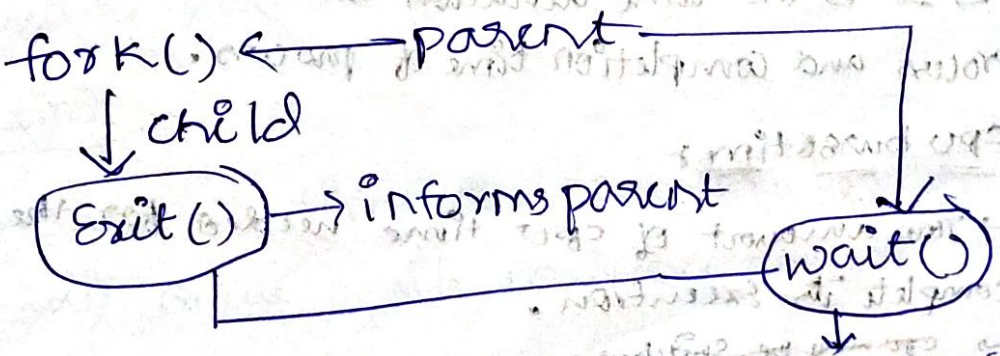
pid_t

```
pid = fork();
if (pid < 0)
  printf("fork is failed");
if (pid == 0)
   printf("welcome to child");
else
 {
  printf("welcome to parent");
  printf("if, else both Executed");
 }
return 0;
}
```

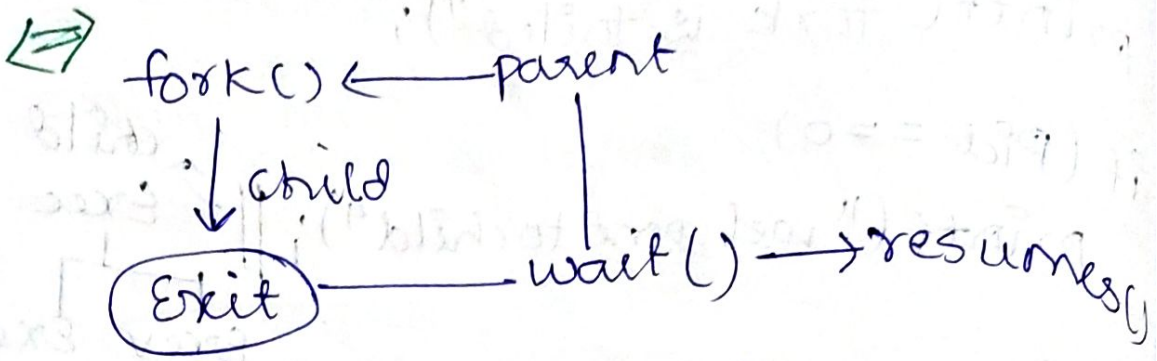
child
Exec
Exev Exel

→ Both if & else gets printed but like in multithreading any block if (or) else may print first.

→ Exec is used to replace the parent memory space with a new process executed by the child.



→ When the execution of child thread is completed it invokes exit().



→ If no parent is waiting for a child who is under execution, if child completes execution, such processes is called as "Zombie process".

→ If parent is terminated ↗ forcefully (abort) but child is in execution, such a child is called "Orphan child".

WAITING Time :-

→ The amount of time the process spends in the ready queue while waiting for CPU.

Turn Around Time :-

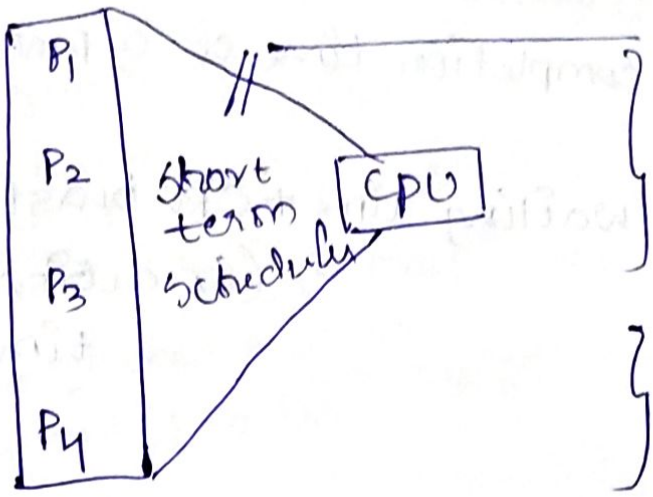
→ It is the time duration b/w the arrival time of a process and completion time of process.

CPU burst time :-

→ The amount of CPU time needed for the process to complete its execution.

→ CPU may be switched.

CPU scheduling (or) Process scheduling :-



FIFO
Shortest Job First (SJF)
Priority based
Round Robin (time slice)
multilevel Queue
multilevel feedback Queue scheduling.

Objectives of scheduling :-

1) To reduce CPU idle time.
2) To improve through put of the system. (no of process getting Executed in particular time).
3) To improve the multi-Programming
4) Utilization of resources
5) To reduce average waiting time, turn around time of a process.

Arrival time :-

→ The time at which the process is arrived into the ready queue.

Waiting time :-

→ The amount of time the process spends in the ready Queue while waiting for the

Turnaround time :-

→ It is the time duration blw the arrival time of a process & completion time of a process.

Turn around time = waiting time + CPU burst time (Execution time)

CPU Burst Time :-

→ The amount of CPU time needed for process to complete its execution.

Eg :- FCFS (First come First Serve) [If not mentioned take same time]

Q)

Process	CPU burst time (ms)	Arrival time
initial time to P_1	27	0
P_2	3	0
P_3	3	0

Find avg waiting, turn around time?

NOTE :- If arrival time is not mentioned, all processes are present in ready queue at $t = 0$.

CPU scheduling

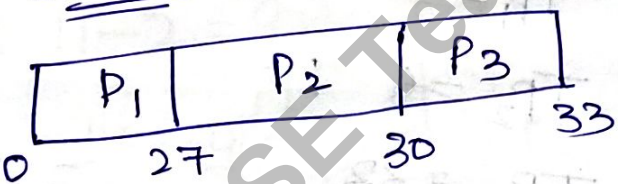
preemptive
↓
when CPU is executing a process the CPU may be switched out to another process.

Non-preemptive
↓
If CPU is allocated to a process, the process won't released the CPU till it gets completed.

→ The FCFS is a non-preemptive scheduling.

Ans:-
for previous question

Ghantt chart



waiting time
$WP_1 = 0$
$WP_2 = 27$
$WP_3 = 30$

Turn around time
$TP_1 = 27$
$TP_2 = 30$
$TP_3 = 33$

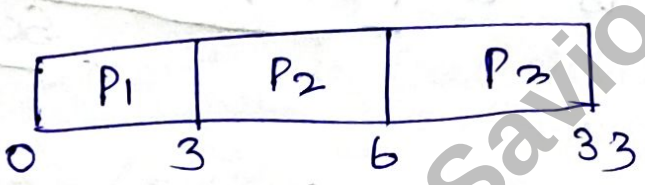
Average waiting time of a process $= \frac{(0+27+30)}{3}$
$= 19$ ms

Average turnaround time of a process
$= \frac{(27+30+33)}{3} = 30$ ms

⇒ FCFS

Process	CPU burst time (ms)	Arrival time
P_1	3	0
P_2	3	0
P_3	27	0

Ans: Gbrantt chart



Waiting time

$WP_1 = 0$

$WP_2 = 3$

$WP_3 = 6$

Turnaround time

$TP_1 = 3$

$TP_2 = 6$

$TP_3 = 33$

Average waiting time of a process

$= \frac{(0+3+6)}{3}$

$= 3$ ms

Average turnaround time of a process

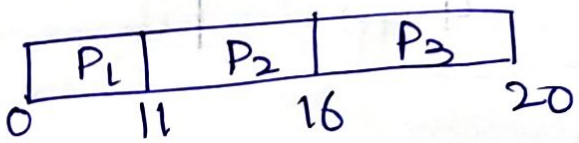
$= \frac{(3+6+33)}{3}$

$= 14$ ms

⇒ FCFS

process	CPU burst time (ms)	Arrival time
P_1	11	0
P_2	5	2
P_3	4	4

Ans:- Gantt chart



Waiting time.

$WP_1 = 0$

$WP_2 = 11 - 2 = 9$

$WP_3 = 16 - 4 = 12$

Turnaround time

$TP_1 = 0 + 11 = 11$

$TP_2 = 9 + 5 = 14$

$TP_3 = 12 + 4 = 16$

Average waiting time of a process

$= \frac{(0+9+12)}{3}$

$= 7$ ms

Average Turnaround time of a process

$= \frac{(11+14+16)}{3}$

$= 41/3$ ms

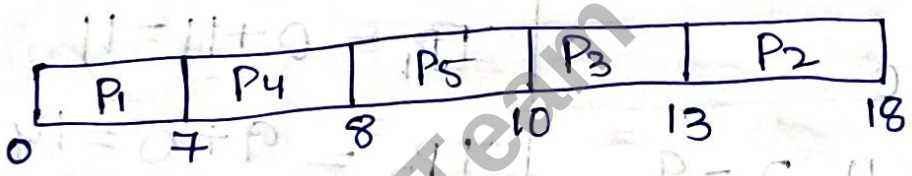
Shortest Job First (SJF) algorithm:-

→ SJF is non-preemptive scheduling.

(a)

Process	Arrival time	cpu burst time (seconds)
P_1	0	7
P_2	1	5
P_3	2	3
P_4	3	1
P_5	4	2

Ans: Gantt chart



$WP_1 = 0$

$WP_4 = 7 - 3 = 4$

$WP_5 = 8 - 4 = 4$

$WP_3 = 10 - 2 = 8$

$WP_2 = 13 - 1 = 12$

$TP_1 = 7$

$TP_4 = 5$

$TP_5 = 6$

$TP_3 = 11$

$TP_2 = 17$

→ Average waiting time $= \frac{(0+4+4+8+12)}{5}$

$= 28/5$ seconds

→ Average turnaround time $= \frac{(7+5+6+11+17)}{5}$

$= 46/5$ seconds

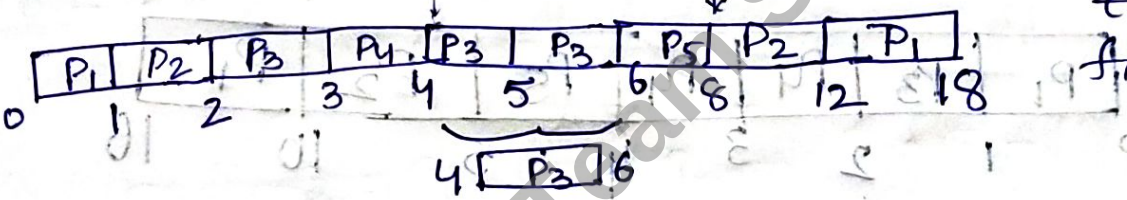
⇒ SRF (preemptive) algorithm :- [SRTF]

Q) Process	Arrival time	CPU burst time
✓ P_1	0	~~7~~ 8 0
✓ P_2	1	~~5~~ 4 0
✓ P_3	2	~~3~~ 2 1 0
✓ P_4	3	~~1~~ 0
✓ P_5	4	~~2~~ 0

Ans :-

Gnattchart

SRTF → shortest Remaining time first



→ If any conflict arises then use FCFS.

$WP_1 = 11$

$WP_2 = 6$

$WP_3 = 1$

$WP_4 = 0$

$WP_5 = 2$

Avg = 4 ms

waiting time = completion time − Execution time − Arrival time.

$TP_1 = 18$ $TP_2 = 11$ $TP_3 = 4$

$TP_4 = 1$ $TP_5 = 4$

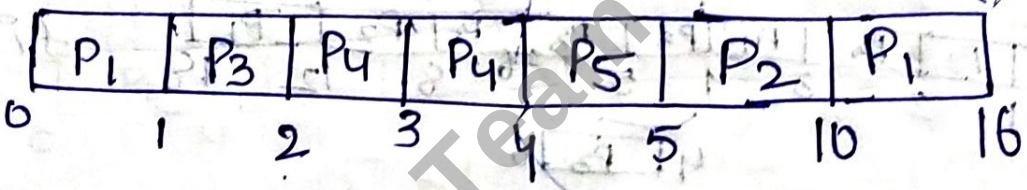
$Avg = \frac{38}{5}$ ms

Q) SRTF (preemptive)

Process	Arrival time	CPU burst time
✓ P_1	0	7 6
✓ P_2	3	5
✓ P_3	1	2 1 0
✓ P_4	2	4 3 0
✓ P_5	7	1 0

Ans

Gibatt chart



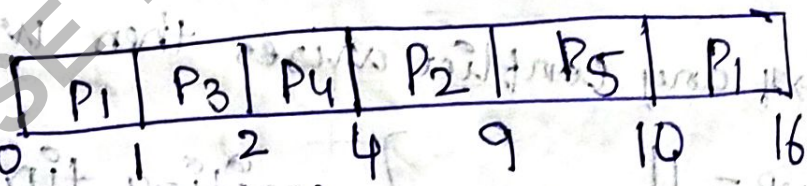
$WP_1 = 9$

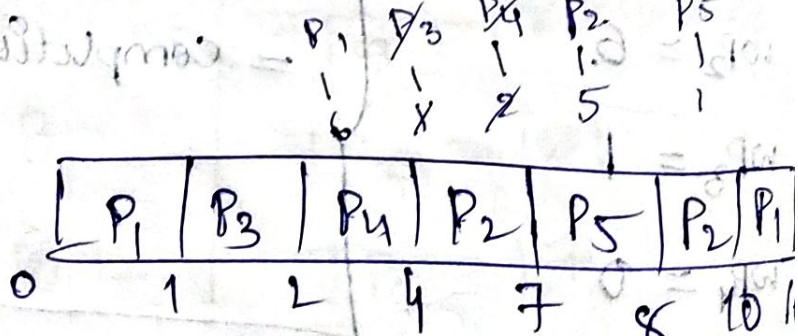
$WP_2 = 2$

$WP_3 = 0$

$WP_4 = 0$

$WP_5 = 0$





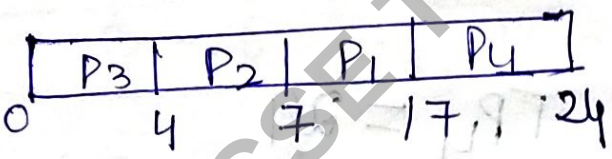
Priority scheduling :- non (preemptive)

UNIX :- (low number — high priority)

Process	Priority	CPU burst time
P_1	3	10
P_2	2	3
P_3	1	4
P_4	4	7

Ans:

Gantt ~~Gantt~~ chart



$WP_3 = 0$

$WP_2 = 4$

$WP_1 = 7$

$WP_4 = 17$

$Avg = 28/4$ ms

$= 7$ ms

$TP_3 = 4$

$TP_2 = 7$

$TP_1 = 17$

$TP_4 = 24$

$Avg = 13$ ms

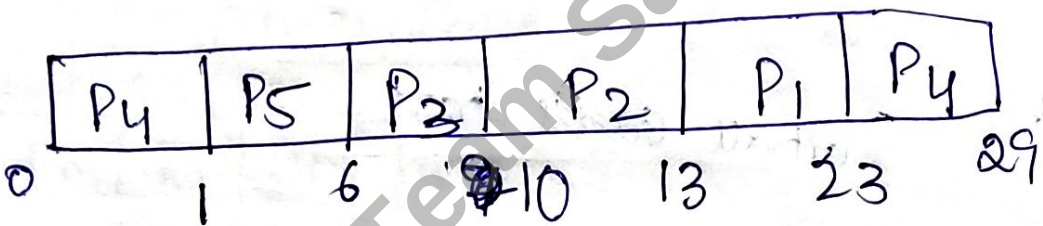
⇒

Process	Priority	Cpu burst time	Arrival time
P_1	3	10	1
P_2	2	3	2
P_3	1	4	4
P_4	4	6	0
P_5	0	5	1

→ preemptive

P_4 P_5 P_3 P_2 P_1 P_4

Ans: Ghatt chart



$WP_4 = 22$

$WP_5 = 0$

$WP_3 = 2$

$WP_2 = 8$

$WP_1 = 12$

$Avg = \frac{44}{5}$ ms

$TP_4 = 29$

$TP_5 = 5$

$TP_3 = 6$

$TP_2 = 11$

$TP_1 = 22$

$AVG = \frac{73}{5}$ ms

Round Robin Scheduling (time slice or time quantiey)

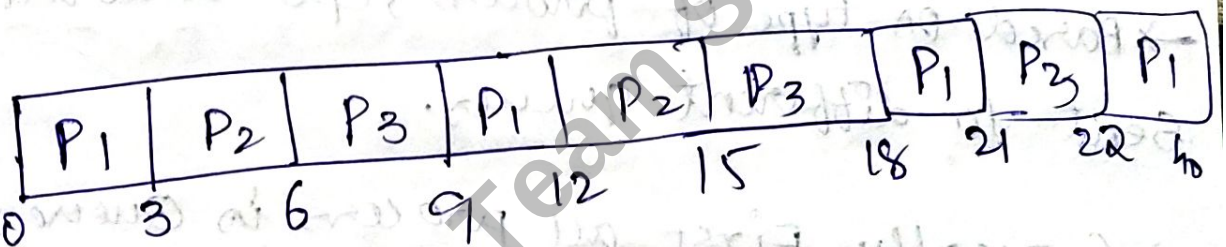
(preemptive)

⇒

process	cpu burst time (time slice = 3ms)
P_1	27 24 21 18
P_2	6 3 0
P_3	7 4 1

Ans:

Ghatt chart



$WP_1 = 13$

$WP_2 = 9$

$WP_3 = 15$

$AVG = \frac{37}{3}$ ms

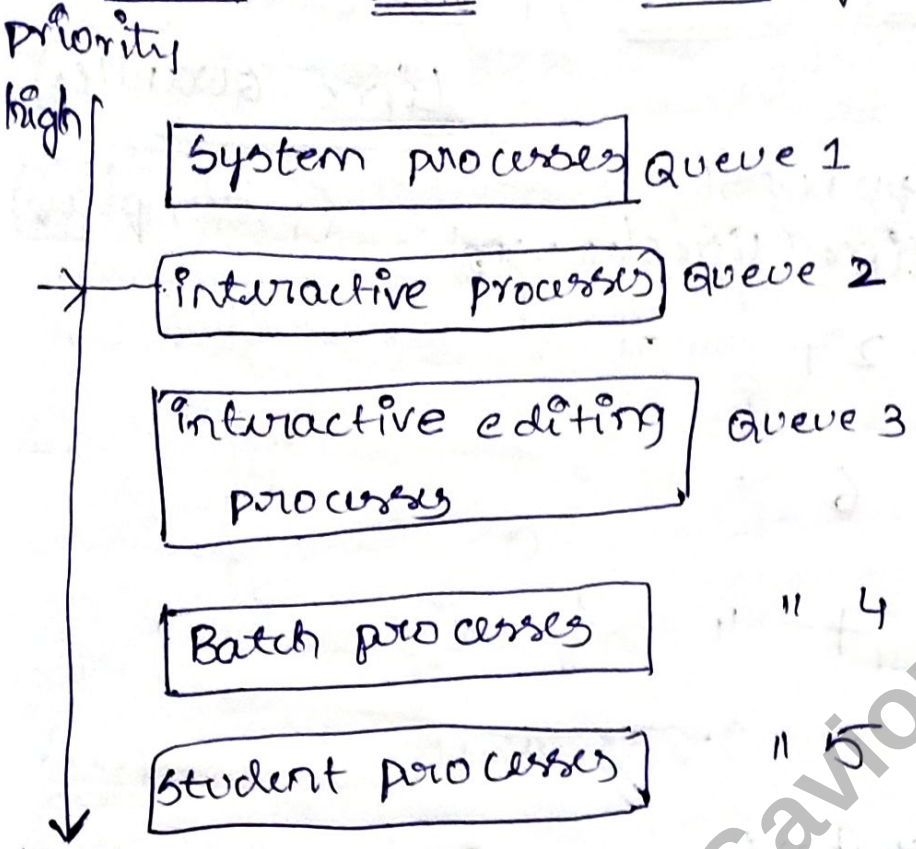
$TP_1 = 40$

$TP_2 = 15$

$TP_3 = 22$

$AVG = \frac{77}{3}$ ms

Multilevel Queue scheduling :-



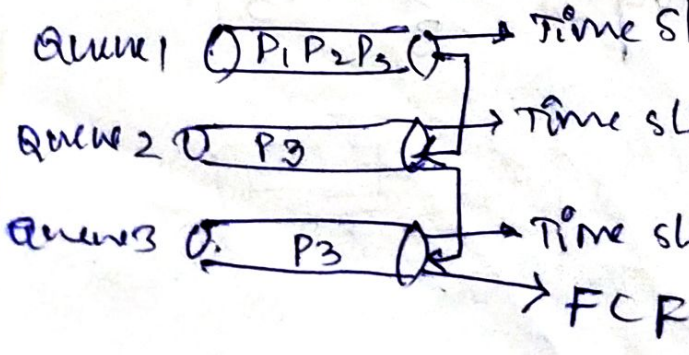
→ Based on type of process, cpu scheduling is sent to different queues.

→ Generally, First all process in Queue 1 gets finished then it enters into Queue 2.

→ Sometimes process in Queue 1 take long cpu time. To overcome this time slicing is done. (Round robin fashion)

(exclusive pre-emptive)

Eg:- MQFS



→ If P3 is not completed in Queue 1, then it moves to Queue 2 and it is Executed.

→ At last level Queue n → The algorithm is used is FCFS.

UNIT-II

Process synchronization

→ A transaction in DBMS is same as process in OS.

→ when there are multiple process simultaneously getting executed and sharing the common data there may arise data inconsistency problem.

→ TO avoid this the processes (dependent processes) communicating are to be synchronization.

→ The process of schronizing the ~~data~~ Process is called process synchronisation.

next produced int buffer[size] next consumed

P_1 count = 0
in = 0
out = 0

P_2

Producer

```
while(count == buffersize);
      do nothing;
buffer[in] = nextproduced
in = (in+1) % buffersize
count++;
```

consumer

```
while (count == 0)
      do nothing
nextconsumed
      = buffer[out]
out = (out+1) % buffer size
count--;
```

count = 5

Time

$T_0 \rightarrow$ register 1 = count (5)

$T_1 \rightarrow$ register 1 = register 1 + 1 (6)

$T_2 \rightarrow$ register 2 = count (5)

$T_3 \rightarrow$ register 2 = register 2 − 1 (4)

$T_4 \rightarrow$ count = register 1 $\longrightarrow$ 6

$T_5 \rightarrow$ count = register 2 $\rightarrow$ 4

Producer

count ++

```
register1 = count
register1 = register1 + 1
count = register1
```

Consumer

count − −

```
register2 = count
register2 = register2 − 1
count = register2
```


Solutions for Process synchronization

- Software solution (peterson's solution 1981)
- Hardware synchronization
  - test and set()
  - swap()
- Semaphores (OS supports)
- Monitors (undefined)

→ When concurrent processes shares a common data, the outcome depends upon the order of Execution of process. It is called as race condition.

→ Critical Section:-

→ It is a portion of process source code used to access the shared data Between the processes.

Critical section Problem:-

→ when one process is in its critical section no other process is allowed to enter into its critical section.

Solution to Critical section problem:-

- Mutual Exclusion
- Progress
- Bounded waiting

Bounded waiting:-

→ There Exists a finite (or) Bounded count before a process can Enter into its critical section when other processes are waiting to Enter into their Critical section.

Software Solution :) (Peterson solution)

→ Let two concurrent process P_i and P_j

→ Let an array flag[2] = {false, false}

turn can be P_i (or) P_j

P_i → This solution Works only 2 Process

```
do {
    flag[i] = true;
    turn = j;
while (flag[j] == true && turn == j)
    Enter into Critical section
    flag[i] = false;
}while(true);
```

P_j

```
do {
    flag[j] = true;
    turn = i;
while (flag[i] == true && turn == j);
    Enter into critical section
    flag[j] = false;
}while(true);
```

General section of critical section:-

```
do{
    entrysection  → used to check if any one in CS, if exist wait, otherwise Enter
    Critical section
    exitsection   → notify others that you came out of CS
    remainder section;
}while(true);
```

P0

```
do{
    flag[i]=true;
    turn=j;
while(flag[j]==true && turn==j);
    i(i=do nothing;
    count++;   S1;
               S2;
    flag[i]=false;
}while(true);
```

Pj

```
do {
    flag[i] = true;
    turn = i;
while (flag[i] == true && turn == i);
    do nothing;

    count--
    CS
    S1;
    S2;

    flag[j] = false;
} while (true);
```

Hardware Synchronization (Processor):-

- Uniprocessor (-disable interrupts)
- Multiprocessor

Test & Set Methods

```
do {
while (Test and Set (& LOCK));
    // do nothing
    ≡ CS
    LOCK = false
} while (true);
```

```
while (Test and Set(&lock)) ;
```

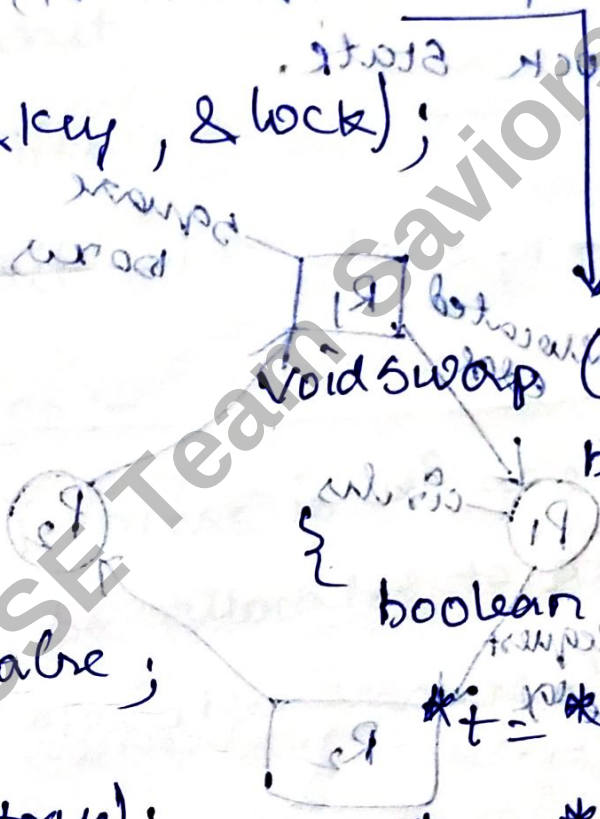
```
boolean Testandset (*target) {
    boolean *temp;
    temp = *target;
    *target = true;
    return temp;
}
```

Swap :-

Pi

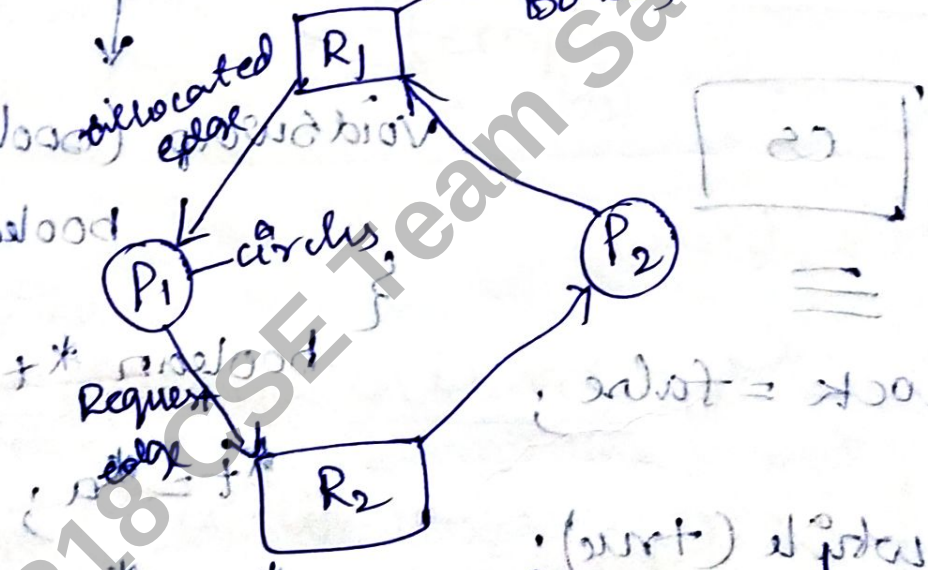
```
lock = false
key = true
do {
while (key == true)
{
swap(&key, &lock);
}
CS
lock = false;
} while (true);
```

```
void swap (boolean *a,
           boolean *b)
{
boolean *t;
*t = *a;
*a = *b;
*b = *t;
}
```



Deadlocks :-

→ When there are set of processes Executing concurrently competing for the resources such that a process P_i holding the Resource R_i already and Requesting a resource R_j which is held by the process P_j and process P_j requiest resource R_i then Process P_i, P_j may be in the Deadlock state.



Resource allocation graph

NOTE :-

→ If resource allocation graph contains a cycle and if the resources are single instance resource then there is a Deadlock

→ If the resources is multiple instance resource then there may be deadlock.

use of Resource :-

① request
② use
③ Release

Necessary conditions for occurance of deadlock

1) Mutual Exclusion
2) hold and wait
3) no preemption
4) Circular wait

1) Mutual Exclusion :-

→ when one process is using a resource it should not be allocated to other process (non sharable resources).

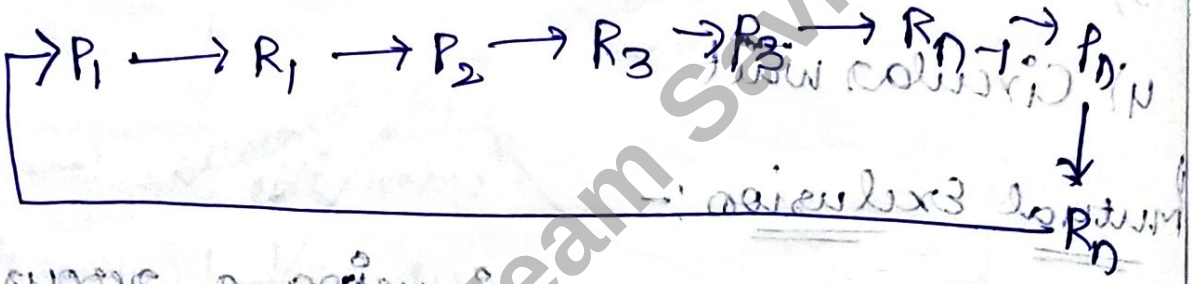
2) hold and wait :-

→ one process is holding a resource and waiting for a resource which is held by other process.

3) No preemption:-

→ If a process is using a resource it won't release the resource till its completion. (revocation of a resource is not possible, it should be done voluntarily by process)

4) Circular wait:-



Deadlock

- Deadlock prevention
- Deadlock Avoidance
- Deadlock detection and Recovery.

Deadlock Avoidance → Bankers Algorithm
- Resource request algorithm
- Safe state detection Algorithm

i) Deadlock prevention:-

→ prevent the occurance of any one of necessary condition.

→ i) let the shareable resources be shared.

→ ii) A process is not allowed to request a resource if it is already holding a resource.

(or)

→ If a process request a resource first let the process release the resource it is holding.

iii) If a process request a resource, if it is not given immediately then the process is supposed to voluntarily release the resources held by it.

iv)

$$\boxed{\begin{array}{l} f(\text{tape drive}) = 1 \\ f(\text{disk drives}) = 5 \\ f(\text{printer}) = 12 \end{array}} \longrightarrow F: R \to N$$

$$f(R_j) > f(R_i)$$

i.e first lesser resources needed process should be allocated.

Deadlock Avoidance :- (Bankers Algorithm)

Safe state :-

→ The system is said to be in safe state if there exists a safe state sequence of processes. Such that the system can distributed the resources and the process in the system can finish their execution.

→ If system is in safe state then deadlock does not occur. If it is in unsafe state then there may be a chance of deadlock.

- Resource request algorithm (P_i)
- Safe state detection

$A = 10$ $B = 5$ $C = 7$

Problem	Allocation			Available			MAX i			Need P_i		
	A	B	C	A	B	C						
P_0	0	1	0	3	3	2	7	5	3	7	4	3
P_1	2	0	0				3	2	2	1	2	2
P_2	3	0	2				9	0	2	6	0	0
P_3	2	1	1				2	2	2	0	1	1
P_4	0	0	2				4	3	3	4	3	1

DATA structure used in Banker's algorithm :-

1) Available [j] = k of length M (no of resources)

2) Request [i] [j] = k [no instances of Resource type R_j]
   ↓ ↓
   P_i R_j

3) Max [i] [j] = k
4) Need [i] [j] = k } Allocation [i] [j] = k

Banker's algorithm for resource request (pseudo)

1) Request i < Need i [yes goto step 2 else request cannot be granted process goes to waiting state]

if

2) Request i < Available

Allocation$_i$ = Allocation$_i$ + Request$_i$

Need$_i$ = Max$_i$ − Allocation$_i$

Available$_i$ = Available$_i$ − Request$_i$

↓

else

process is in waiting state

Safe state detection algorithm:-

1) finish[i] = false for all i

Available = k

2) find a process such that

if finish[i] = false, Request <= Available

else

goto step 4

3) Available = available + allocation i

finish[i] = true goto step 2

4) finish[i] = true for all i then

System is in safe state

finish[i] is not true for all i then

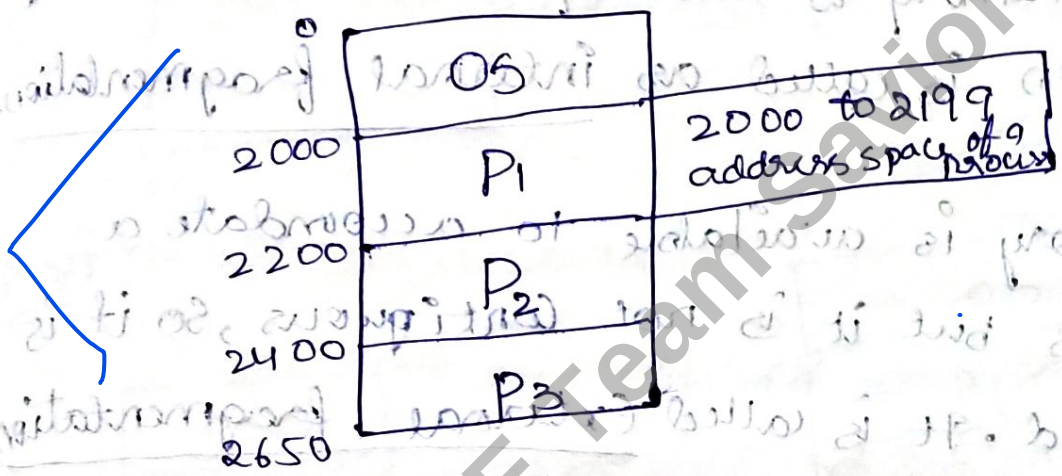
System is in unsafe state.

If P_2 process is unsafe state then the process P_2 will be in waiting state and protection will be go to process P_3

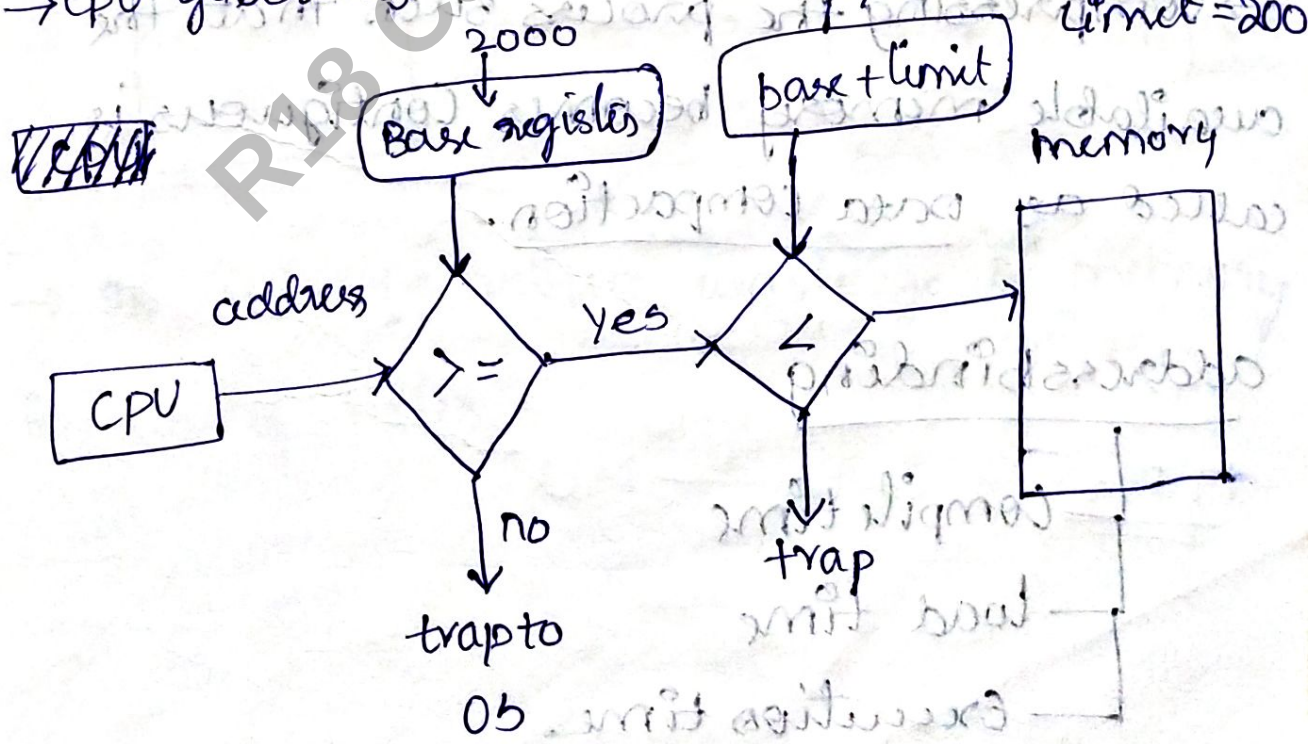
UNIT-III

Memory Management

- cache memory Management
- Primary memory Management
- Virtual memory Management
- Secondary memory Management



→ cpu generates the address



Hardware protection to memory address

Primary Memory Management:-

- Multiple partition technique (MFT) → fixed size partitions
- Multiple variable partition Technique (MVT)
  - Paging
  - Segmentation
  - Paging with Segmentation

→ If memory is wasted/unused within the partition is called as internal fragmentation.

→ Memory is available to accomodate a process but it is not contiguous, so it is unused. It is called External fragmentation.

→ Compressing the process such that the available memory becomes contiguous is called as Data compaction.

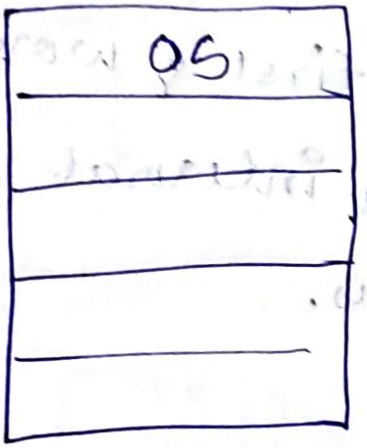
addressbinding

- Compile time
- load time
- Execution time

data loading

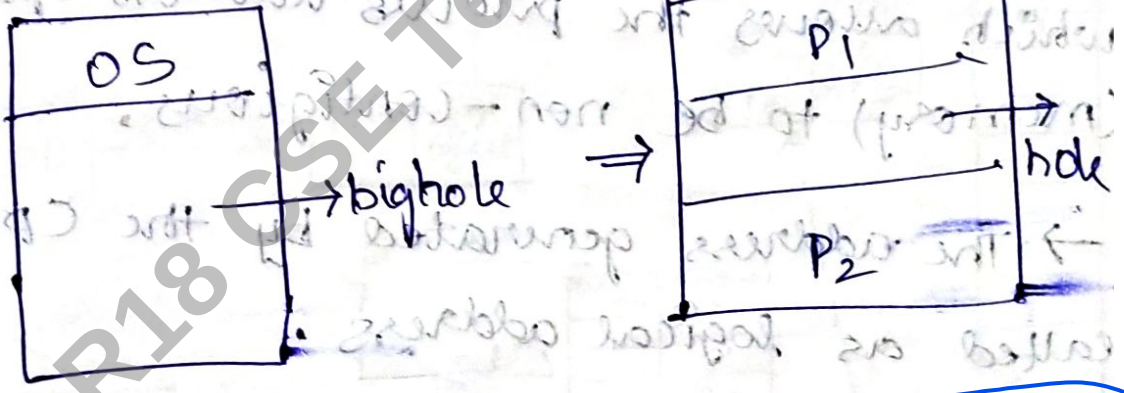
- load time loading (static)
- Dynamic loading

MFT :-



→ fixed partition

MVT :-



→ In this technique wasteage of memory is no of holes.

Holes Management :-

- first fit
- Best fit
- Worst fit

→ Apart from Best fit, first & worst fit suffer from both internal and external fragmentation.

Paging :-

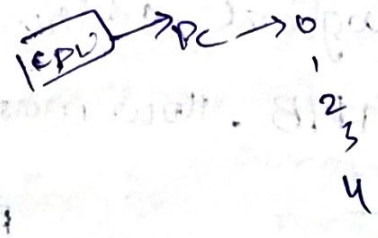
→ It is memory management technique which allows the process address space (memory) to be non-contigious.

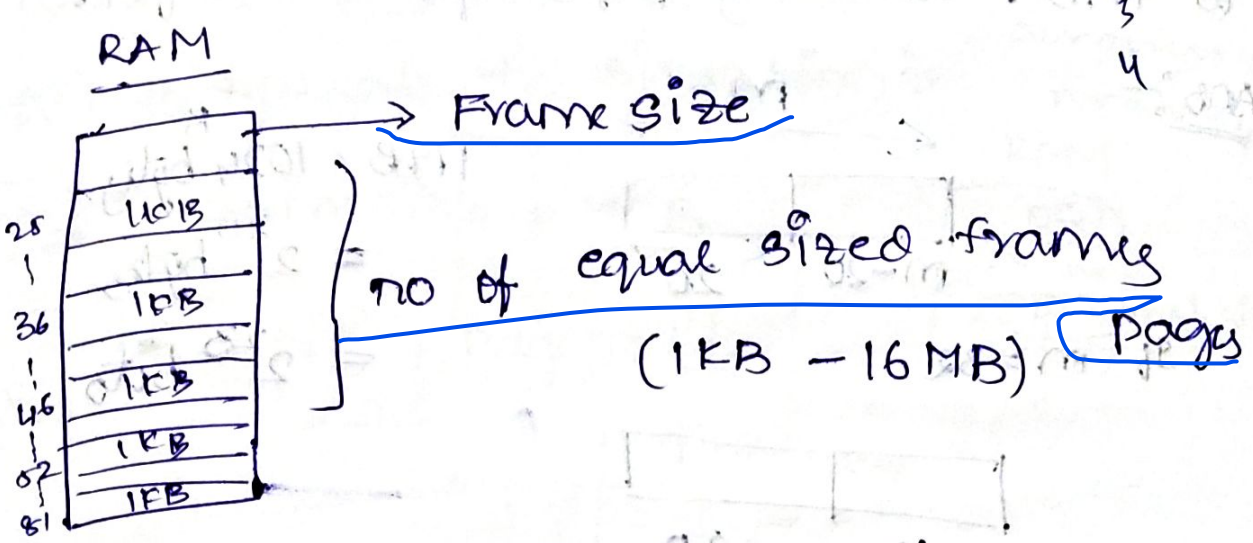
→ The address generated by the CPU is called as logical address.

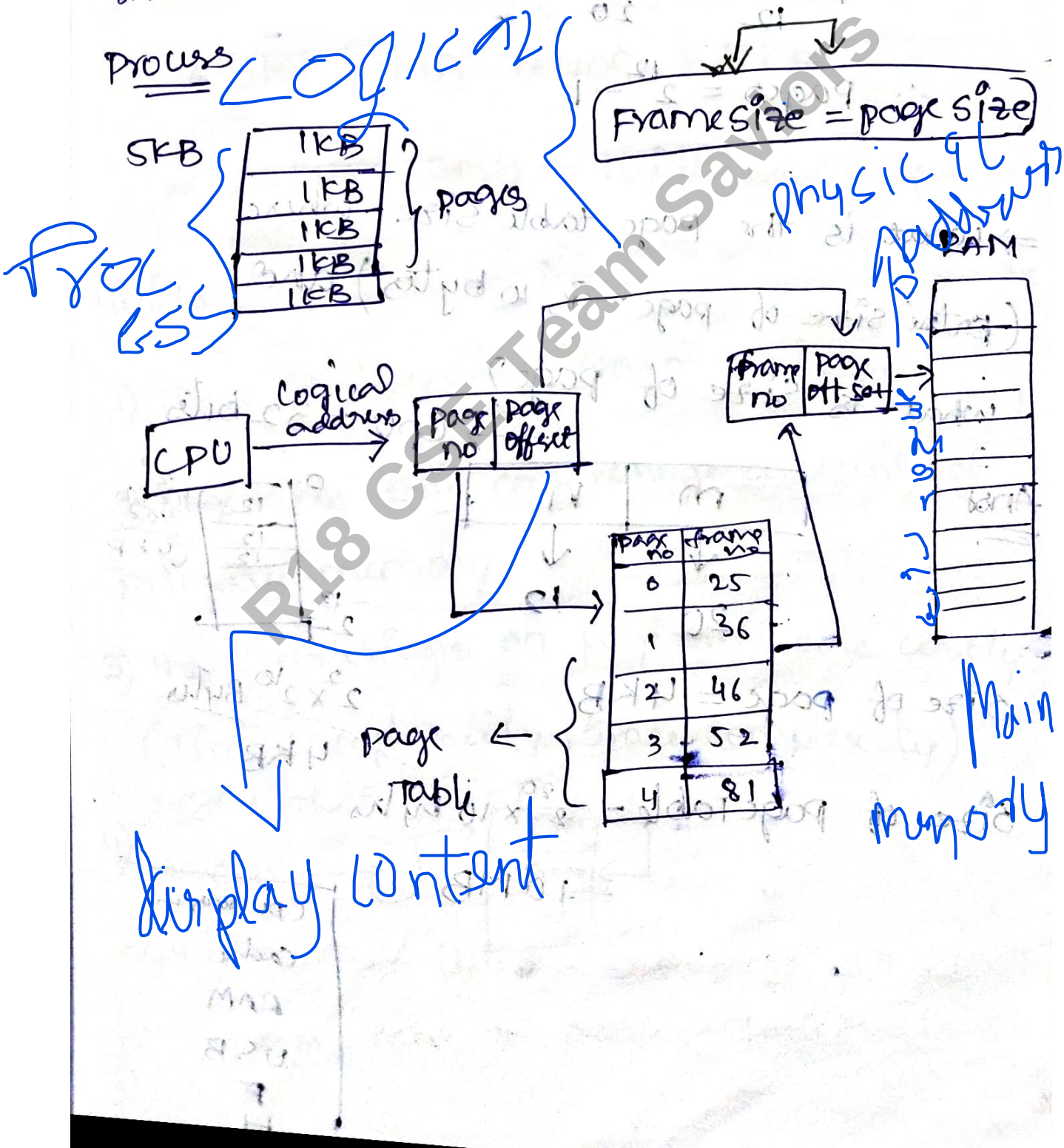
→ The address seen by the memory unit is called as physical address.

MMU :- CPU — MMU — MU

paging table

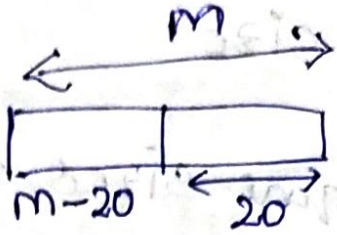






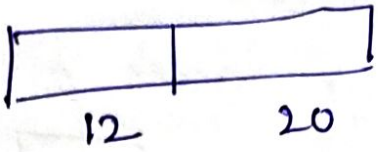
→ logical Address m-bits, size of the page is 1MB. How many pages cpu can refer.

Ans:-



1MB = 1024 bytes
= 2^{20} bytes
= 2^{23} bits

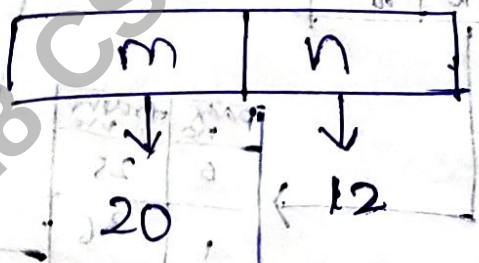
If m = 32

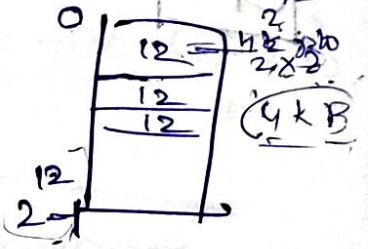


∴ Pages = $2^{12} - 1$

⇒ What is the page table size where (entry size of page is 10 bytes) and what is size of page? → CPU = 32 bits

Ans





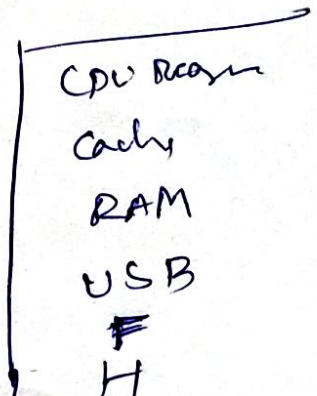
Size of page = 4KB

$2^2 \times 2^{10}$ Bytes

4KB

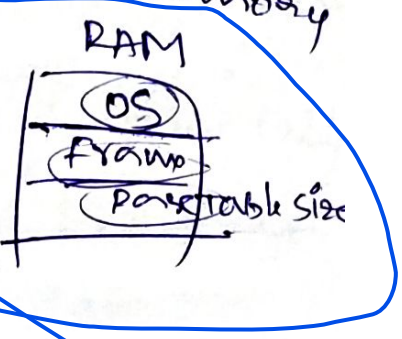
Size of PageTable = $2^{20} \times 10$ bytes

= 10MB



continue before questions

⇒ OS size is 512 MB if there are 4096 Frames. what would be the Max size of RAM to operate the situation? amount of memory



Ans:- Frame size = 4KB

RAM = 512MB + Page Table + Frames

= 512MB + 10MB + 4096 × 4KB

= 512MB + 10MB + 16MB

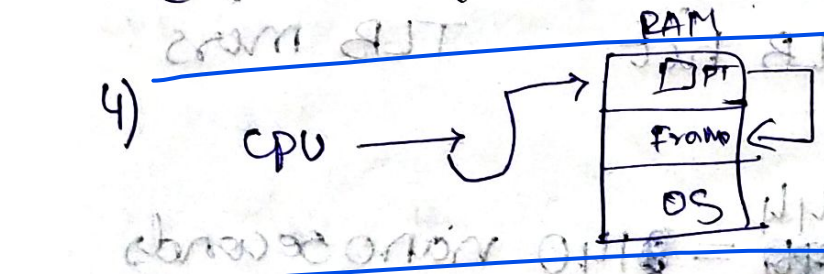
Limitations with paging:-

1) There is internal fragmentation

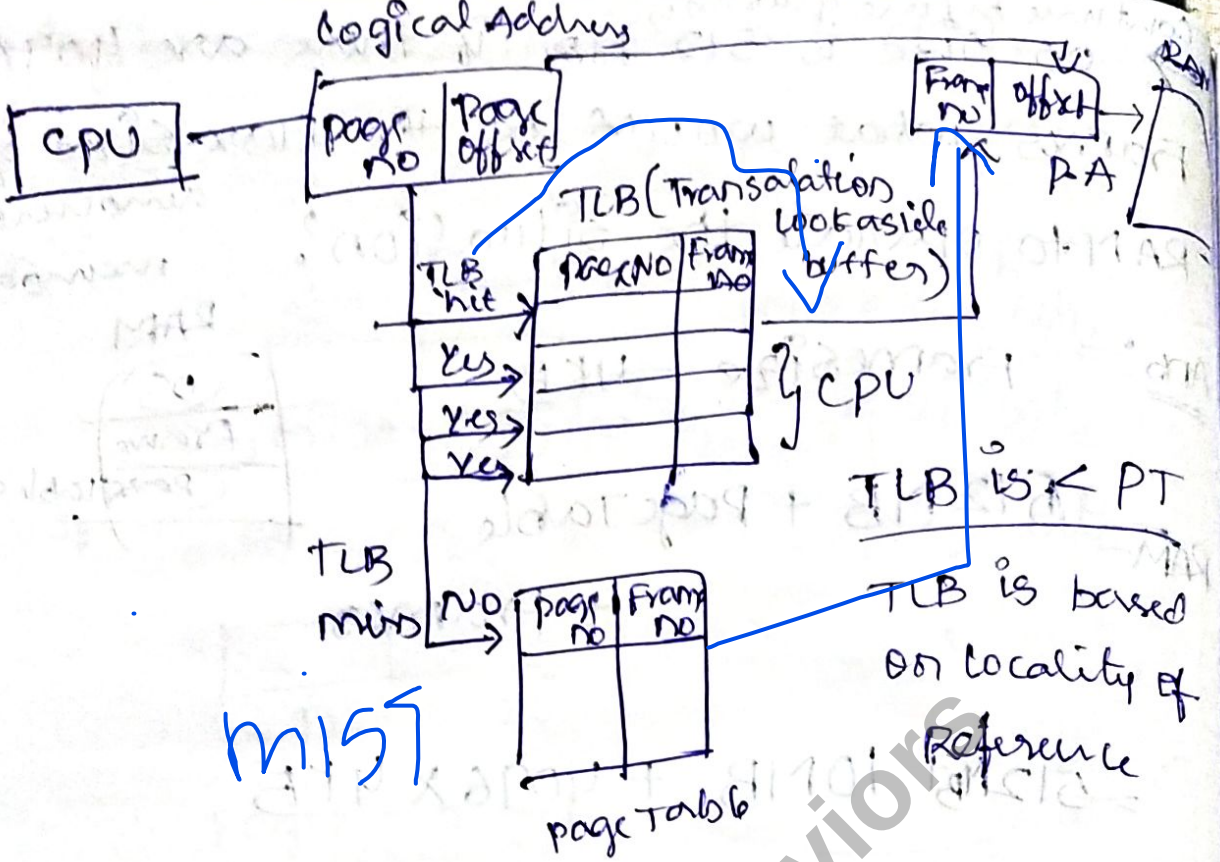
2) Page table may consume huge amount of primary memory.

3) The operations on page table are costly. (Time complexity, Space complexity)

4)



RAM need to access twice.



TLB is < PT

TLB is based on locality of Reference

→ page no which are frequently used are Stored in TLB.

Problem :-

1) 80% probability for TLB hit
TLB access time is 20 nanoseconds
Main memory access time is 100.

Ans. EAT = $0.8 \times (20 + 100) + 0.2 \times (20 + 100 + 100)$

TLB hit TLB miss

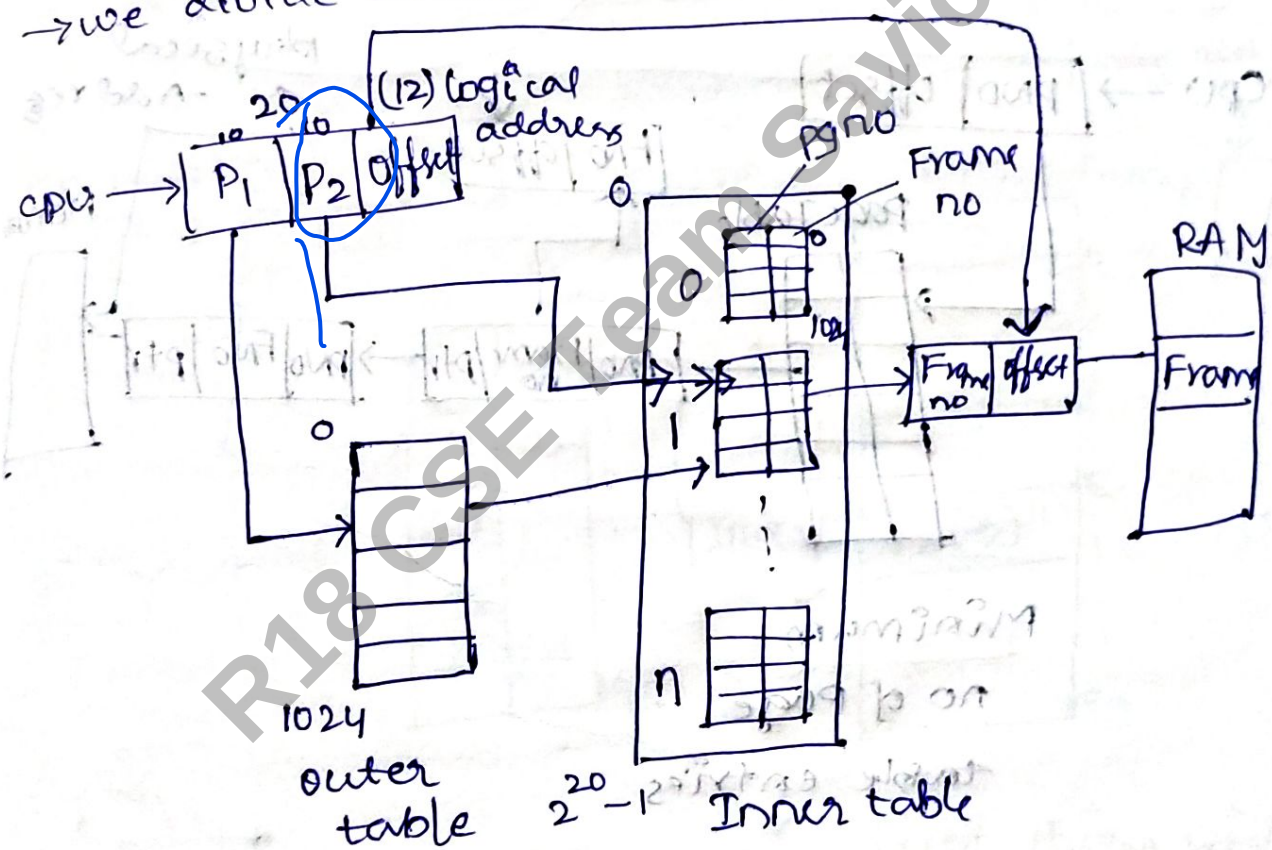
EAT = 96 + 44 = 140 nanoseconds

Page - tables :-

1) Multi-level page table } page table for each process
2) Hash-based page table
3) inverted page table → single page table ∀ processes

1) Multi-level page table :-

→ we divide all the possibilities



→ outer table gives info about which table i.e., 1 or 2.

→ P_2 gives the line number in that table from there we will get frame no.

2) Hash based page Table :-

a[7] = {0, 1, 2, 3, 4, 5, 6}

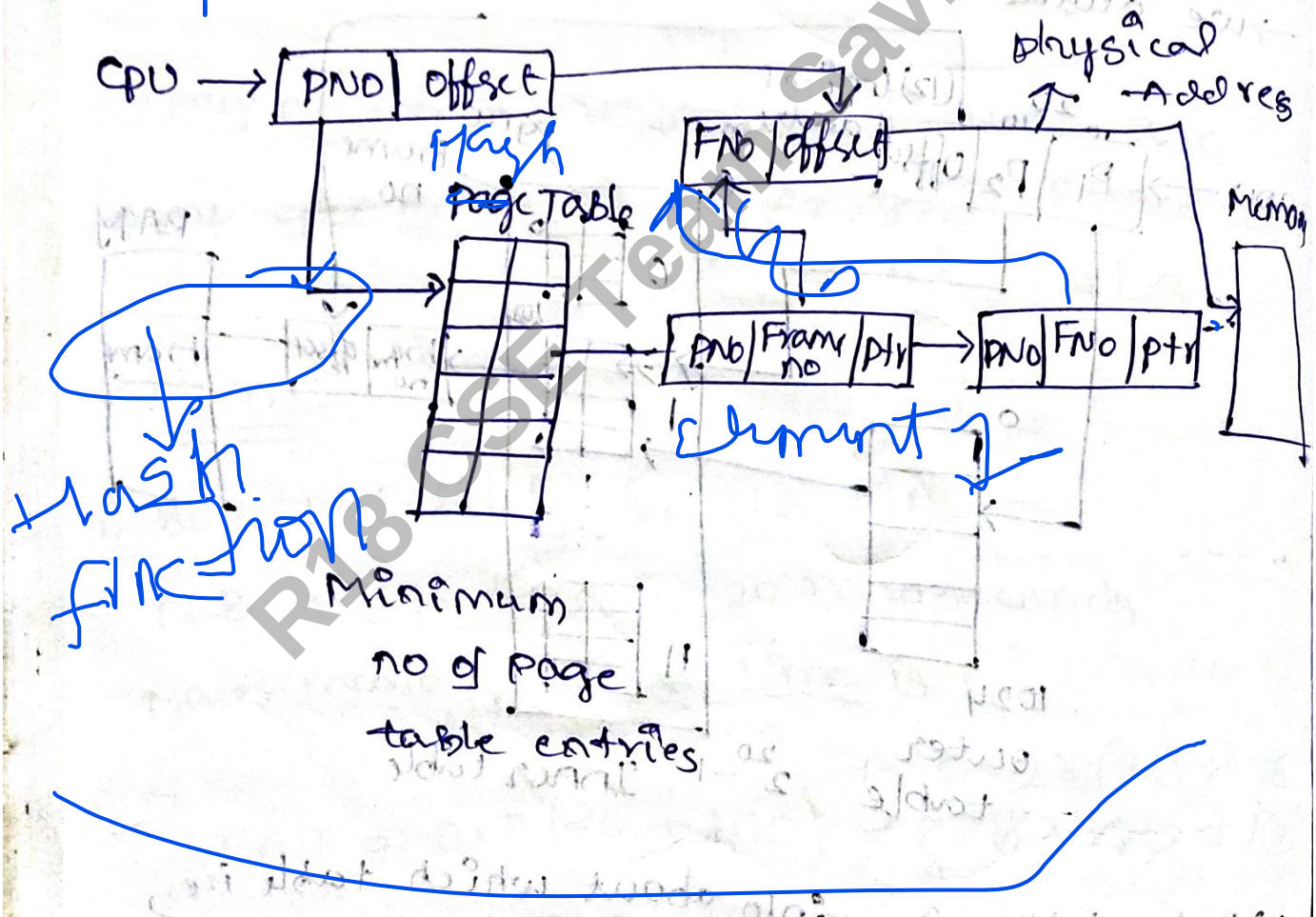
Linear Search

$T(n) = O(n)$ (W.C)

$T(n) = \Omega(1)$ (B.C)

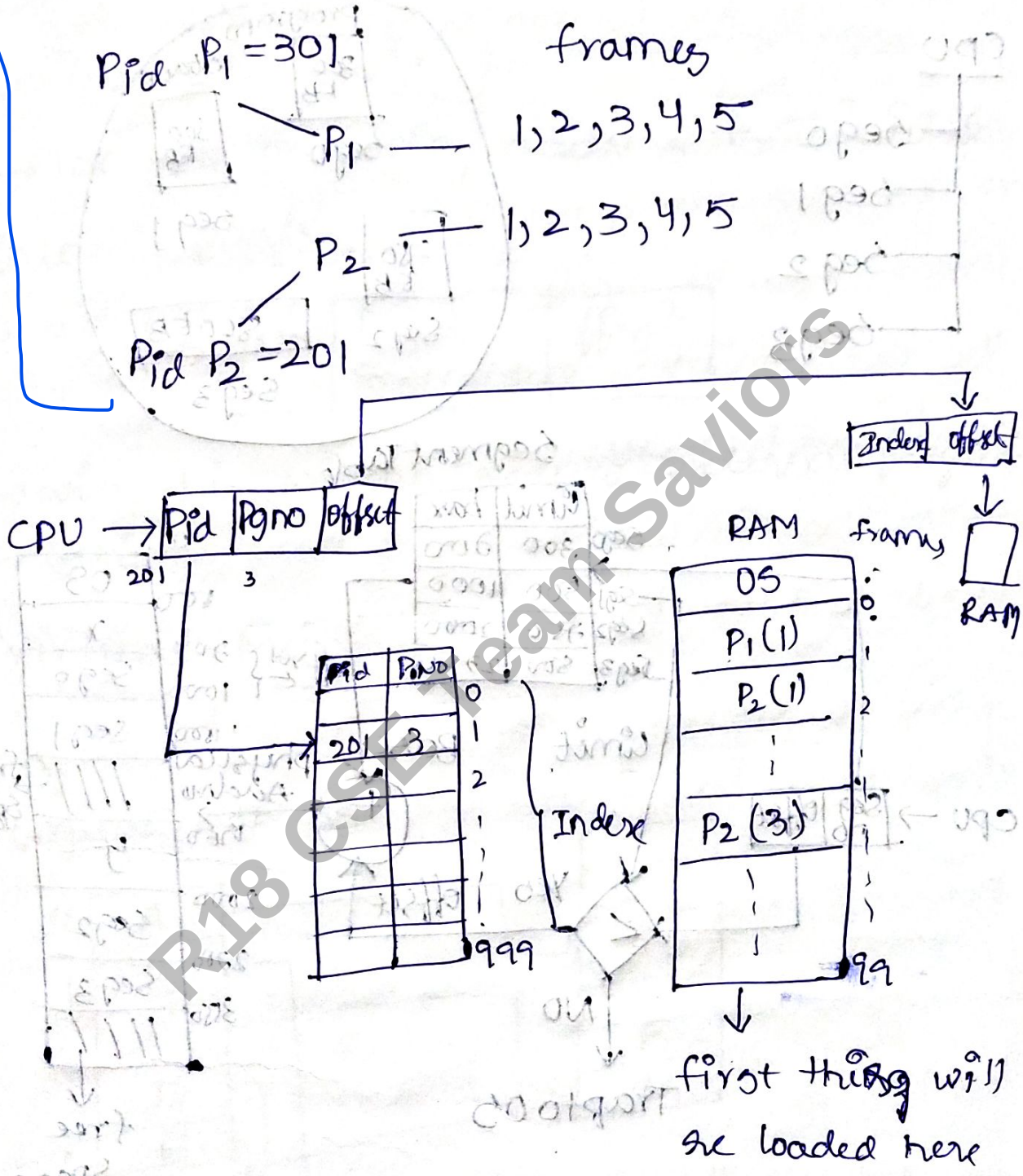
$T(n) = \Theta(n/2)$ (A.C)

$\rightarrow O(1)$



3) Inverted page Table :-

→ one pageTable is maintain for all the process.

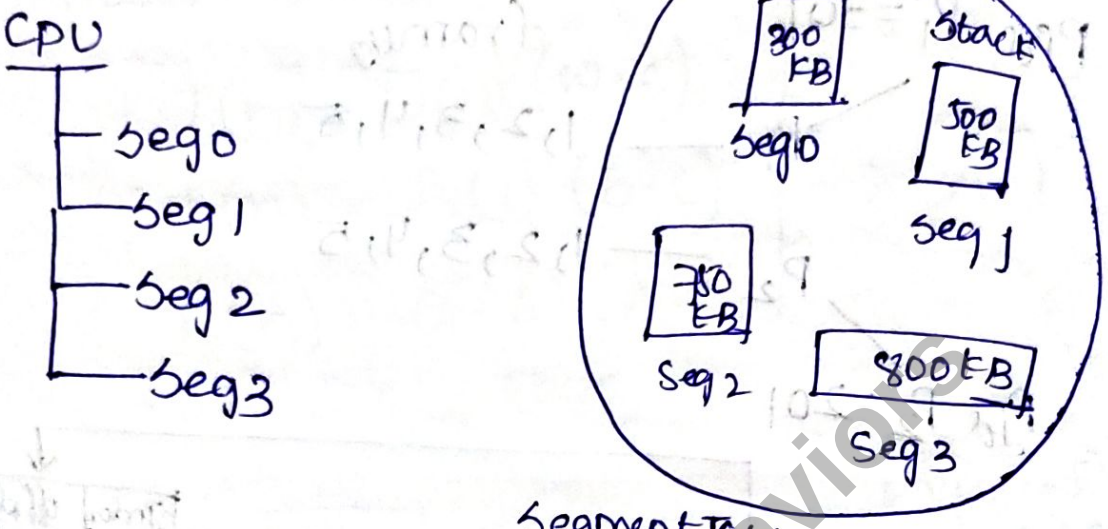


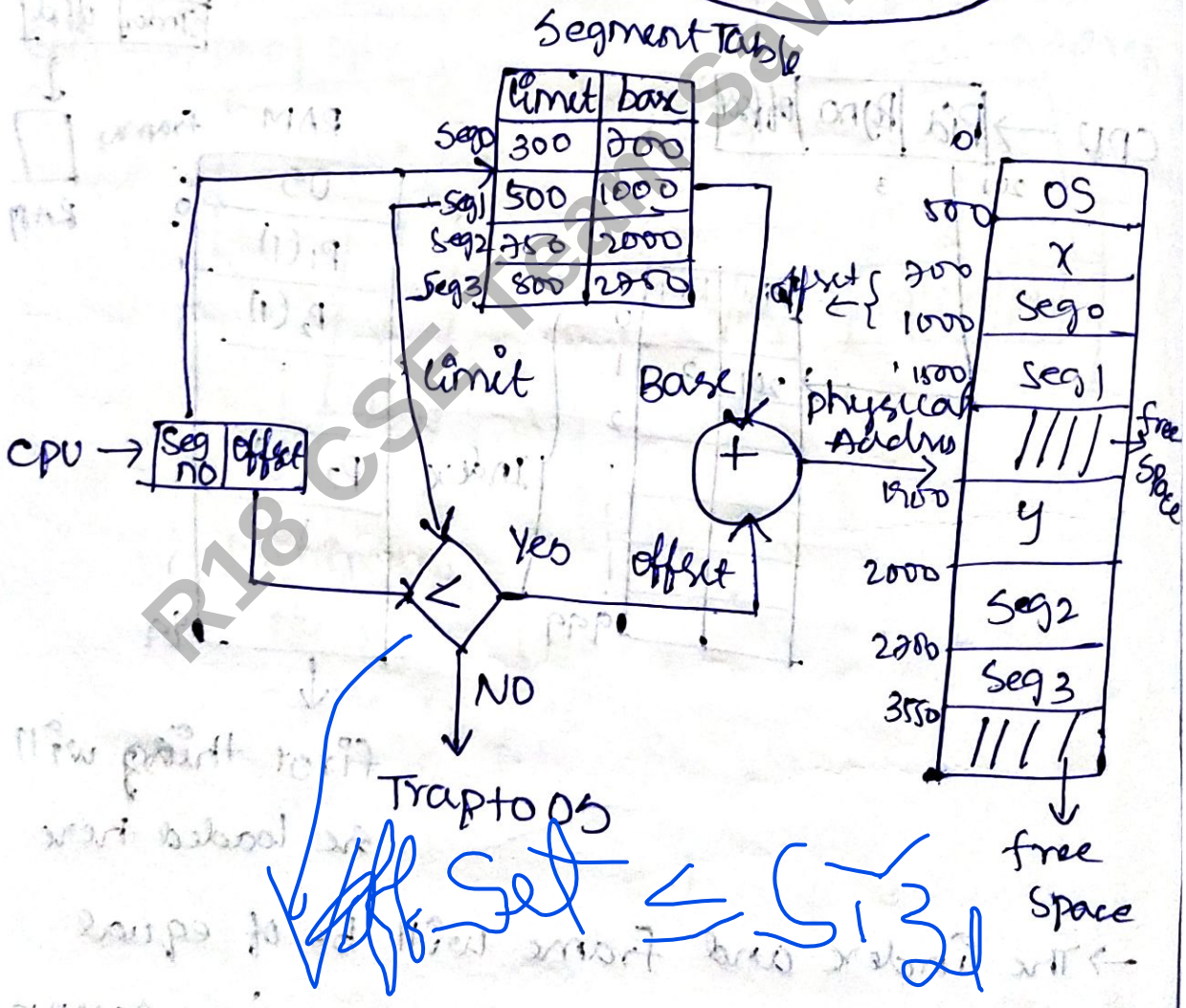
→ The index and frame will be of equal number and the index of the requires pageNo will be the same for the frame number.

Index = frame no

Segmentation :-

→ It allows process address space to be non-contiguous.





offset < S1, S2

Paging with Segmentation:-

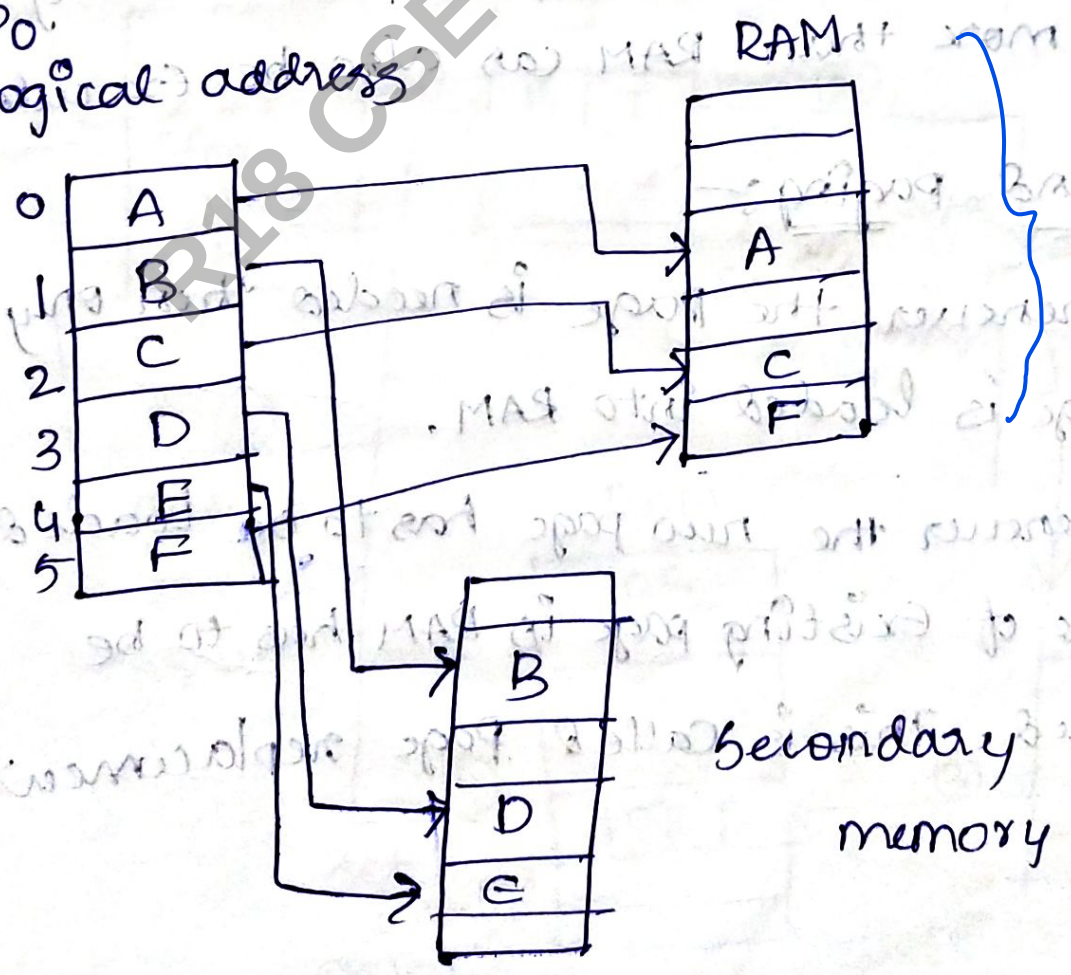
→ In segmentation, we divide memory into segment. now for each segment we will have a page table.

→ The CPU producing address is divided into 3 parts

SegNO	Page no segment	offset

→ with this we can reduce external fragmentation.

Virtual Memory



→ In this technique, only few pages of the process are stored in the RAM and the rest are stored in secondary memory.

→ whenever the process is getting executed they will be added to RAM] → Demand Paging

→ The CPU can generate range of addresses larger than the primary memory addresses, the extra memory refferred CPU is called as Virtual memory.

→ The particular section of hard disk is used to store the entire pages of a process.

→ Due to virtual memory a process having size more than RAM can also be executed.

Demand paging :-

→ whenever the page is needed then only the page is loaded into RAM.

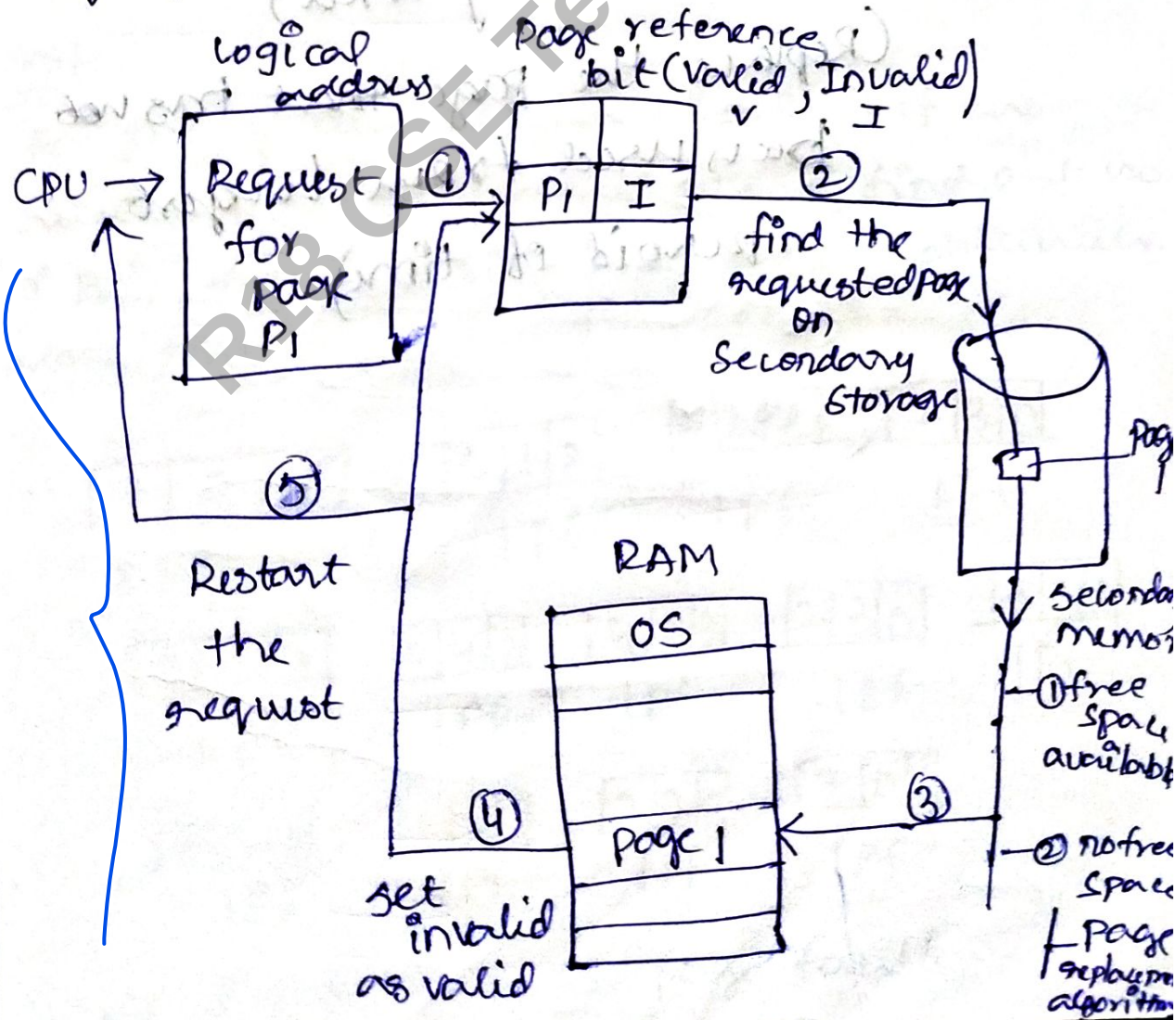
→ whenever the new page has to be loaded, the one of existing page in RAM has to be removed. This is called page replacement.

Page fault :-

→ If the Requested page is not available in the physical/primary/RAM memory then it is called page fault.

pure Demand Paging :-

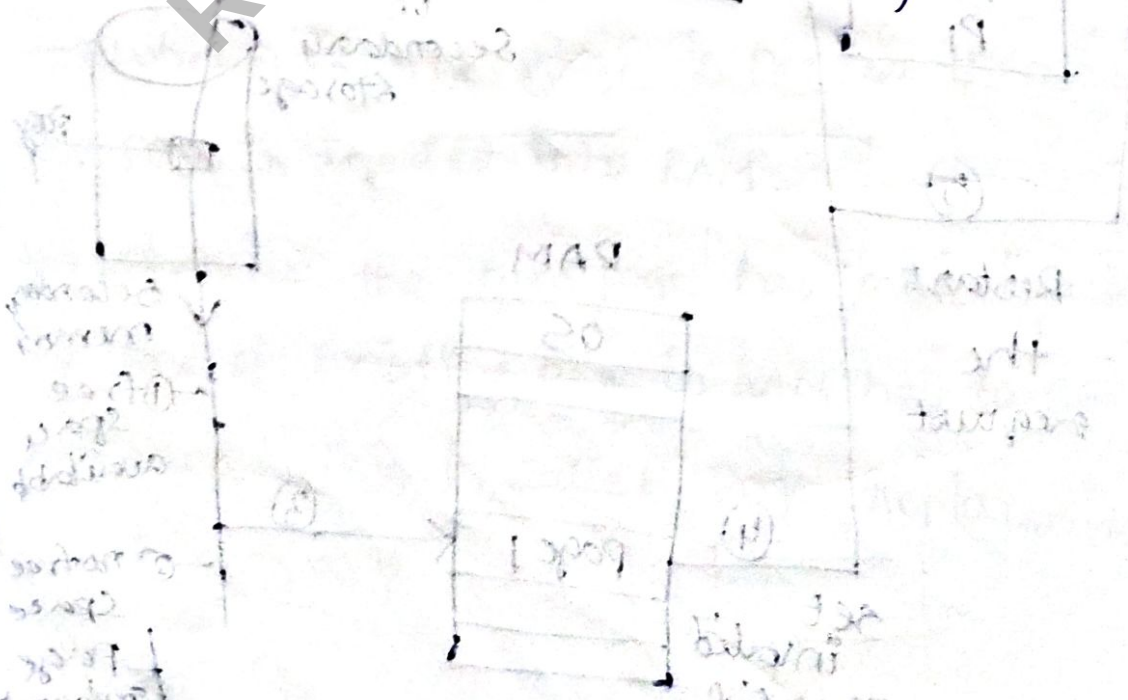
→ If we want to Execute a process the first request would be a page fault because no pages are loaded. when the execution of process starts with page fault and gets completed is called pure demand paging.



Page replacement algorithms:

Issues that arise during Virtual memory (In multiprocess systems):-

1. Frame allocation
2. Page replacement
   - FIFO → Replace oldest page
   - Optimal page replacement (replace the page that will not be used for longest period of time)
   - LRU (least recently used) (Replace the page that has not been used from longest period of time)



Reference String:-

→ The order of requests for the pages to execute a process is represented in the form of a memory string is called reference string of a process.

Example:-

FIFO:-

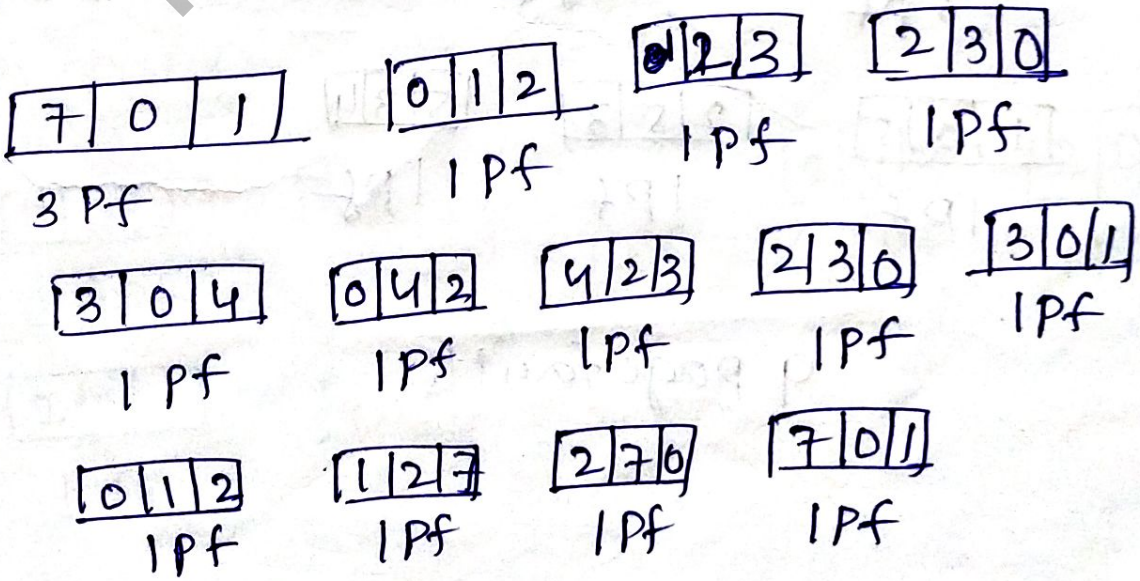
Reference String:-

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

p → Initially there are 3 free frames.

Ans:-

For the given reference string, if the available free frames are 3, find out no of page faults using FIFO page replacement algorithms.



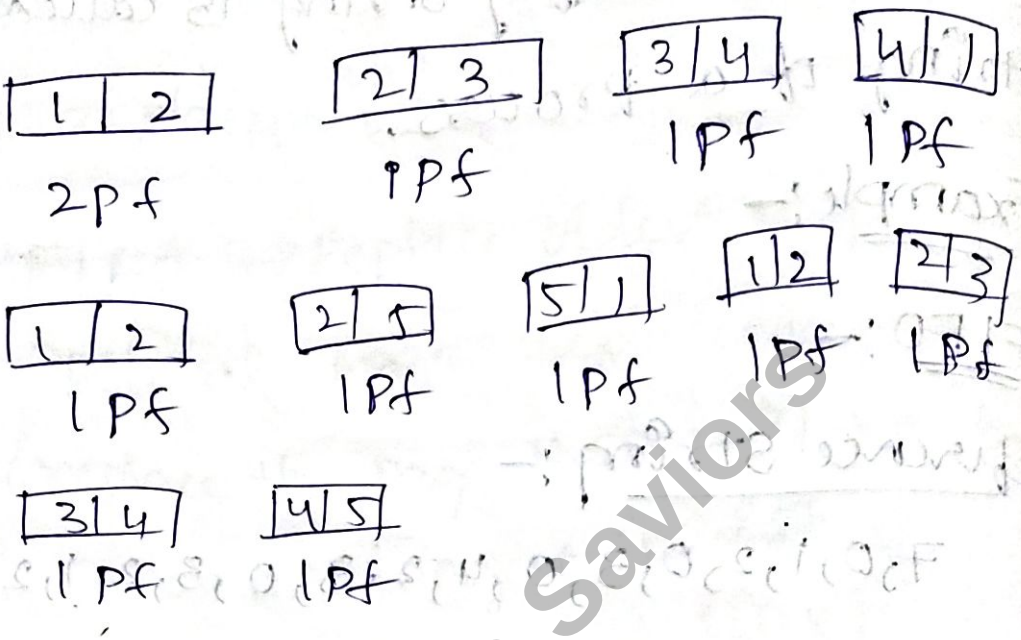
∴ There are 15 page faults.

⇒ FIFO

RS = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

free frames = 2

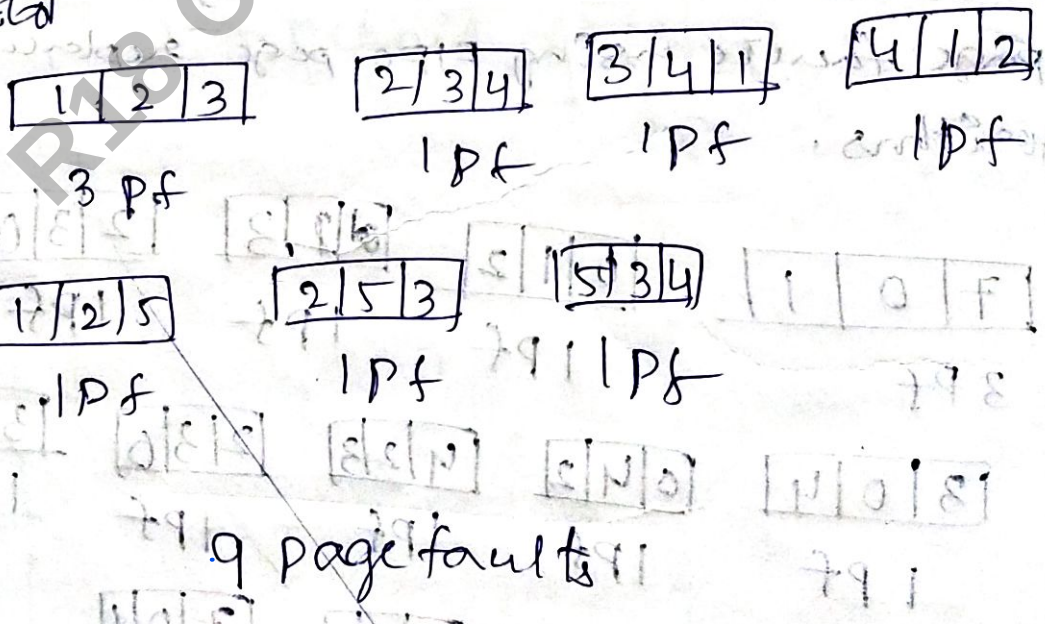
Ans ⇒



⇒ 12 page fault

⇒ FIFO using 3 frames

RS ⇒



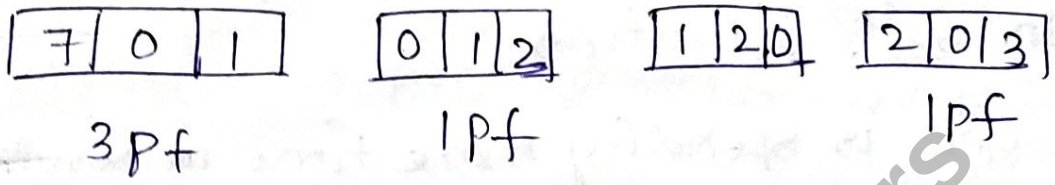
9 page faults

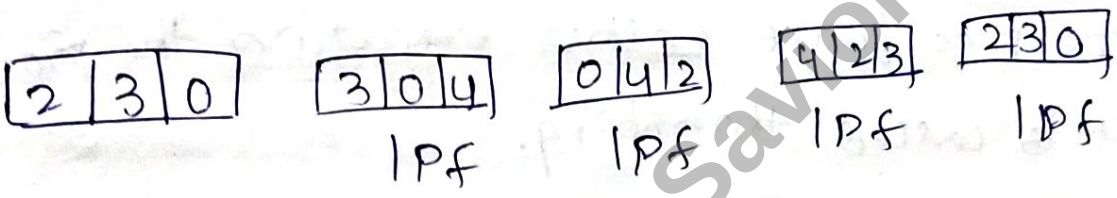
⇒ use the LRU page Replacement algorithm to find no of page faults available free frames = 3

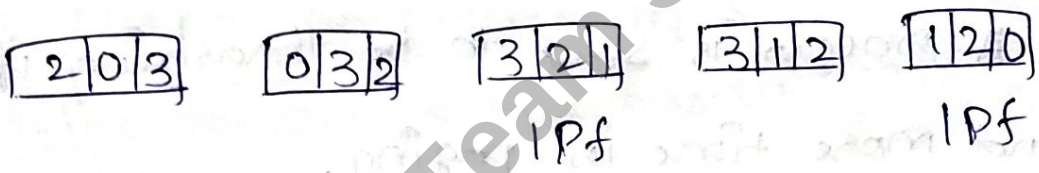
Reference String

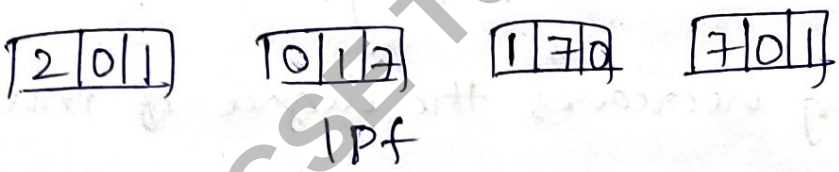
= 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Ans:



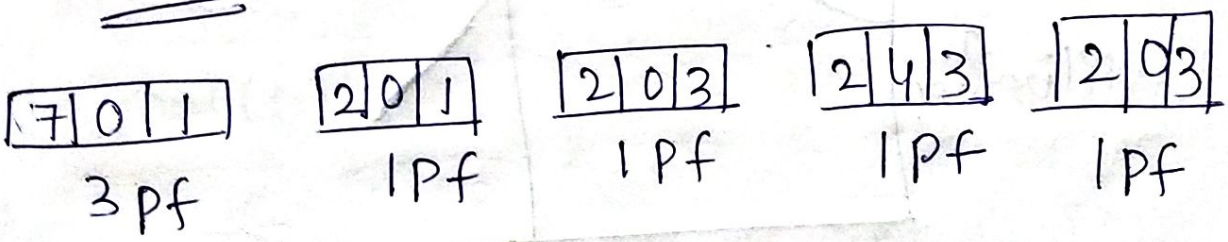


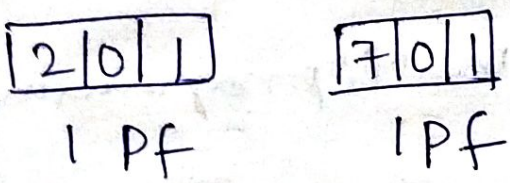




⇒ 12pf

Ans: optimal





⇒ 9 page faults

Thrashing :-

→ The high paging activity is called thrashing.

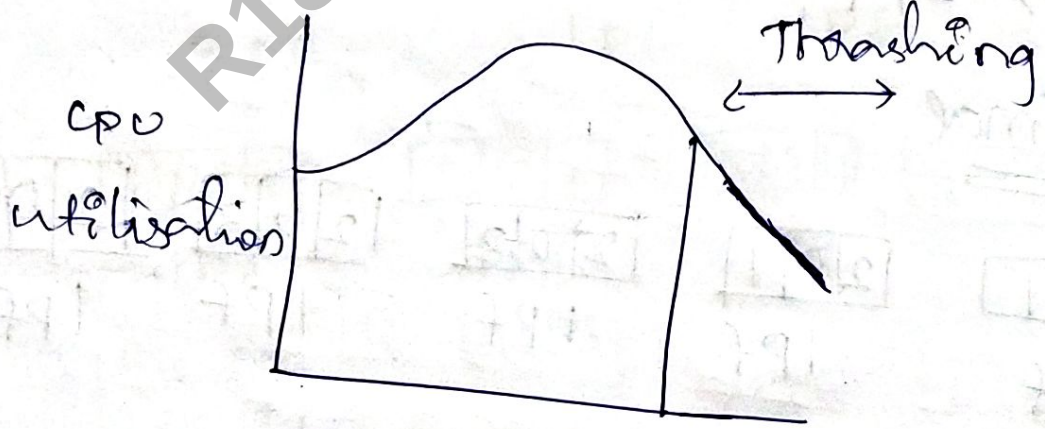
→ If no of page faults are more then the cpu utilisation is less.

→ In such situation,

The CPU is spending more time in servicing the page fault rather executing the process. It is called thrashing.

→ A process is said to be thrashing if it spend more time in paging.

→ Thrashing decreases the degree of multi-programming.



Semaphores :-

→ Semaphores is a integer variable used to implement Mutual exclusion to a critical section problem.

→ It is generally accessed by two methods

1) wait(S) → S-operation → used to decrement S value.

2) signal(S) → P-operation → used to increment S value.

S → Semaphore

$S = 1$

→ S=1 Means no process in CS.

→ P_1 tries to access but has to wait.

Po

{

wait(S) ← wait(S){

↓ while (S<=0)

CS no operation

; S--;

signal(S) }

}

signal(S){

S++;

}

→ If P_0 in cs and $P_1, P_2, P_3, \ldots$ are waiting then if P_0 complete then a random process will enter cs. Hence it satisfies mutual exclusion but bounded waiting is not satisfied.

Semaphore
- Binary (0 or 1)
- counting (takes values in unrestricted domain)

```
wait(S){
    S--;
    if(S<0)
    add process to process blocked list
    block();
}

Signal(S){
    S++;
    if(S<=0)
    wake up first process of blocked list
    wakeup(P)
}
```

classic problem of synchronization:-

1) producer and consumer
2) reader and writer problem
3) dining philosopher's problem

empty = n
mutex = 1
full = 0

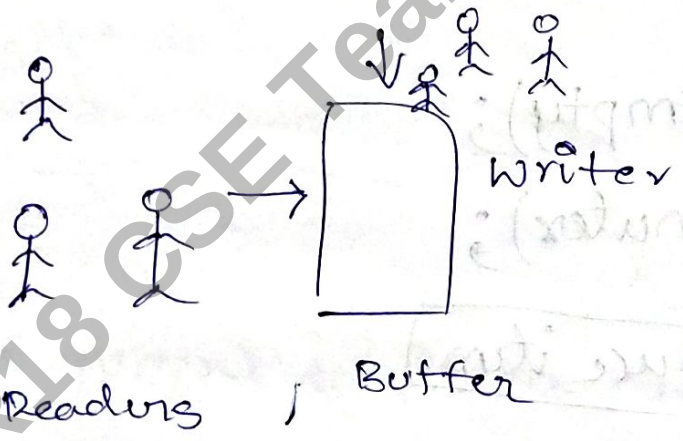
① producer :-

```
do {
    wait (empty);
    wait (mutex);

    produce item

    signal (full);
    signal (mutex);
} while (true);
```

Consumer:

```
do {
    wait(full);
    wait(mutex);
        Consume item;
    signal(empty);
    signal(mutex);
} while(true);
```

② Readers and writers Problem :-



i) When one writer is writing no reader is allowed to read.

ii) When readers are reading no writer is allowed to write

iii) When one reader is reading if one more reader is ready to read please allow.

wrt → binary semaphore

↓ =1 (no one in CS)

=0 (process in CS)

read initially = 0

count = to count the no of readers in

mutex = 1 (for mutual Exclusion on read count by readers).

Writers code :-

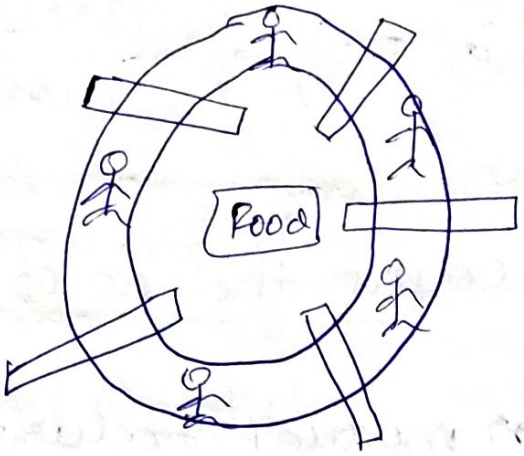
```
do{
  wait(wrt);
  write data
  signal(wrt);
}while(true);
```

Readers code :-

```
do{
  wait(mutex);
    read count++;
  if(readcount==1)
    wait(wrt);
  signal(mutex);
  Read data
  wait(mutex);
  read count--;
  if(readcount==0)
    signal(wrt);
}while(true);
```

Read operation

③ Dining philosopher problem:-



eg:- 5 philo
5 Chopstick

```
do{
wait(chopstick[i])
wait(chopstick[i+1]%5)
    eating;
signal(chopstick[i]);
Signal(chopstick[i+1]%5);
}while(true);
```

UNIT-4

File Management

→ service offered by the OS to the public.

- File system
  - file
  - file system
  - file attributes
  - file operations
  - file access methods
  - file types
  - directory structure
  - file sharing and protection
- File system implementation
  - file system structure
  - file system implementation
  - file space allocation methods
  - free space management
  - efficiency
- Mass-storage structure (secondary)
  - structure of hard disk
  - disk scheduling Algorithm
  - RAID levels.

FILE:-

→ Named collection of related data stored permanently on secondary storage device.

File system:-

→ It is a collection of Files, directory Structure [DS → It provides and organises the information related to all the files in system]

File system Mounting:-

Mount point:-

→ It is the point at which new file system (file/device) is added to the Existing file system.

Ex:- USB = copy

File System Mounting:-

→ The concept of adding a new file System to Existing file system as a new file.

File Attributes:-

- name of the file
- file identifier
- file location
- file size
- file type
- file Protection
- owner
- time, date of creation

File operations :-

- OPEN
- write
- Append
- delete
- Truncate
- Random Access

fopen, fprint, fscanf, fseek()

System calls at OS level

read()
write()
OPEN()

File access Methods :-

- Sequential access → Editor, compiler, Assembler
- Random access (or) direct access → At specific location
- Indexed access

↓

fseek(fp, 2,)

It creates index file.

File types :-

- text file — .txt, .doc
- Zip file — .zip
- programming language — .c, .CPP, .Java,
- Executable file
  .exe, .lib, .bin
- Read only file .ppf, inPEG
- batch file .bat, .sh

DIRECTORY STRUCTURE (folders)

→ It organises the files by keeping entries of a related file.

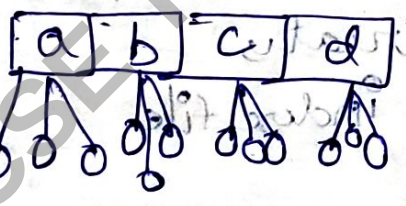
- single level directory
- two level directory
- tree structured directory
- Acyclic Graph directory
- General Graph directory

Single level directory :-

folder □ file ○

→ We cannot create sub folders

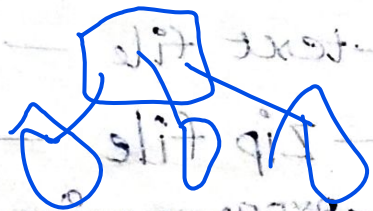
folders :-



Important issues of Directory Structure

1) Name collisions

2) Grouping

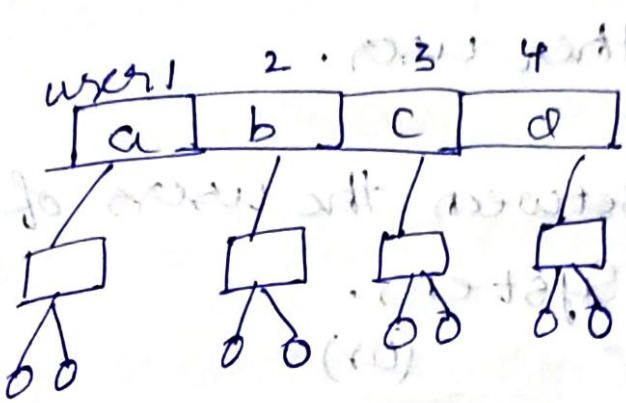


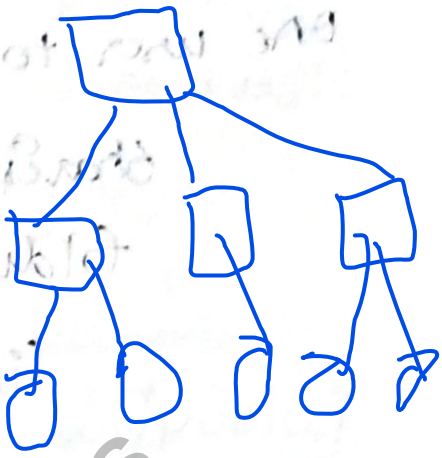
→ All Directory will have same

Issues.

Two-level directory:-

→ Each user can create its own sub-folders

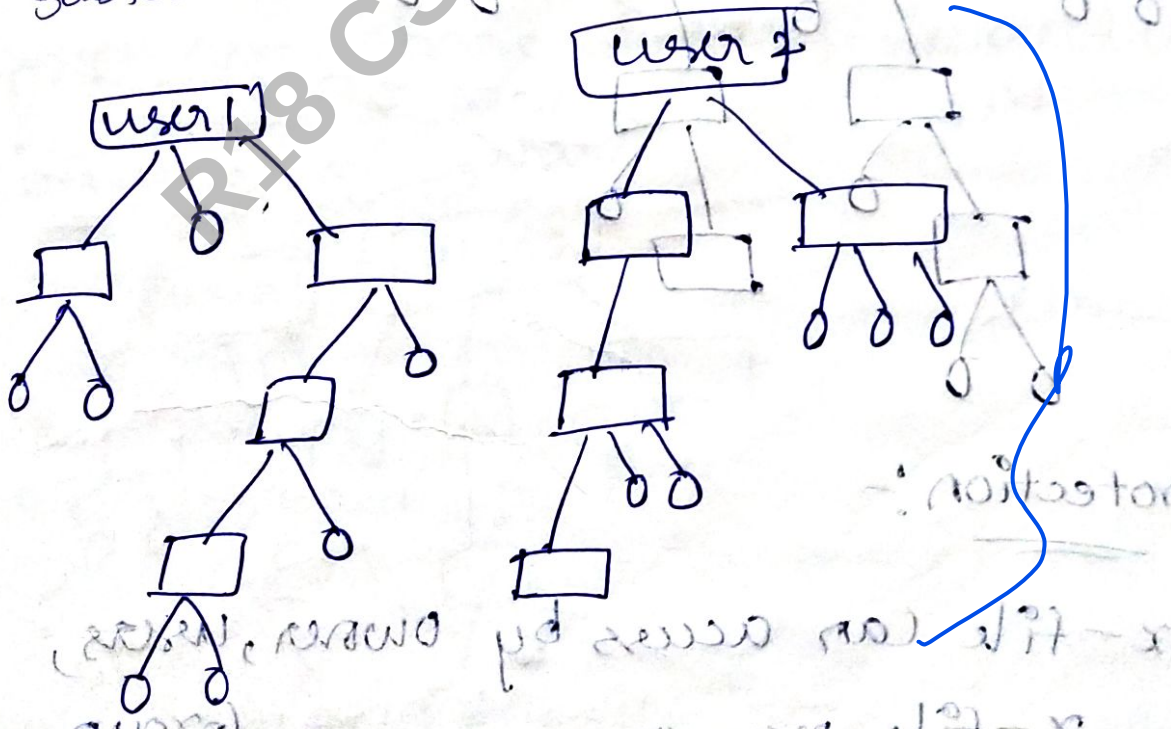




Tree Structured directory:-

→ We can create n sub folders & n files

→ And in subfolders we can create subfolders and files.



⇒ Acyclic graph directory:-

→ In this a folder can share from one user to another user.

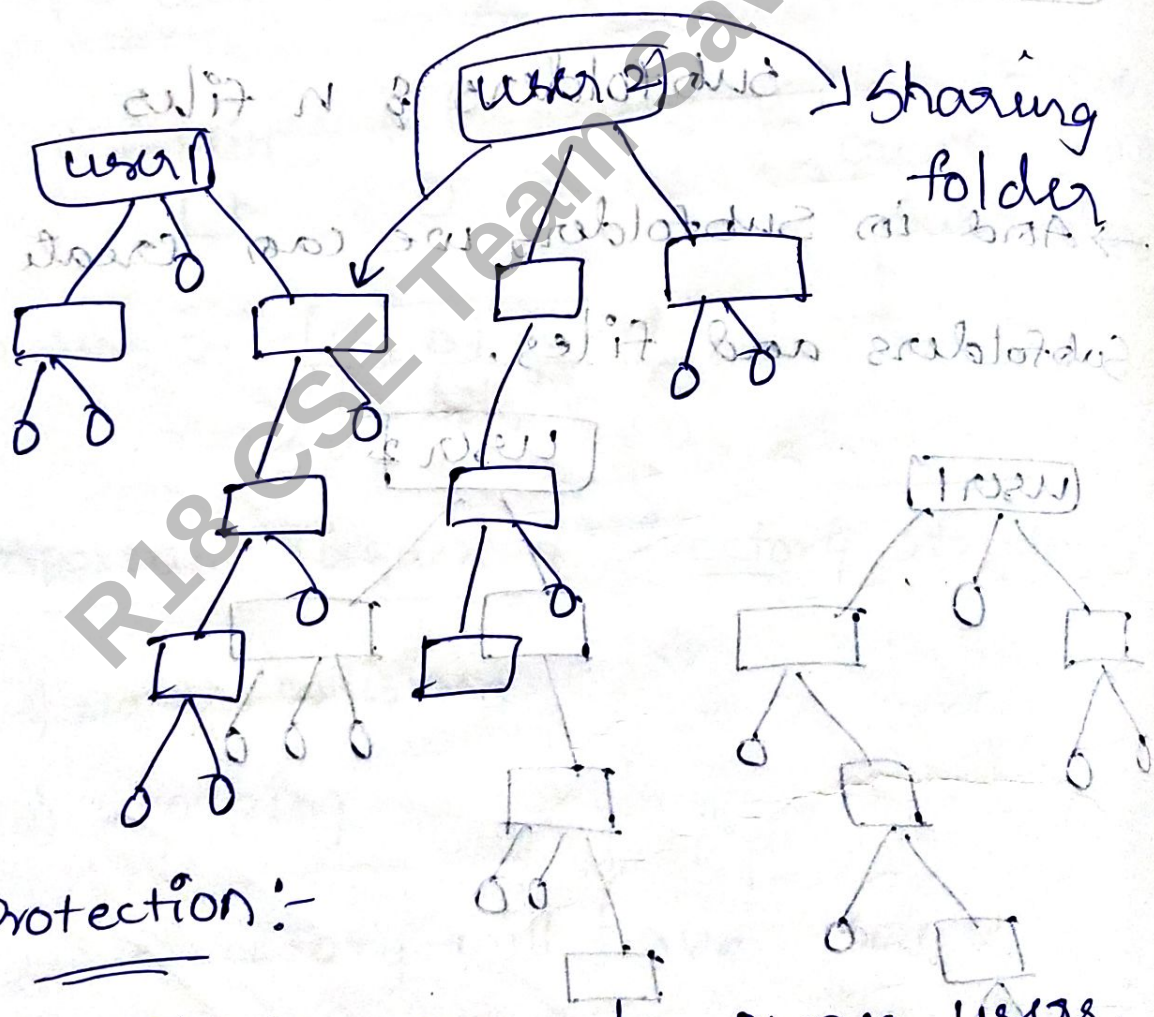
Sharing folder / file } Between the users of System.

(or)

Lan network

(or)

client - Server



Protection:-

x-file can access by owner, users, Group.

x-file memory

read → 4
write → 2
execute → 1

In Linux

command :

>> chmod 761

- 7 → Group
- 6 → owner
- 1 → user
- x → file Permissions for Group, owners

owner can read, write, Execute.

Group can read, write

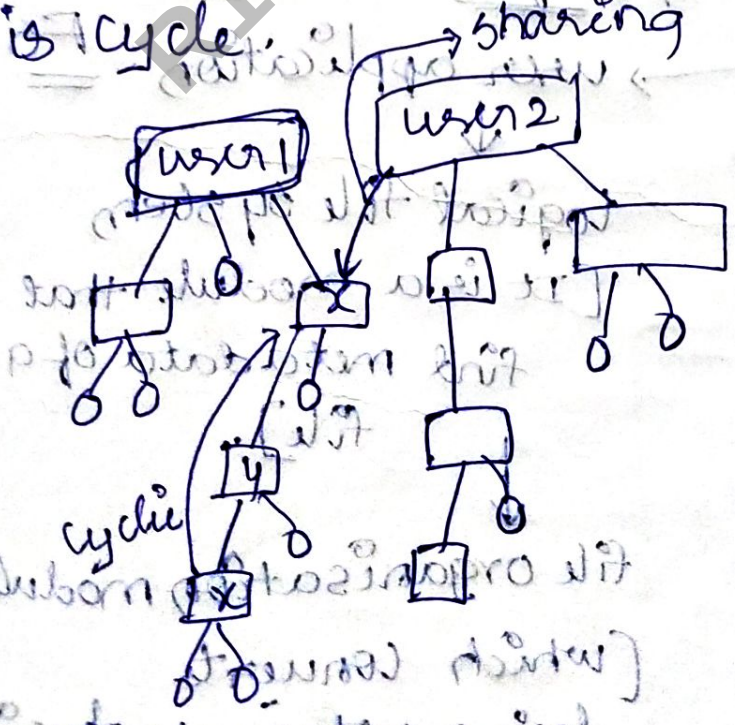
user can Execute

[user/public]

→ In all Windows Acyclic Graph directory is Implemented.

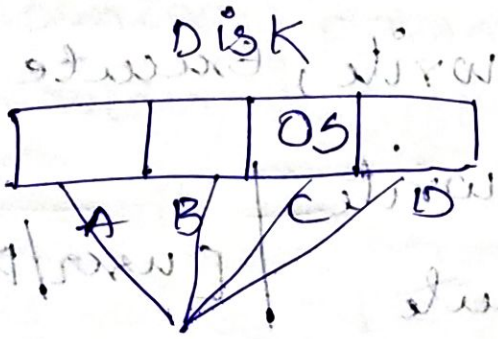
⇒ General graph directory :-

⇒ It can share folder / file and it is cycle.

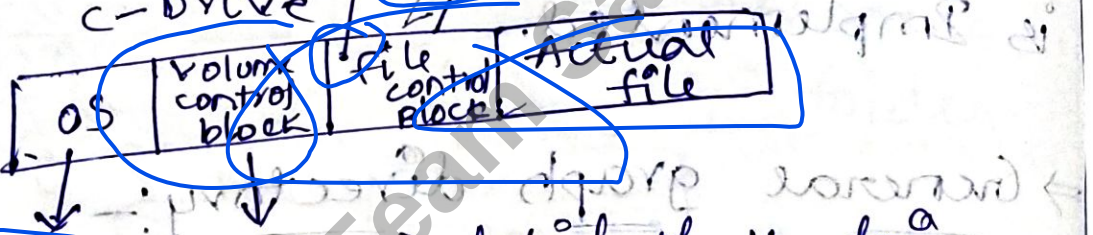


File system implementation

→ File system belongs to each partition.



filesystem → directory structure



boot control block

super block } details of the drive

⇒ Layered structured of FS

user application

Read(x)

logical file system

[It is a module that find metadata of a file]..

file organisation module

[which converts logical Address to physical Address]

↓

Basic file system

[It takes Address from FOM and 90 to that Address]

→ [It creates Generic command 72 cylinder, 32 tract, sector]

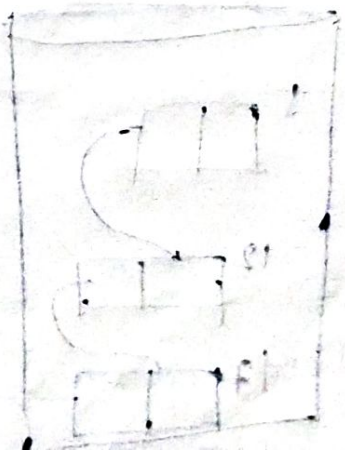
↓

Device control → device controller, device driver

↓

device (HD)

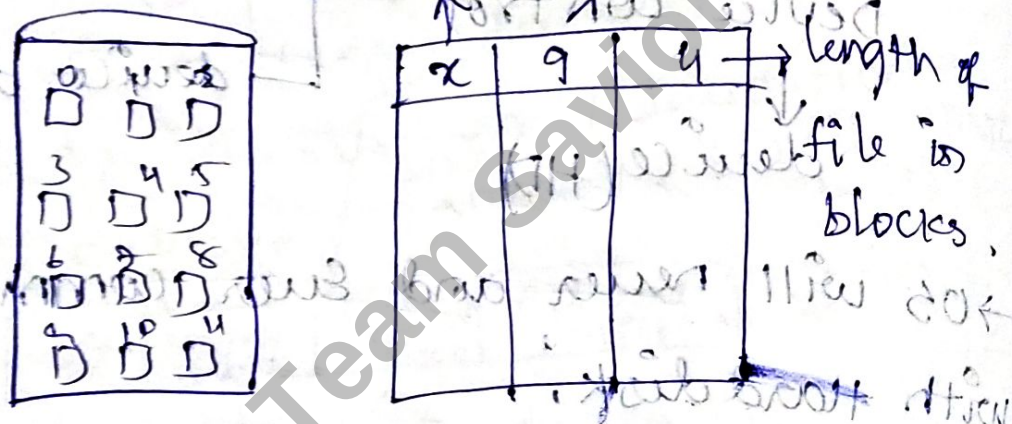
→ OS will never and ever communicate with Hard disk.



File space allocation methods :-

- Continuous Space Allocation
- Linked space allocation
- Indexed space allocation

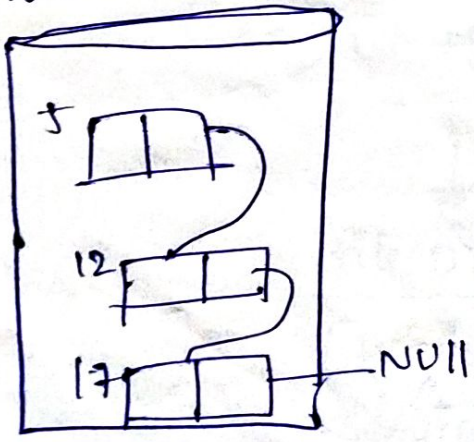
Continuous Space allocation :-

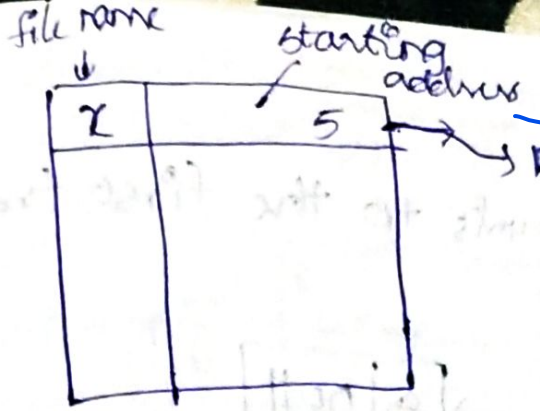


∴ Block has file name x.

Linked Space Allocation :-

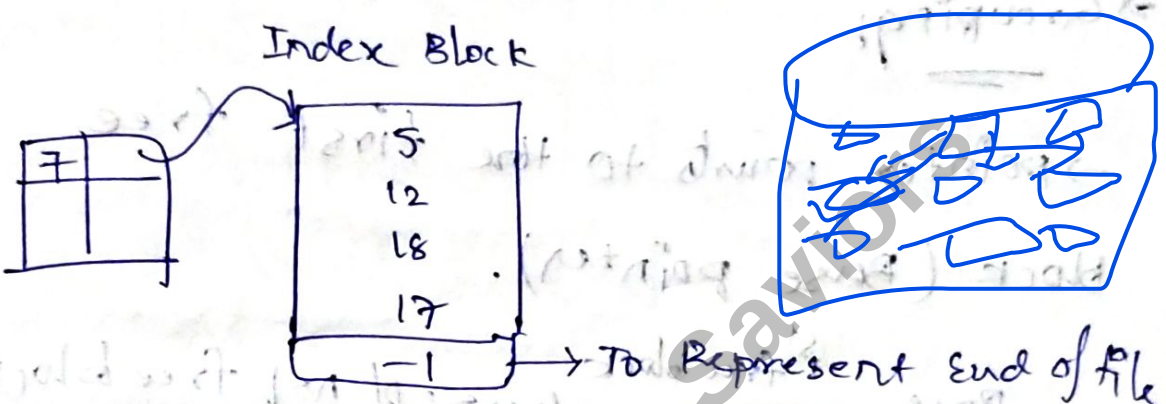
→ The Blocks are connected by pointers.





→ Reference to first Block of the HD

⇒ Indexed Space allocation:-

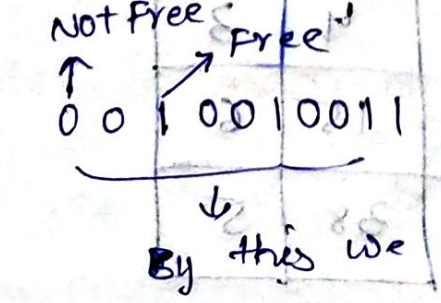


Free Space Management:-

- Bit Vector method
- Linked method
- Grouping
- counting

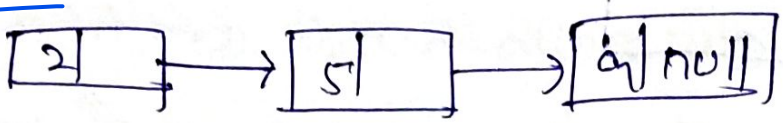
Bit Vector Method:-

→ In this free space is managed to single Vector.



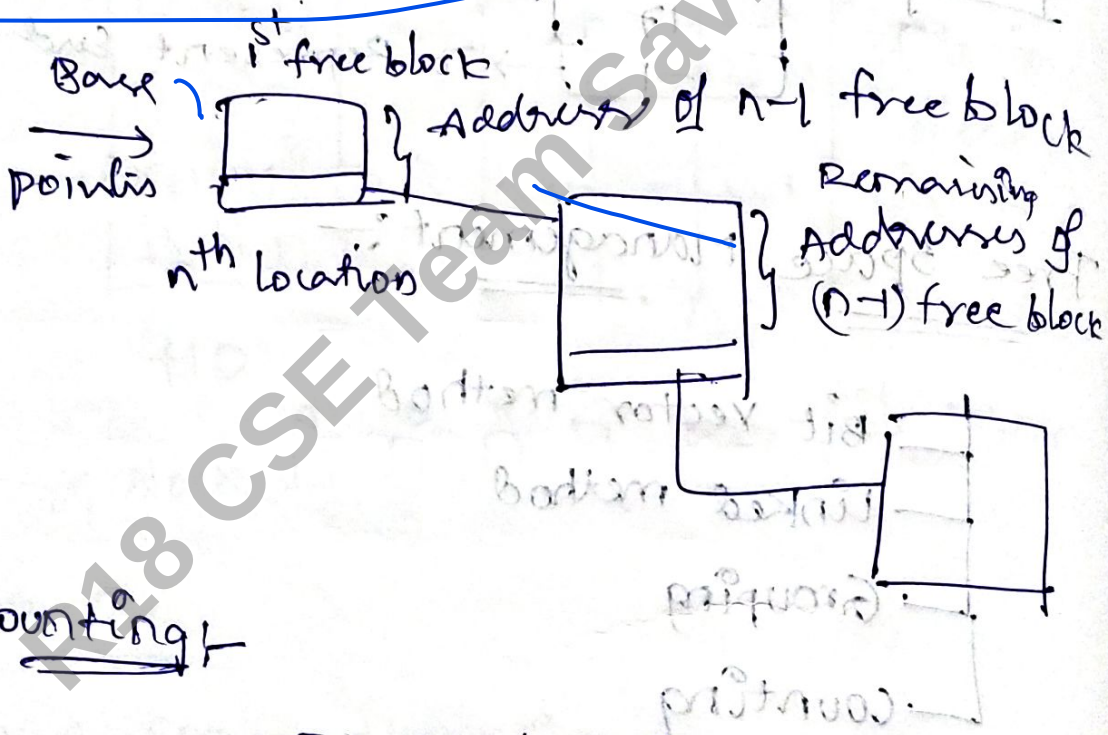
⇒ Linked methods

→ First pointer points to the first free block.



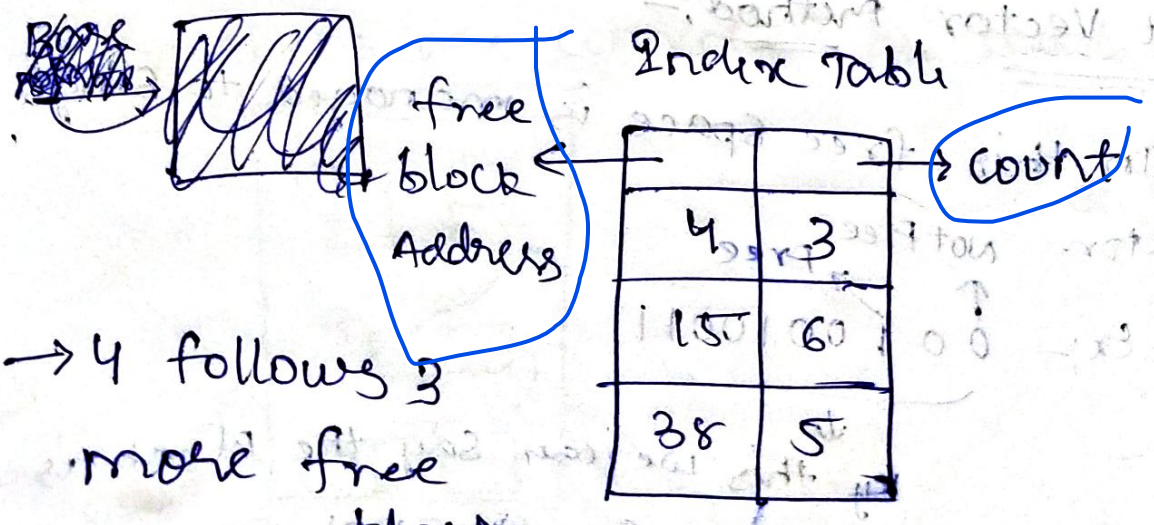
⇒ Grouping

→ Pointer points to the first free block (Base pointer).



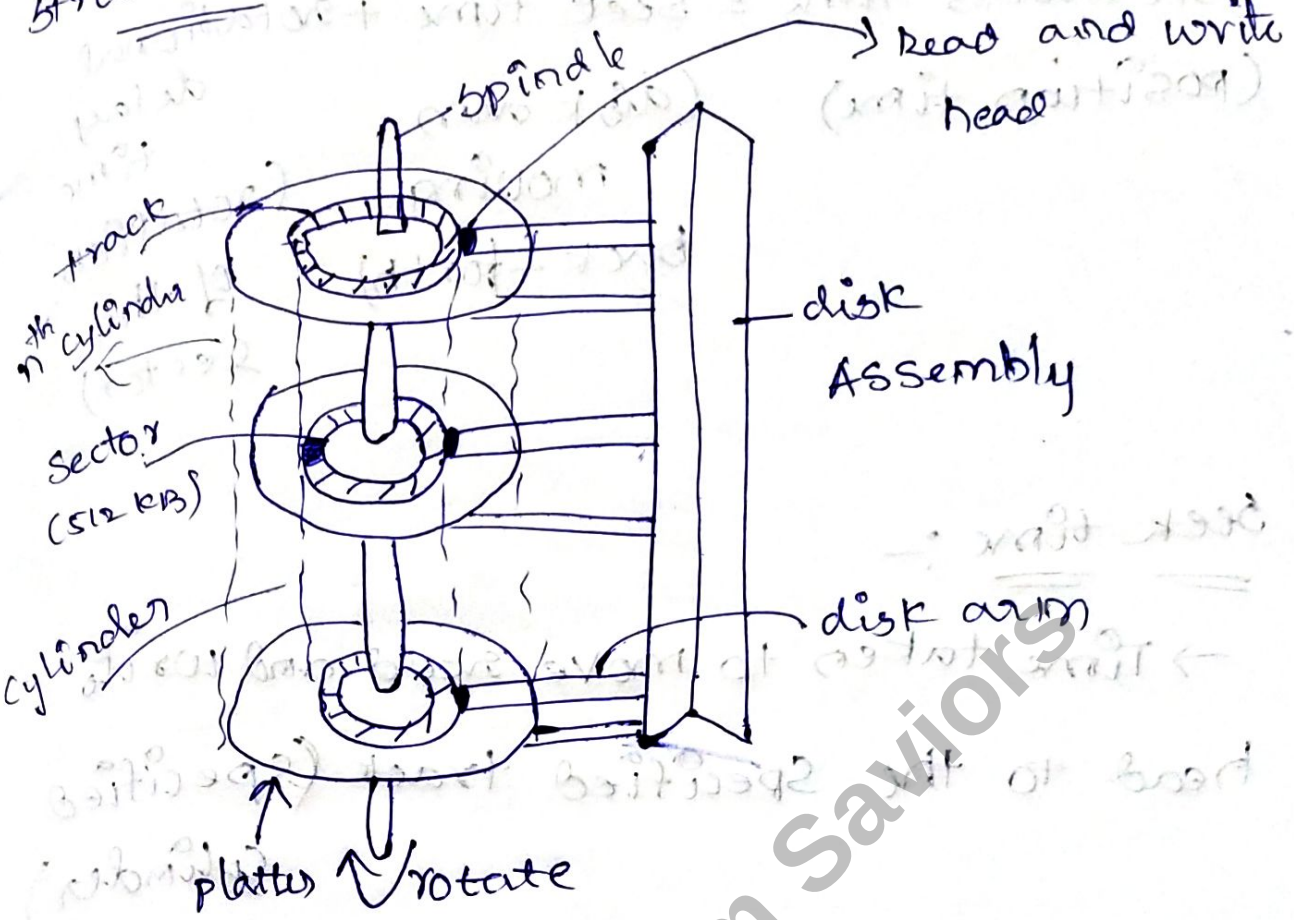
⇒ Counting

→ It maintaince Index table.



→ 4 follows 3 more free blocks

Structure of Hard disk:-



→ platter has magnetic coated material which has ability to store and record data.

platter → divided into many tracks.

Track contains sector.

→ File Reading (x)

↳ data of x started from topmost track.

→ All tracks are of same radius which forms a cylinder.

→ platter is made of n concentric cylinders.

→ When we read data, the data is written on the outermost cylinder.

Disk access time = Seek time + rotational delay time
(position time) (disk arm moving back-forth) (rotation of the sector)

Seek time :-

→ Time taken to move read and write head to the specified track (specified cylinder)

Rotational delay time :-

→ Time taken to rotate specified sector to the read and write head.