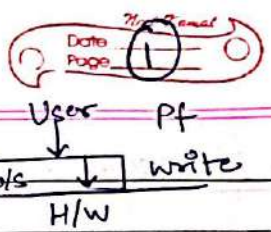
# 1. PROCESS MANAGEMENT



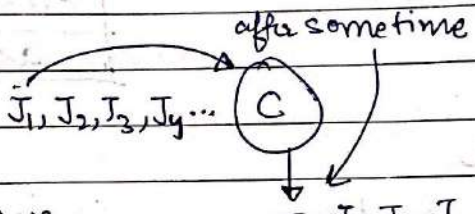

## Introduction to OS

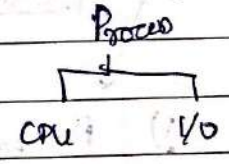
→ OS is an interface between user and hardware.
→ Resource allocation
→ Manager → memory, processes, files, security, etc.

Goals:-
→ Primary → Convienience → Secondary → Efficiency

## Types of OS:-

→ Batch OS
* Starvation
* Efficiency very less
* NOT interactive

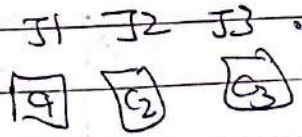



→ Multiprogramming :- * CPU will be used efficiently
* CPU will be busy all the time

→ Multitasking :-
* Interactiveness improved
* Periodically performs job one by one.

→ Multiprocessing :-
* lots of CPU in one computer
* Parallelism improved.

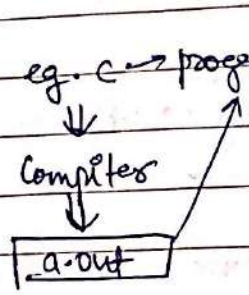


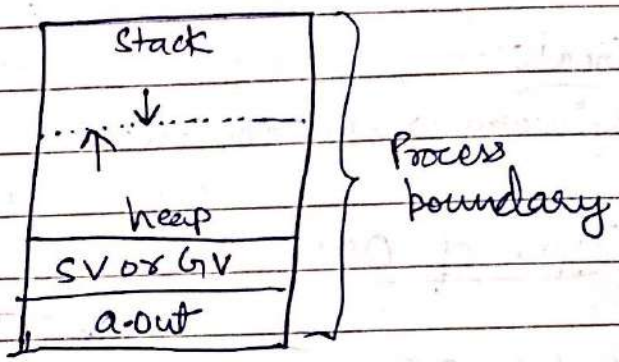

→ Realtime OS :-
* Job in deadlines (strict)
* eg:- Military applications

## Process Management

### Attributes of a process:-

Process:- Something (data-structure) created by OS in order to run a program

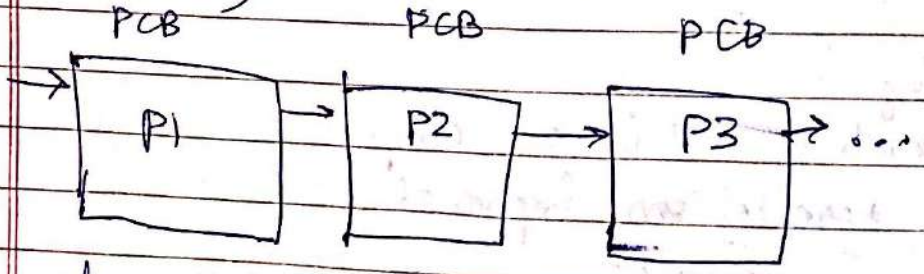



1) Process id
2) Program Counter
3) Process state
4) Priority
5) General Purpose Registers → store before switching processes
6) List of open files
7) List of open devices
8) Protections

(Process Control Block)

* linked list



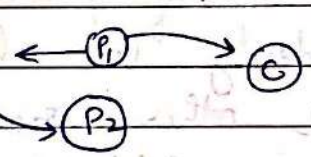

also called as "context"

## States of a Process :-

1) New :- Process is about to created → it will be in secondary memory. Whenever we pick up it to create it in primary memory, we call it new.

2) Ready :- The new process created in main memory is ready to run. There ~~are~~ may be many processes which are ready to run. It is our responsibility that which process to run.

Multiprocessing

- With preemption
  - 
  - Stop a currently running process forcefully & run new process to run (Context switching)
  - → Also called "multitasking" or "Time sharing"
- Without preemption
  - → once you give a process to CPU, you cannot force it to stop ! It will run completely

3) Run :- One process (if system has 1 CPU) which is currently running inside CPU. It is present obviously in main memory.

4) Block or wait :- Process might want to read/write a file (I/O work). whenever any process requires I/O to be performed, we pull it out, put it into block or wait state until it finishes the I/O. Once it finishes I/O, we will bring it back to ready state., it will be in main memory

5) Termination or completion :- Once the process finishes, it is killed and even the context or PCB is deleted.

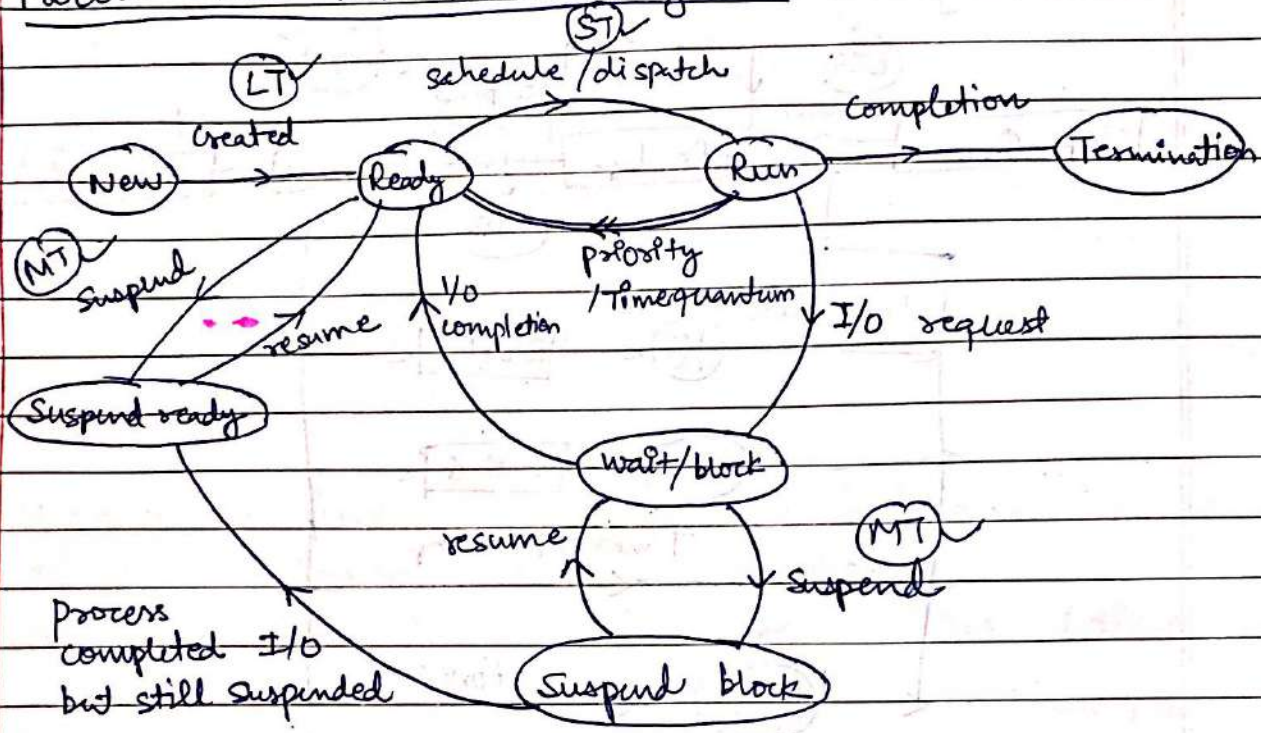
6) Suspend ready :- There are 100's of processes in main memory, and suddenly we like to use main memory for some other process. Make the room for new process which has much more priority, we move some process to secondary memory.

7) Suspend wait or Suspend block :- Suspending processes which are blocked is good as compared to those which are ready.

Operations on processes :-

1) Creation
2) Scheduling
3) Executing
4) Killing / Delete.

## Process State Transition Diagram :-



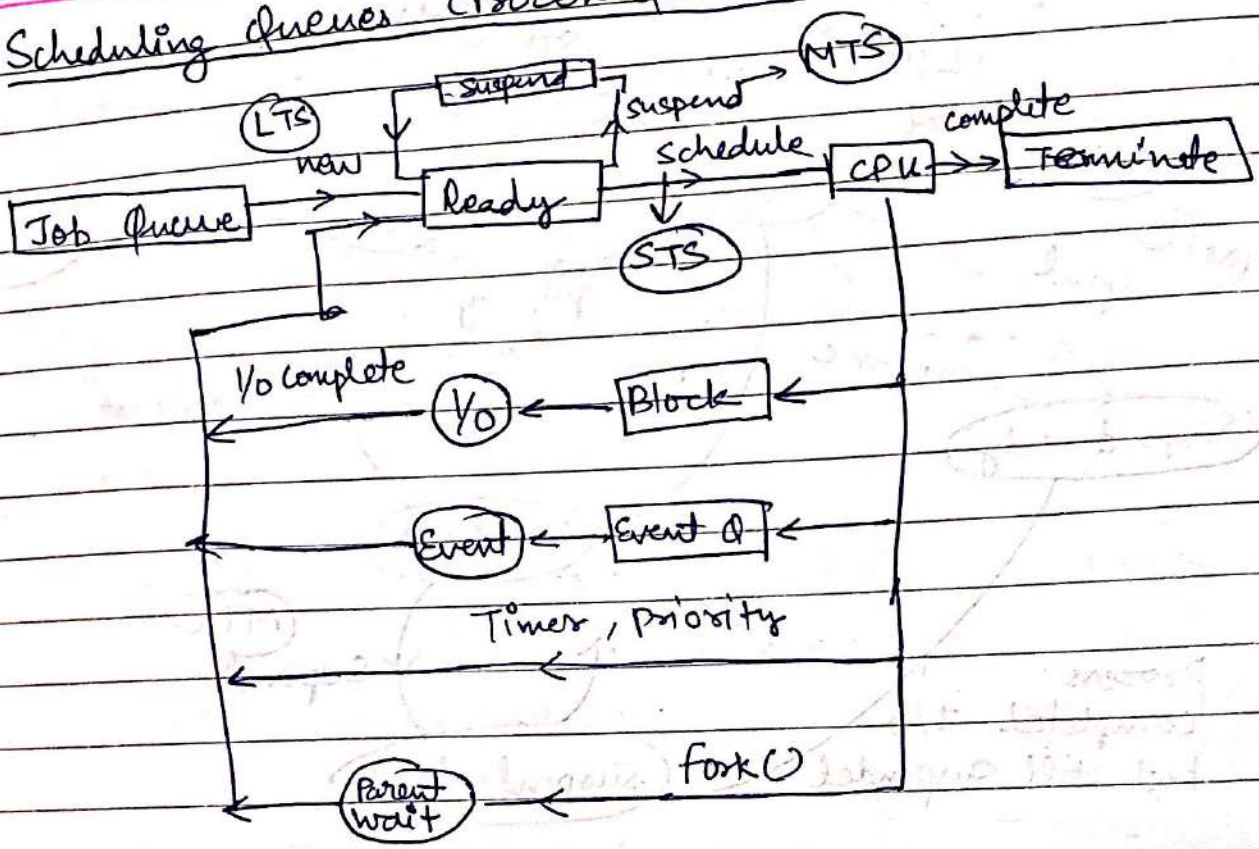

schedulers
- Long term (LT) — creation
- Medium term (MT) — suspension decision
- Short term (ST) / dispatcher

min # states for completion = 4

~~If~~ because of this ~~transaction~~ transition, the entire process becomes multitasking or preemtive multiprogramming

- Degree of multiprogramming :- # process that can be present in ready state at maximum (determined by long-term scheduler)
- Swapping :- Suspension + Resume.
- Dispatcher does the context switching work.
- Short-term scheduler just chooses one process to run from all ready present & rest is done by dispatcher.

## Scheduling queues (Process Queues) :-



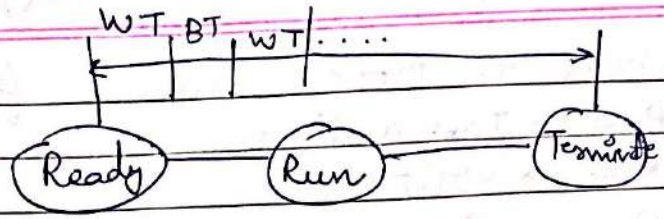

all ☐ boxes ⇒ queues

Q. Consider a system with 'N' CPU processors and 'M' processes, then # processes that can be in :-

| | Min | Max |
|---|---|---|
| ready | 0 | M |
| running | 0 | N |
| block | 0 | M |

## Important Parameters of Processes :-

1) Arrival time :- Time at which a process enters the ready queue. (Point of time)
2) Burst time :- The amount of CPU time taken by a process to finish. (Duration)
3) Completion time :- Time taken by process to finish. (Point of time)




Completion Time − Arrival Time = Burst Time + Waiting Time

Turnaround time

4) Turnaround time :- Difference between the completion time & Arrival time (duration)

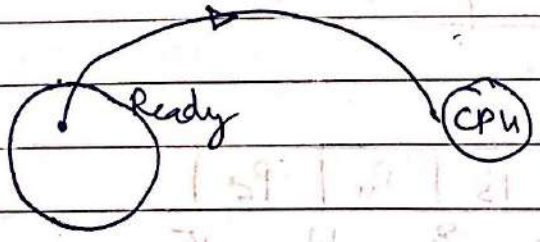
5) Waiting time :- waiting for the CPU (time spent) (duration)

$WT = TAT - BT$

6) Response Time :- First time the process ~~or~~ gets scheduled (time taken) $(FT - AT)$ (duration).

## CPU Scheduling :-

Picking up a process from the ready state and giving it to CPU. who does that ?-
Short Term Scheduler along with the dispatcher.

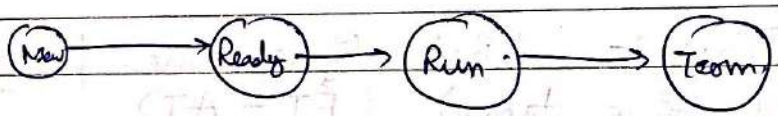



who :- short term scheduler
where :- Ready state to run state

when :- when a process moves from
1) Run → Termination
2) Run → wait
3) Run → Ready
2) New → Ready i.e. when a process is just created.
3) wait → ready

## First Come First Serve (FCFS)

criteria : ~~Arrival~~ time
mode : Non-preemptive

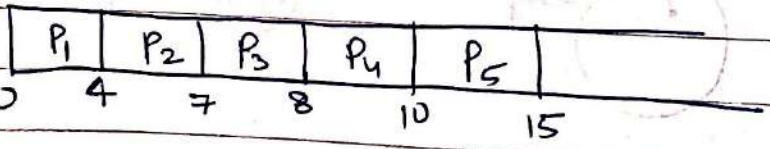



No Block state & preemption → assumption

eg.

| PNo | AT (Arrival time) | BT (Burst time) |
|---|---|---|
| 1 | 0 | 4 |
| 2 | 1 | 3 |
| 3 | 2 | 1 |
| 4 | 3 | 2 |
| 5 | 4 | 5 |

• In non-preemptive ~~time~~ algo, waiting time = response time.




| PNo | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|
| CT | 4 | 7 | 8 | 10 | 15 |
| TT (Turnaround time) | 4 | 6 | 6 | 7 | 11 |
| WT (Waiting time) | 0 | 3 | 5 | 5 | 6 |

WT + BT = TT

Avg. TT = 34/5

Avg. WT = 19/5

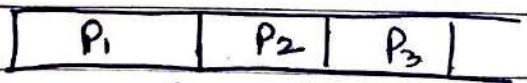
- If 2 process have same AT, lower PID exec. first

## Convoy Effect :-

Also called Starvation

Eg:

| PNo | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 20 | 20 | 20 | 0 |
| 2 | 1 | 2 | 22 | 21 | 19 |
| 3 | 1 | 1 | 23 | 22 | 21 |

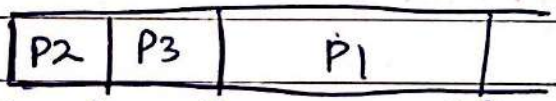



avg TT = 21

avg WT = 40/3 ≃ 13.33

Eg:

| PNo | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| P1 | 1 | 20 | 23 | 22 | 2 |
| P2 | 0 | 2 | 2 | 2 | 0 |
| P3 | 0 | 1 | 3 | 3 | 2 |




convoy effect

Avg TT = 9

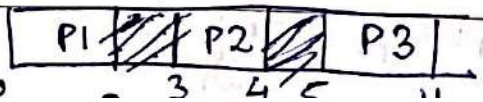
Avg WT = 4/3 ≃ 1.33

Convoy Effect :- If a process with large burst time, arrives early, then other process may have to starve long. causing the increase in avg. W.T.

Eg

| PNo | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 2 | 2 | 2 | 0 |
| 2 | 3 | 1 | 4 | 1 | 0 |
| 3 | 5 | 6 | 11 | 6 | 0 |

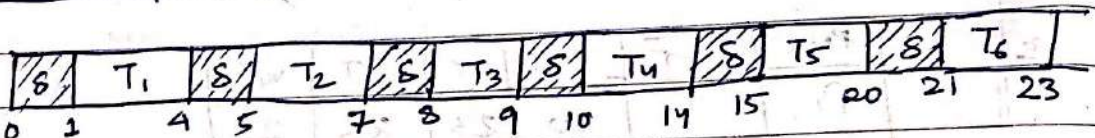
- DS to implement FCFS = queue
- Time = O(n)



eg. FCFS with overhead

$\delta$ = 1 unit

| PNo | AT | BT |
|---|---|---|
| 1 | 0 | 3 |
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 3 | 4 |
| 5 | 4 | 5 |
| 6 | 5 | 2 |




Inefficiency = $6/23 \times 100\ \%$

$\eta = \left(1 - \frac{6}{23}\right) \times 100\ \%$

## Shortest Job First (SJF)

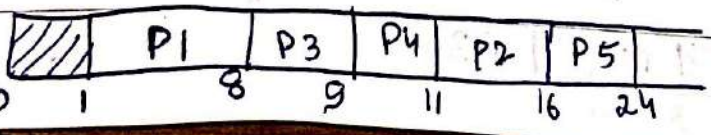
(Among available process - SJF)

criteria :- Burst time
mode :- Non preemptive

- DS :- min heap
- Time → O(n log n)

eg.

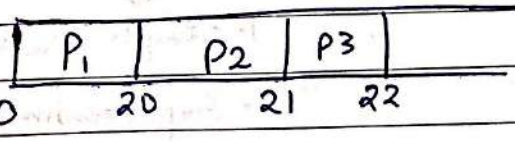
| PNO | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| ✓ 1 | 1 | 7 | 8 | 7 | 0 |
| ✓ 2 | 2 | 5 | 16 | 14 | 9 |
| ✓ 3 | 3 | 1 | 9 | 6 | 5 |
| ✓ 4 | 4 | 2 | 11 | 7 | 5 |
| ✓ 5 | 5 | 8 | 24 | 19 | 11 |



- If two process have same burst time → choose which arrived early

eg. Convoy effect in SJF

| PNO | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 20 | 20 | 20 | 0 |
| 2 | 1 | 1 | 21 | 20 | 19 |
| 3 | 2 | 1 | 23 | 21 | 20 |

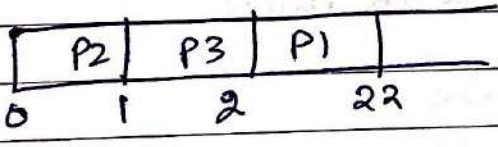



avg WT = $\frac{39}{3}$ = 13

Convoy effect

eg.

| PNo | AT | BT | CT | TT | WT |
|---|---|---|---|---|---|
| 1 | 2 | 20 | 22 | 20 | 0 |
| 2 | 0 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 2 | 1 | 0 |




avg WT = 0

Throughput = $\frac{\text{No. of proces}}{\text{unit time}}$

= $\frac{3}{22}$

* Not practical to implement because burst time is NOT known prior to execution.

SJF

Advantages
→ maximum throughput
→ minimum avg WT and TAT

Disadvantages
→ Starvation to longer jobs
→ It is not implementable because BT of processes cannot be known ahead.

Solution:- SJF with predicted BT.

Prediction techniques

- Static
  - 1. Process size
  - 2. Process type
- Dynamic
  - 1. simple averaging
  - 2. Exponential averaging / Aging

Process Size :- (Bytes)

$P_{old}$ = 200 KB ⇒ 20 units

$\therefore P_{new}$ = 201 KB ⇒ 20 units

Process type :-

- Process
  - OS
    - 3-5 units
  - User
    - Inter-active (5-8)
    - foreground (10-15)
    - background (15-20)

Simple Average :-

Given n-processes $(P_1 \dots P_n)$

→ let $t_i$ be the actual BT

→ let $\tau_i$ denotes predicted BT

$$\tau_{n+1} = \frac{1}{n} \sum_{i=1}^{n} t_i$$

## Exponential Average / Aging :-

$$\mu_{n+1} = \alpha t_n + (1-\alpha)\mu_n \quad , 0 \le \alpha \le 1 \quad -(1)$$

$\alpha$ :- smoothening factor

$$\mu_n = \alpha t_{n-1} + (1-\alpha)\mu_{n-1} \quad -(2)$$

Subst. (2) in (1) :-

$$\mu_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \mu_{n-1}$$

$$= \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 [\alpha t_{n-2} + (1-\alpha)\mu_{n-2}]$$

$$= \dots$$

$\mu_{n+1}$ depends on $\underbrace{t_n \cdots t_1}_{\text{all previous process}}$, $\mu_n \cdots \mu_1$, all.

eg $\alpha = 0.5$, $\mu_1 = 10$, actual BT $(t_1, t_2, t_3, t_4) = (4, 8, 6, 7)$

then $\mu_5 = ?$

$\mu_2 = 0.5 \times 4 + 0.5 \times 10$

$= 7$

$\mu_3 = 0.5 \times 8 + 0.5 \times 7$

$= 7.5$

$\mu_4 = 0.5 \times 6 + 0.5 \times 7.5$

$= 6.75$

$\mu_5 = 0.5 \times 7 + 0.5 \times 6.75$

$= \boxed{6.875}$

## Shortest Remaining Time First (SRTF):

mode:- preemptive
criteria:- burst time

Q

Sol

eg.

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 1 | 0 | 7 | 19 | 19 | 12 | 0 |
| 2 | 1 | 5 | 13 | 12 | 7 | 0 |
| 3 | 2 | 3 | 6 | 4 | 1 | 0 |
| 4 | 3 | 1 | 4 | 1 | 0 | 0 |
| 5 | 4 | 2 | 9 | 5 | 3 | 3 |
| 6 | 5 | 1 | 7 | 2 | 1 | 1 |

Response time ↳ time to exec for 1st time after arrival

avg WT = 4

| P1 | P2 | P3 | P4 | P3 | P3 | P6 | P5 | P2 | P1 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 13 | 19 |

P1 P2 P3 P4 P5 P6
~~6~~ ~~4~~ ~~3~~ ~~0~~ ~~2~~ ~~1~~
~~1~~

- If 2 process have same burst time, choose early arrival

Gate 2011

Q

Sol.

| PNo | AT | BT | CT | ~~what is avg~~ TAT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 9 | 13 | 13 | 4 |
| 2 | 1 | 4 | 5 | 4 | 0 |
| 3 | 2 | 9 | 22 | 20 | 11 |

what is avg waiting time using SRTF?

| P1 | P2 | P2 | P1 | P3 |
|---|---|---|---|---|
| 0 | 1 | 2 | 5 | 13 | 22 |

P1 P2 P3
~~8~~ ~~4~~ 9

avg WT = $\frac{15}{3}$.

= 5

Sol.

Q

| PNo | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 20 | 20 | 20 | 0 |
| 2 | 15 | 25 | 65 | 40 | 15 |
| 3 | 30 | 10 | 40 | 10 | 0 |
| 4 | 45 | 15 | 70 | 25 | 10 |

Gate 07

waiting time of P2 using SRTF? 15

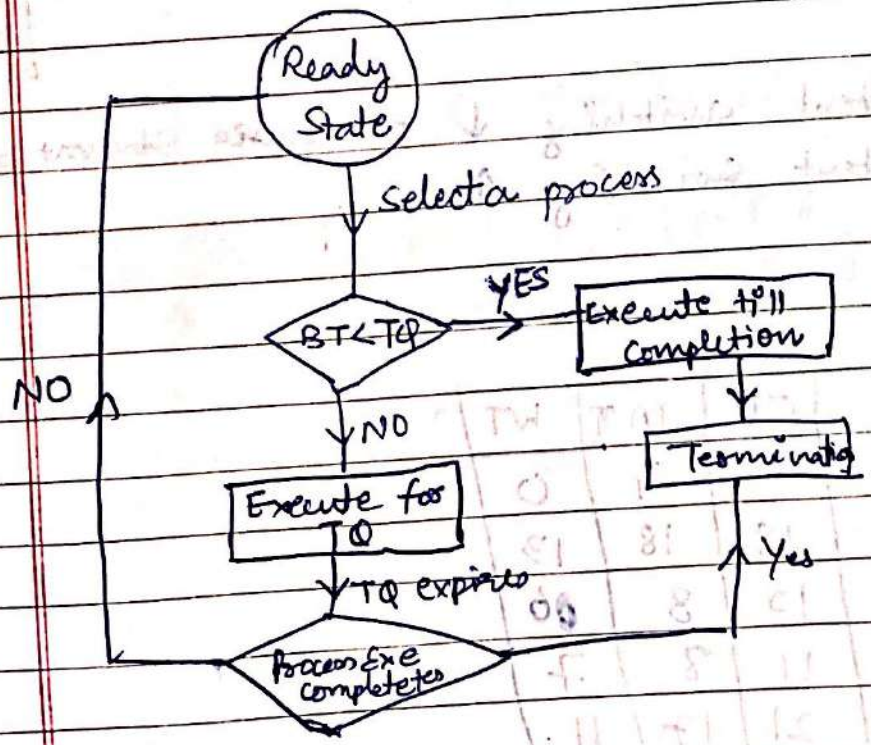
| P1 | P1 | P2 | P3 | P2 | P2 | P4 |
|---|---|---|---|---|---|---|

0 15 20 30 40 45 55 70

P1 P2 P3 P4
~~20~~ ~~25~~ ~~10~~ ~~15~~
~~5~~ ~~15~~
~~10~~

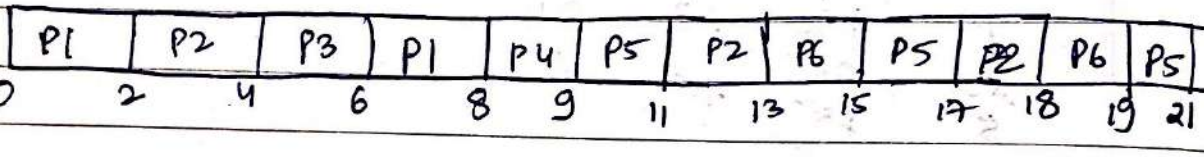
## Round Robin Algorithm :-

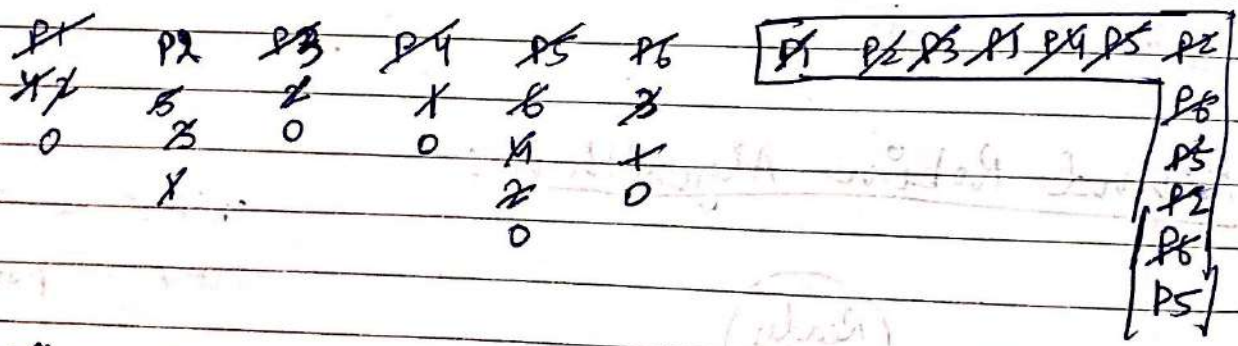
criteria :- TQ+AT
mode :- preemptive



eg. TQ = 2

| PNo | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 8 | 8 | 4 |
| 2 | 1 | 5 | 18 | 17 | 12 |
| 3 | 2 | 2 | 6 | 4 | 2 |
| 4 | 3 | 1 | 9 | 6 | 5 |
| 5 | 4 | 6 | 21 | 17 | 11 |
| 6 | 6 | 3 | 19 | 13 | 10 |



- TQ↑, # Context switching ↓ → causes Starvation
- TQ↓, # Context switching ↑

eg. TQ = 4

| PNo | AT | BT | CT | TAT | WT |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 4 | 4 | 0 |
| 2 | 1 | 5 | 19 | 18 | 13 |
| 3 | 2 | 2 | 10 | 8 | 6 |
| 4 | 3 | 1 | 11 | 8 | 7 |
| 5 | 4 | 6 | 21 | 17 | 11 |
| 6 | 6 | 3 | 18 | 12 | 9 |

Queue | P1 P2 P3 P4 P5 P6 P2 P5 |

Gnatt chat

| P1 | P2 | P3 | P4 | P5 | P6 | P2 | P5 |
|---|---|---|---|---|---|---|---|

0 4 8 10 11 15 18 19 21

P1 P2 P3 P4 P5 P6

4 5 2 1 6 3

0 1 0 0 2 0

0 0

eg. TQ=3

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 1 | 5 | 5 | 32 | 27 | 22 | 10 |
| 2 | 4 | 6 | 27 | 23 | 17 | 5 |
| 3 | 3 | 7 | 33 | 30 | 23 | 3 |
| 4 | 1 | 9 | 30 | 29 | 20 | 0 |
| 5 | 2 | 2 | 6 | 4 | 2 | 2 |
| 6 | 6 | 3 | 21 | 15 | 12 | 12 |

P4 P5 P3 P2 P4 P1 P6 P3 P2 P4 P1 P3

| | P4 | P5 | P3 | P2 | P4 | P1 | P6 | P3 | P2 | P4 | P1 | P3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 1 4 6 9 12 15 18 21 24 27 30 32 33

P1 P2 P3 P4 P5 P6

3 3 1 3 2 3

2 2 4 2 0 0

0 0 1 0

0 0

- TQ ↑ ⇒ avg RT ↑ and CS ↓
  TQ ↓ ⇒ avg RT ↓ and CS ↑
- TQ = ∞ ⇒ FCFS

eg. Consider 4 jobs P1, P2, P3 and P4 arriving in ready queue in the same order at time $t = 0$, if BT requirements of these jobs are 4, 1, 8, 1 respectively, what is the completion time of P1, assuming round robin with TQ = 1

Sol: P1 P2 P3 P4 P1 P3 P1 P3 P1 P3 P1 P3

| P1 | P2 | P3 | P4 | P1 | P3 | P1 | P3 | P1 | P3 | P3 | P3 | P3 | P3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

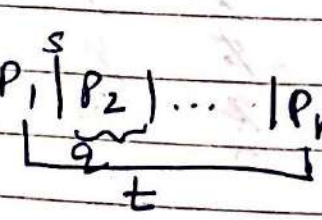
| P1 | P2 | P3 | P4 |
|---|---|---|---|
| 4 | 1 | 8 | 1 |
| 3 | 0 | 7 | 0 |
| 2 | | 6 | |
| 1 | | 5 | |
| 0 | | | |

Comp(P1) = 9.

eg. Consider 'n' processes sharing the CPU in RR fashion. If the context switching time is 's' units. What must be time quantum 'q' such that the no. of context switches are reduced, but at the same time each process is guaranteed to get the turn at the CPU for every 't' seconds.

Sol: $ns + (n-1)q \leq t$

$\Rightarrow q \leq \frac{t - ns}{(n-1)}$



## Longest Job First (LJF) :-

criteria :- Burst time
mode :- Non-Preemptive
Process having longest BT gets scheduled first

eg.

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 1 | 0 | 3 | 3 | 3 | 0 | 0 |
| 2 | 1 | 2 | 20 | 19 | 17 | 17 |
| 3 | 2 | 4 | 18 | 16 | 12 | 12 |
| 4 | 3 | 5 | 8 | 5 | 0 | 0 |
| 5 | 4 | 6 | 14 | 10 | 4 | 4 |

| P1 | P4 | P5 | P3 | P2 |
|---|---|---|---|---|
| 0 | 3 | 8 | 14 | 18 | 20 |

## Longest Remaining Time First :-

Criteria :- Burst time + Arrival Time
mode :- Preemptive

eg.

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 18 | 17 | 15 | 0 |
| 2 | 2 | 4 | 19 | 17 | 13 | 0 |
| 3 | 3 | 6 | 20 | 17 | 11 | 0 |
| 4 | 4 | 8 | 21 | 17 | 9 | 0 |

| | P1 | P2 | P3 | P4 | P3 | P4 | P3 | P4 | P2 | P3 | P4 | P2 | P3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

P1 P2 P3 P4

| P4 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 17 | 18 | 19 | 20 | 21 |

- BT $\xrightarrow{\text{same go to}}$ AT least $\xrightarrow{\text{same go to}}$ PNo least


Gate 06.

Q ——— Sol.

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 12 | 12 | 10 | 8 |
| 2 | 0 | 4 | 13 | 13 | 9 | 4 |
| 3 | 0 | 8 | 14 | 14 | 6 | 0 |

what is avg TAT using LRTF?

| P3 | P2 | P3 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P3 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

~~P1 P2 P3~~
~~2 4 8~~
~~1 3 4~~
0 ~~2~~ ~~3~~
~~1~~ 2
0 1
0

avg (TAT)
$= \frac{12+13+14}{3}$
$= \boxed{13}$

## Highest response ratio Next (HRRN) :-

Criteria :- Response Ratio (RR) = $\frac{W+S}{S}$

W :- waitime time for a process so far

S :- Service time of a process or BT.

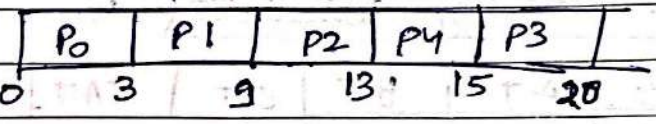
mode :- Non-preemptive

→ HRRN not only favours shorter jobs but also limits the waiting time of longer jobs.

eg.

| PNo | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 3 | 3 | 0 | 0 |
| 1 | 2 | 6 | 9 | 7 | 1 | 1 |
| 2 | 4 | 4 | 13 | 9 | 5 | 5 |
| 3 | 6 | 5 | 20 | 14 | 9 | 9 |
| 4 | 8 | 2 | 15 | 7 | 5 | 5 |

HRRN :-

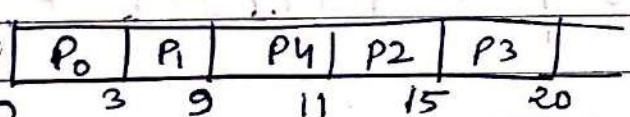



$RR(P2) = \frac{5+4}{4} = 2.25$ $RR(P3) = \frac{3+5}{5} = 1.6$

$RR(P4) = \frac{1+2}{2} = 1.5$

$RR(P3) = \frac{7+5}{5} = 2.4$ $RR(P4) = \frac{5+2}{2} = 3.5$

SJF :-




(same eg.)

## Priority Scheduling :-

Priority

- Static
  - ↓
  - Doesn't change throughout the execution of the process
- Dynamic
  - ↓
  - Changes at regular intervals of time

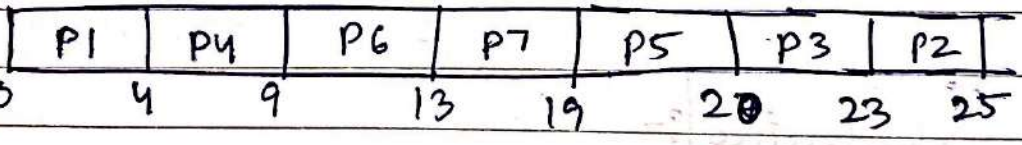
Priority Scheduling

- Preemptive
- Non-preemptive

## Non-Preemptive Priority scheduling :-

eg:

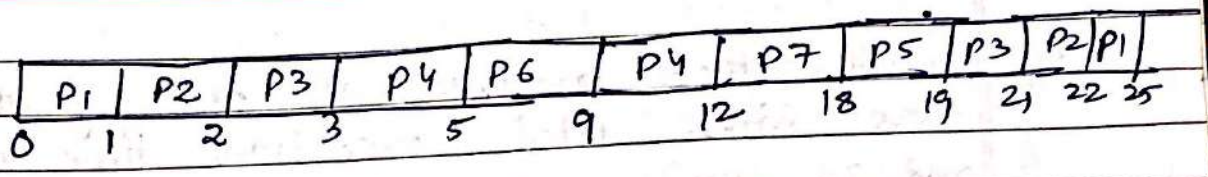
| PNo | Priority | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|---|
| 1 | 2 (L) | 0 | 4 | 4 | 4 | 0 | 0 |
| 2 | 4 | 1 | 2 | 25 | 24 | 22 | 22 |
| 3 | 6 | 2 | 3 | 23 | 21 | 18 | 18 |
| 4 | 10 | 3 | 5 | 9 | 6 | 1 | 1 |
| 5 | 8 | 4 | 1 | 20 | 16 | 15 | 15 |
| 6 | 12 (H) | 5 | 4 | 13 | 8 | 4 | 4 |
| 7 | 9 | 6 | 6 | 19 | 13 | 7 | 7 |




## Preemptive Priority scheduling :-

eg:

| PNo | Priority | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 4 | 25 | 25 | 21 | 0 |
| 2 | 4 | 1 | 2 | 22 | 21 | 19 | 0 |
| 3 | 6 | 2 | 3 | 21 | 19 | 16 | 0 |
| 4 | 10 | 3 | 5 | 12 | 9 | 4 | 0 |
| 5 | 8 | 4 | 1 | 19 | 15 | 14 | 14 |
| 6 | 12 | 5 | 4 | 9 | 4 | 0 | 0 |
| 7 | 9 | 6 | 6 | 18 | 12 | 6 | 6 |

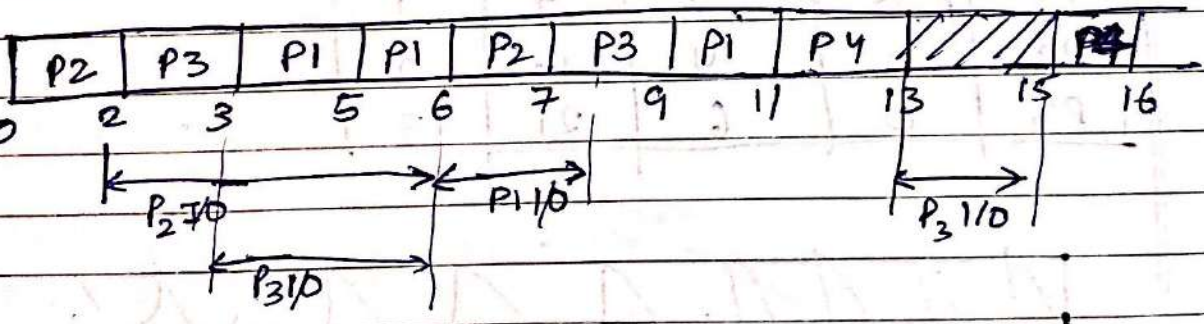



~~P1 P2 P3 P4 P5 P6 P7~~
4 2 3 5 X 4 5
3 X 2 5 0 0 0
0 0 0

SRTF with processes contains CPU and I/O time:-

eg.

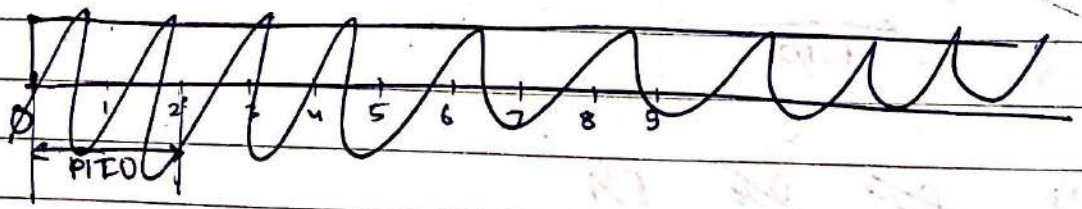
| PNo | AT | BT | IO BT | BT | Total BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | (3 | 2 | 2) | 5 | 11 | 11 | 6 | 5 |
| 2 | 0 | (2 | 4 | 1) | 3 | 7 | 7 | 4 | 0 |
| 3 | 2 | (1 | 3 | 2) | 3 | 9 | 7 | 4 | 0 |
| 4 | 5 | (2 | 2 | 1) | 3 | 16 | 11 | 8 | 6 |

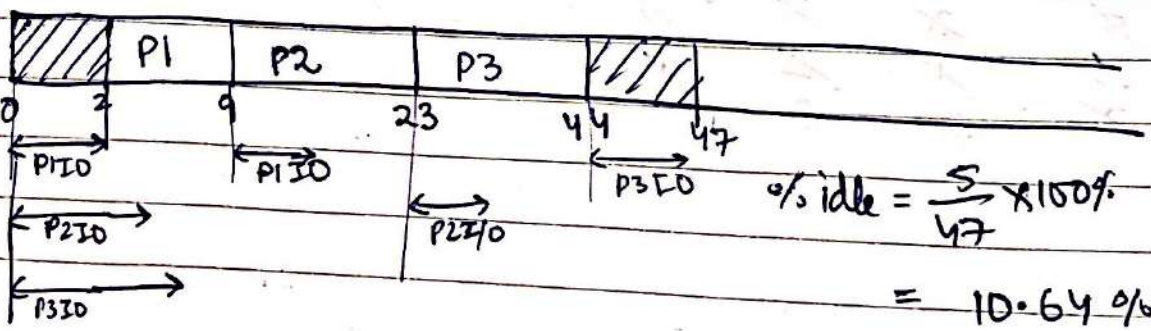



~~P1 P2 P3 P4~~
5 3 3 3
3 X 2 X
2 0 0 0

eg. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% time doing computation, and the last 10% time doing I/o again. The OS uses a SRTF algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume all I/O operations can be overlapped as much as possible. For what % of time does the CPU remains idle?

| PNo | AT | IOBT | BT | IOBT |
|---|---|---|---|---|
| P1 | 0 | 2 | 7 | 1 |
| P2 | 0 | 4 | 14 | 2 |
| P3 | 0 | 6 | 21 | 3 |



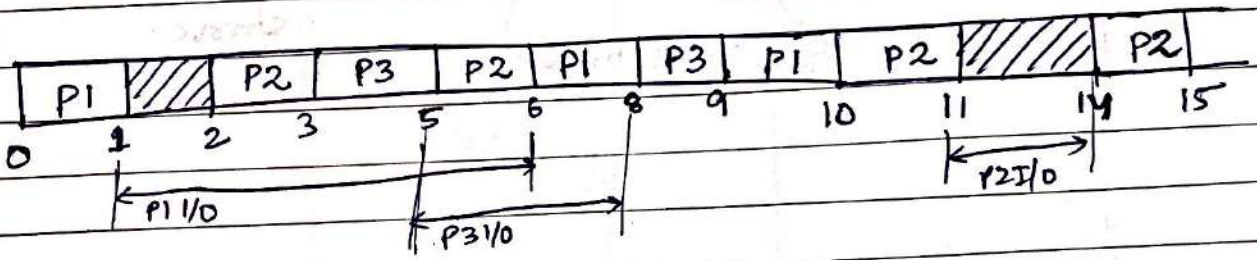

% idle = $\frac{5}{47} \times 100\%$

= 10.64 %

## Preemptive Priority with process contain CPU & I/O time :-

eg.

| PNO | AT | Priority | CPU | I/O | CPU | Total BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 2 | (1 | 5 | 3) | 4 | 10 | 10 | 6 | 0 |
| P2 | 2 | 3 (L) | (3 | 3 | 1) | 4 | 15 | 13 | 9 | 0 |
| P3 | 3 | 1 (H) | (2 | 3 | 1) | 3 | 9 | 6 | 3 | 0 |

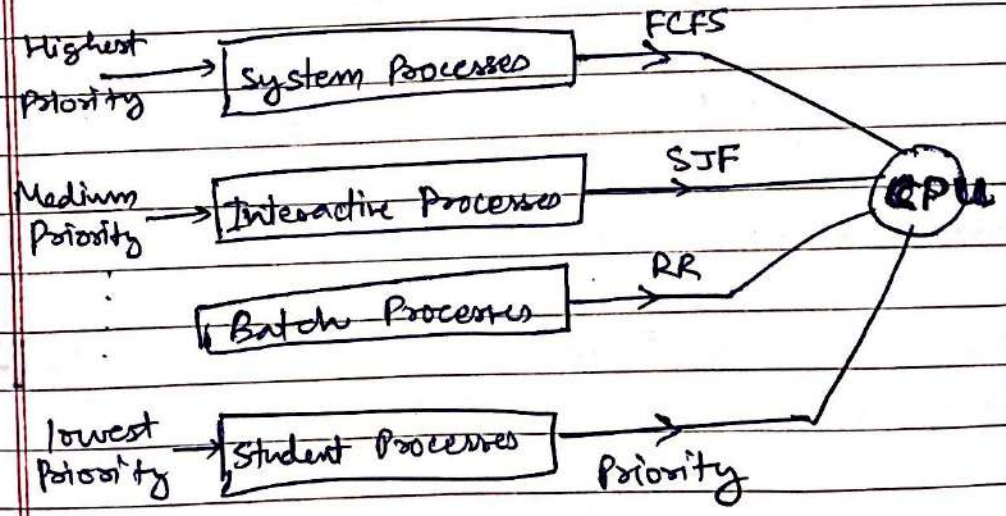



~~P1~~ ~~P2~~ ~~P3~~
~~4~~ ~~4~~ ~~6~~
~~3~~ ~~3~~ ~~4~~
~~2~~ ~~2~~ 2
~~1~~ ~~1~~ 1
0 0

$\eta = \frac{11}{15} \times 100\%$ (used)

$\%\ \text{Idle} = \frac{4}{15} \times 100\%$

## Multilevel Queue Scheduling

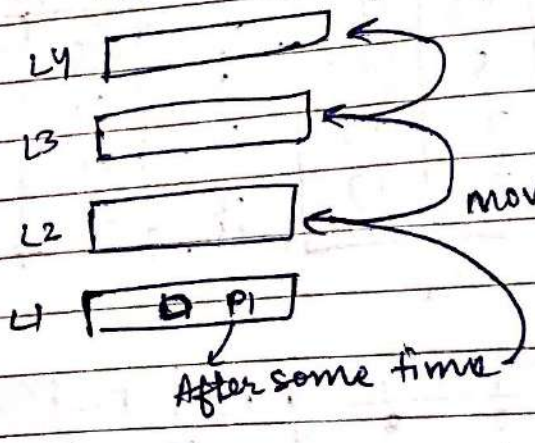



Disadv :- Starvation @ lower level queues

adv :- can apply diff. sch. algo to diff. queues

Solution of disadvantage (Feedback) – Feed back queues

* ML feedback scheduling



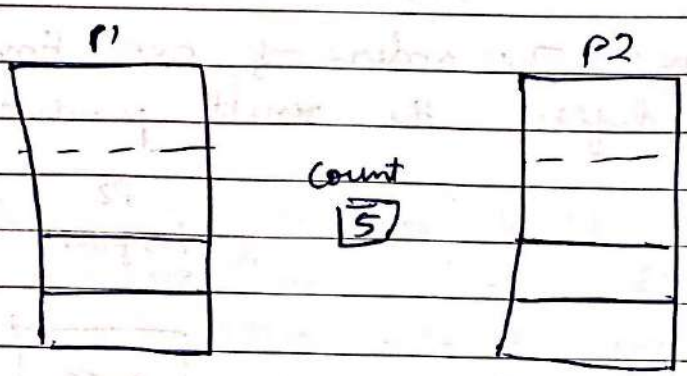

move it upwards.
so that it does not starve.

# 2. PROCESS SYNCHRONIZATION

Page 27

→ Process want to communicate or compete.
  ↳ using shared memory,
    ↳ data inconsistency is possible.

eg




```
P1                          P2
{                           {
...                         ...
count++;                    count--;
...                         ...
}                           }
```

| count++ | count-- |
|---|---|
| 1) mov count, R0 | 1) mov count, R1 |
| 2) inc R0 | 2) dec R1 |
| 3) mov R0, count | 3) mov R1, count |

Order 1 → P1 : 1, 2, 3 P2 : 1, 2, 3. count = 5

↓

without preemption → consistent

Order 2 → P1 : 1, 2 | P2 : 1, 2, 3 | P1 : 3 count = 6

↓

with preemption → inconsistent

Order 3 → P1 : 1, 2 | P2 : 1, 2 | P1 : 3 | P2 : 3 count = 4

↳ inconsistent again

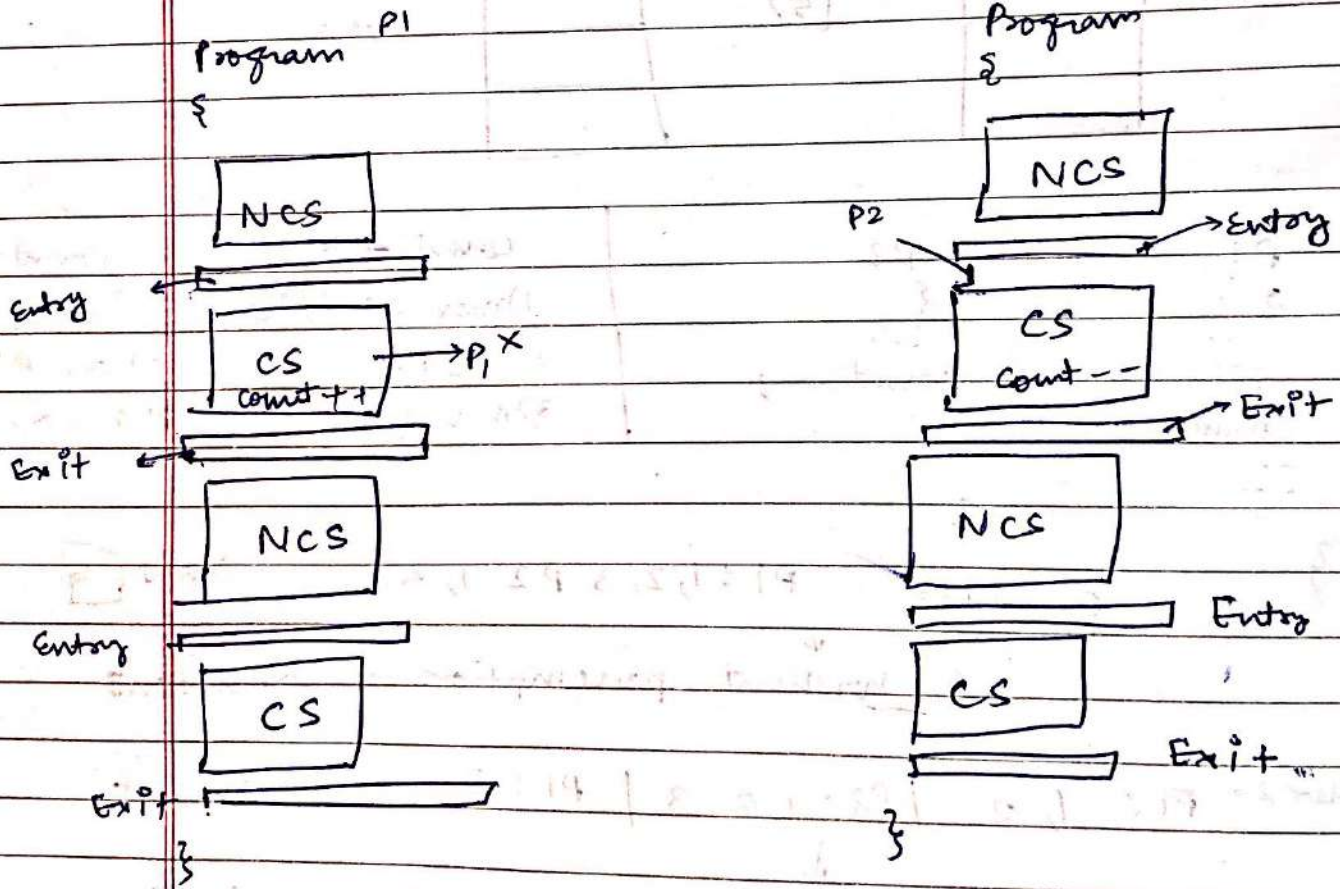
called 'Race condition' → avoided

↳

final output of order of execution

Critical section :- The part of code where the processes are using a common resource is called critical section. It is the part of the program where shared resources are accessed by processes

Race condition :- The order of execution of instructions defines the result produced.




Requirements of synchronization mechanisms :-

Primary :-
- Mutual Exclusion
- Progress

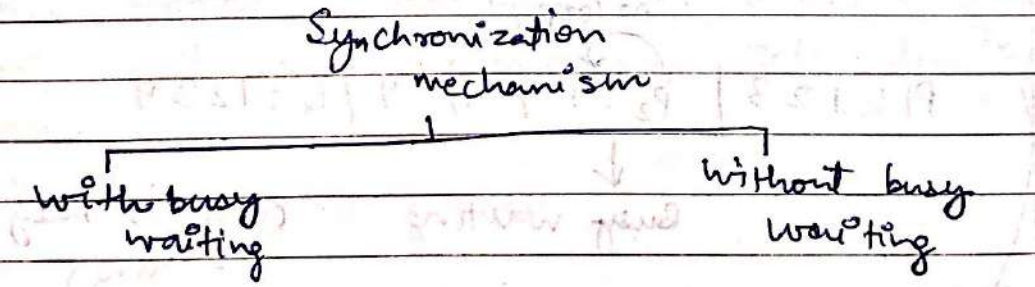
Secondary :-
- Bounded waiting
- Portability (or) architectural neutrality :

Mutual Exclusion :- If one process is in its critical section, then other process must not enter into its critical section.

Progress :- If a process does not want to get into critical section, then the process should not stop other processes from going into critical section.

Bounded waiting :- You should be able to say that ~~how~~ after how much finite amount of time, will other critical section get a chance to get into critical section if one process is in its critical section.

Potability or architectural neutrality :- Solution must not be dependent upon hardware or OS. (Able to run on all the platforms).



Lock variable :-

→ Software mechanism implemented in user mode
(user mode :- No support from OS
kernel mode :- Support from OS)

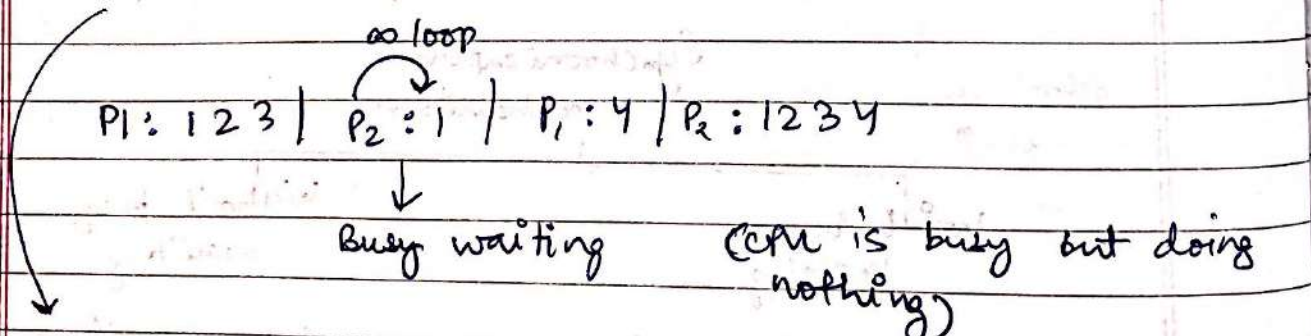
→ Busy waiting solution

→ Can be used even for more than two processes

LOCK = 0 - critical section is vacant
LOCK = 1 - critical section is occupied

entry section
```
1) while (LOCK != 0);
2) LOCK = 1;
```
3) CS

exit section
```
4) LOCK = 0
```




m/c code of this HL code :-

1) LOAD LOCK, R0
2) CMP R0, #0
3) JNZ Step 1
4) Store #1, LOCK
5) CS
6) Store #0, LOCK

Let initially LOCK = 0

$P_1$ : 1 | $P_2$ : 1, 2, 3, 4, CS | $P_1$ : 2, 3, 4, CS |

⚠ No mutual Exclusion

Every process can be in critical section. No mutual exclusion guaranteed.

1) Load LOCK, R0
2) CMP R0, #0
3) JNZ Step 1
4) Store #1, lock

Why does this do not work?

→ Once a process has loaded the value of lock and if gets preempted after step (1), (2) or (3) i.e. before storing it back, then some other process coming in middle and this process both simultaneously get inside critical section.

↓ To stop this to happen

Stop anyone to come in the middle

Soln:-
1) Load LOCK, R0
2) Store #1, LOCK
3) CMP R0, #0
4) Store #1, LOCK

} decreases the chance of someone coming in the middle

Still, error exist :-

P1 : 1 | P2 : 1, 2, 3, 4, CS | P1 : 2, 3, 4, CS |

No. mutual exclusion still
still the problem is not solved

solution

1) Load LOCK, R0
2) Store #1, LOCK
3) Cmp R0, #0
4) JNZ Step 1

→ Make these 2 operations a single atomic operation

TSL LOCK, R0

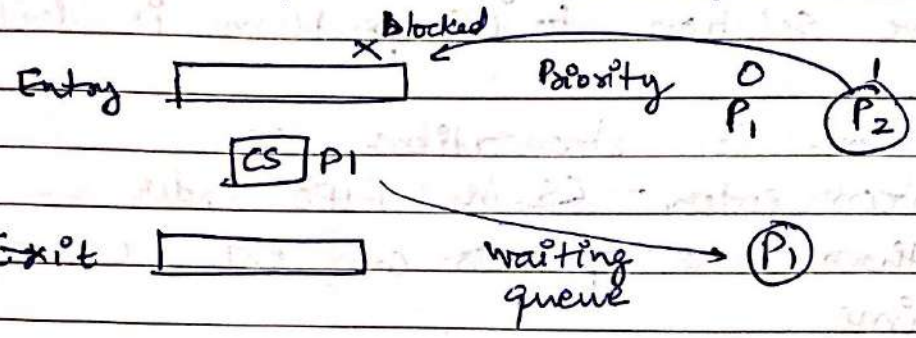
Test Set Lock

Single atomic instruction.
Solves the problem.
✓ Mutual exclusion guaranteed.

- Mutual Exclusion ✓
- Progress ✓
- Bounded waiting ✗ NOT guaranteed
- Architectural neutral ✗

Hardware must provide with "T.S.L" instruction.

one more problem :- Priority Inversion




CPU is with $P_2$ because of priority, but the critical section is with $P_1$ because of synchronization technique

CPU → P2
CS → P1

Deadlock like thing called "SPINLOCK"

Gate08 The enter_cs() and leave_cs() functions to implement critical section of a process are realised using test-and-set instruction as follows :-

```
void enter_cs(x)
{
  while (test-and-set(x));
}
void leave_cs(x)
{
  x=0;
}
```

'x' is initialised to '0'.

Which of the following is true :-

a) The above solution to CS problem is deadlock free
b) The solution is starvation free
c) The process enter CS in FIFO order
d) more than one process can enter CS at the same time.

Ans:- ~~(a)~~ (a) ~~(b)~~ ~~(c)~~ (A)

Gate12 Important! Fetch_and_add (X,i) is an atomic Read-modify-write instruction that reads the value of memory location X, increments it by value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value 0 corresponds to lock being available, while any non-zero value corresponds to lock being not available.

```
Acquire Lock (L) {
    while (Fetch-And-Add (L,1))
        L=1;
}
Release Lock (L) {
    L=0;
}
```

This implementation

A) Fails as L can overflow.

B) Fails as L can take on a non-zero value when the lock is actually available.

C) Works correctly but may starve some processes

D) works correctly without starvation.

Ans. A, B

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | ... | $P_n$ |
|---|---|---|---|---|---|
| L=0 | L=1 | L=2 | L=3 | | L=n-1 |
| L=1 | L=2 | L=3 | L=4 | | L=0 → overflow |
| CS | | | | | |

$L = \cancel{0} \to \cancel{0} \to 1$

1. while
2. L=1
3. [CS]
4. L=0;

$P_0$: 1, 2, CS | $P_1$: 1 | $P_0$: 3 | $P_1$: 2

$R_0 = 0$ $R_0 = 1$

↓

sets L=1 incorrectly and No process enters CS now.

Gate10. Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared boolean variables S1 and S2 are randomly assigned.

| Method used by P1 | Method used by P2 |
|---|---|
| while (S1 == S2); | while (S1 != S2); |
| Critical section | critical section |
| S1 = S2; | S2 = not (S1); |

Which one of the following statements describes the properties achieved?

A) Mutual exclusion but not progress
B) Progress but not Mutual exclusion
C) Neither mutual exclusion nor progress
D) Both mutual exclusion & progress

Ans: (A).

## Disabling Interrupt:-

Disable int
CS
Enable int
} allows only one process to be inside critical section

- Mutual Exclusion ✓
- Progress ✓
- Bounded waiting ✗ Not guaranteed
- Architectural neutrality ✗ → architecture must provide interrupt control.

Problem :- You cannot give the interrupt control to user process. (he might not give it back to OS). Thus it is rarely used method.

## Strict alternation approach OR Turn variable :-

→ Software mechanism implemented at user mode
→ Busy waiting solution
→ 2-process solution $(P_0, P_1)$ or $(P_i, P_j)$

→ Mutual Exclusion ✓
→ Progress ✗
→ Bounded waiting ✓ → only one time go & pass the key
→ Architectural neutrality ✓ → no restrictions on platform

For Process $P_0$ :- (int turn) = 0

```
Non CS
while (turn != 0);
CS
turn = 1;
Non CS
```

0 → $P_0$
1 → $P_1$

For process $P_1$ :-

```
Non CS
while (turn != 1);
CS
turn = 0;
Non CS
```

## Interested variable

Interested [0] = T — $P_0$
Interested [1] = T — $P_0$

$P_0$

```
non CS
Entry  Int[0] = T;
       while (int[1] == T);
CS
Exit   Int[0] = F;
```

$P_1$

```
nonCS
Entry  Int[1] = T;
       while (int[0] == T);
CS
Exit   Int[1] = F;
```

• Deadlock ✓ → BW ✗

→ • mutual Exclusion ✓
→ progress ✓
→ bounded waiting ✗
→ Architectural neutrality ✓

which has passed while() first

| $I_0$ | $I_1$ | |
|---|---|---|
| T | T | — ? $P_0/P_1$ |
| T | F | — $P_0$ |
| F | T | — $P_1$ |
| F | F | — ✗ |

| $P_0$ | $P_1$ | $P_0$ | $P_1$ | P2 |
|---|---|---|---|---|
| $I_0 = T$ | $I_1 = T$ | $I_0 = F$ | W2 | W2 |
| CS | W2 | $I_0 = T$ | | |
| | | W2 | | |

Deadlock

## Peterson's Method :-

→ S/W mechanism at user mode
→ Busy waiting solution
→ 2 process solution
→ It uses (Turn + Interested) variable.

Algorithm :-

• Progress ✓
• Mutual Exclusion ✓
• Acc. Neutral ✓
• Bounded wait ✓

```
# define N 2
# define TRUE 1
# define FALSE 0
int interested[N] = FALSE;
int turn;
void Entry_Section(int process)
{
1) int other;
```

2) other = 1 - process;
3) interested [process] = TRUE;
4) turn = process;
5) while (interested [other] == TRUE && TURN == process);
}

void Exit_Section (int process)
{
6) Interested [process] = FALSE;
}

Entry section
CS
Exit Section

Tracing Peterson's solution :-

Interested • 0 F T F  1 F T

Turn 0 1

✓ P0 : 1 2 3 4 5 CS | P1 : 1 2 3 4 5 | P0 : 6 | P1 : 5 CS 6

int : 0 F  1 T  Turn 1 0

✓ P0 : 1 2 3 | P1 : 1 2 3 4 5 | P0 : 4 5 | P1 : 5 CS 6 |

P0 : 5 CS 6

→ The process which executes the line (4) first will enter CS First.

int [T] (0) [T] (1) Turn [0] 1

$P_0$: 1 2 3 | $P_1$: 1 2 3 | $P_0$: 4 5 ↺ | $P_1$: 4 5 ↺

$P_0$: 5 CS 6 | $P_1$: 5 CS 6

→ Mutual Exclusion ✓
→ Progress ✓
→ Bounded waiting ✓
→ Arch. Neutrality ✓

## Synchronization Mechanisms without Busy waiting :-

Busy waiting → Priority inversion "surely"
↳ SPINLOCK

## Sleep & Wake :

[Producer / Consumer problem]

```
# define N 100          // slots in buffer
# define count 0        // items in buffer
void producer (void)
{
    int item;
    while (TRUE)
    { item = produce_item();
      if (count == N) sleep();
      insert_item (item);
      count++;
      if (count == 1) wakeup (consumer);
    }
}
```

```
void consumer (void)
{
  int item;
  while (TRUE)
  {
    if (count == 0) sleep ();
    item = remove_item ();
    count --;
    if (count == N-1)
        wakeup (producer);
    consume_item (item);
  }
}
```

If preemption of consumer procers occurs here, than both of them can sleep thinking that other one will wake them up; causing infinite sleep for both. "DEADLOCK"

## Semaphores

→ Variables on which read, modify and update happens atomically in Kernel mode. (no preemption).

→ There are two types:-
1. Counting Semaphores.
2. Binary Semaphoies (Mutexes).

## Counting Semaphores :-

→ When a resource is allowed to be shared by some particular number of processes but not more than that, then counting semaphores are used.

```
Struct semaphore
{
    int value ;      // # proces that can enter CS
    queue type L;
}
Down (semaphore S)
{
  S.value = S.value - 1;
  if (S.value < 0)
  {
     put process (PCB) in L;
     sleep ();
  }
  else
     return;
}
Up (semaphore S)
{
  S.value = S.value + 1;
  if (S.value ≤ 0)
  {  select a process from L;
     wakeup ();
  }
}
```

→ For mutual exclusion, use initial value = 1.
→ Bounded waiting guaranteed because of Queue in semaphore
→ Progress is guaranteed.
→ Arch. Neutrality ✗ because operates in kernel mode

| Decrease | Increase |
|---|---|
| Down | up |
| P | V |
| wait | Signal |

} any terminology can be used

used by Dijstra in research paper in 1965.

Gate 98. A counting semaphore was initialized to 10. Then 6P and 4V operation were computed on this semaphore. what is the result?

Sol. value = 8 list :- 2 processes.

Gate 92 S=7, then 20P, 15V, S=?

Sol. S = 7 - 20 + 15 = 2

## Binary Semaphores :-

```
struct Bsemaphore
{
  enum value (0, 1);
  Queue type L;
}
```

'L' contains all PCBs corresponding to processes got blocked while performing down operation unsuccessfully.

```
struct Bsemaphore
{
   enum value (0, 1)
   Queue type L;
}
Down (BSemaphore S)
{
   if (S. value == 1)
        S. value = 0;
   else
   {
      put the process (PCB) in S. L;
      sleep ();
   }
}
up (BSemaphore S)
{
   if (S.L is empty)
      S. value = 1;
   else
   {
      Select a process from S.L;
      wakeup ();
   }
}
```

eg. • mutex can also be used to order the processes. Even stop the scheduler and use your order.

```
mutex a, b ;   a=1, b=0;

P0                          P1
while (true)                while (true)
{                           {
   P(a);                       P(b);
   Print ("1");                print ("0");
   V(b);                       V(a);
}                           }
```

sol. only order possible :- 101010...

↳ implemented strict alternation through mutexes.

eg.

```
mutex a, b;   a=1, b=1;

P0                          P1
while (true)                while (true)
{                           {
1. P(a);                    5. P(a);
2. P(b);                    6. P(b);
   <CS>                        <CS>
3. V(a);                    7. V(a);
4. V(b);                    8. V(b);
}                           }
```

P0 : 1 | P1 : 5 ↓ B | P0 : 2 CS 3 | P1 : 6 ↓ B | P0 : 4 | P1 6 CS 7 8 |

order of these instructions describe the behaviour of the mutexes

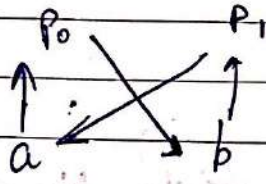
eg:

| $P_0$ | $P_1$ |
|---|---|
| while (true) | while (true) |
| { | { |
| 1. P(a); | 5. P(b) |
| 2. P(b); | 6. P(a) |
| <CS> | <CS> |
| 3. V(a); | 7. V(a); |
| 4. V(b); | 8. V(b); |
| } | } |

$P_0$ : 1 | $P_1$ : 5 | $P_0$ : 2 | $P_1$ : 6

B B

Deadlock




Implemented through mutexes:

Gate 13: Three concurrent process X, Y and Z execute three different code segments that access and update certain shared variable. Process X executes the P operation (i.e. wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores ~~b, c and d~~; c, d and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the

V operation (i.e. signal) on its three semaphores. All semaphores are binary semaphores initialized to 1. Which one of the following represents a deadlock-free order of invoking the P operations by the processes?

(A) X: P(a) P(b) P(c) Y: P(b) P(c) P(d) Z: P(c) P(d) P(a)

✓(B) X: P(b) P(a) P(c) Y: P(b) P(c) P(d) Z: P(a) P(c) P(d)

(C) X: P(b) P(a) P(c) Y: P(c) P(b) P(d) Z: P(a) P(c) P(d)

(D) X: P(a) P(b) P(c) Y: P(c) P(b) P(d) Z: P(c) P(d) P(a)

Sol: X: P(a) | Y: P(b) | Z: P(c) P(d) P(a) | X: P(b) | Y: P(c)

(b) ~~X:~~ Y: P(b) ✓

(c) X: P(b) | Y: P(c) P(b) | X: P(a) P(c) X

(d) X: P(a) P(b) | Y: P(c) P(b) | X: P(c) X

Gate 00 Let m[0] .. m[4] be mutexes (binary semaphores) ~~for~~ and $P_0 \dots P_4$ be processes. Suppose each process P[i] executes the following :-

wait (m[i]) ; wait (m[(i+1] mod 4))

CS

release (m[i]) ; release (m[(i+1) mod 4])

This could cause :-

a) Thrashing b) ✓ Deadlock

c) Starvation but NO DL d) None of these.

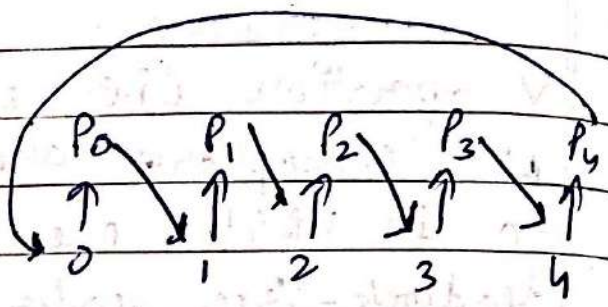
Sol:- $P_0$ : P(m[0]) P(m[1])

$P_1$ : P(m[1]) P(m[2])

$P_2$ : P(m[2]) P(m[3])

$P_3$ : P(m[3]) P(m[4])

$P_4$ : P(m[4]) P(m[0])




$P_0 \to P_1 \to P_2 \to P_3 \to P_4 \to P_0$

deadlock

B

Gate96 $P_i$ : 0, 1, 2, 3 & $M_i$ : 0, 1, 2, 3

$P_i$ : P($M_i$), P($M_{(i+1)}$ mod 4)

CS

V($M_i$), V($M_{(i+1)}$ mod 4)

$P_0$ : P($m_0$) P($m_1$)

$P_1$ : P($m_1$) P($m_2$)

$P_2$ : P($m_2$) P($m_3$)

$P_3$ : P($m_3$) P($m_0$)

} deadlock

but

$P_0$ : P($m_0$) P($m_1$)

$P_1$ : P($m_1$) P($m_2$)

$P_2$ : P($m_2$) P($m_3$)

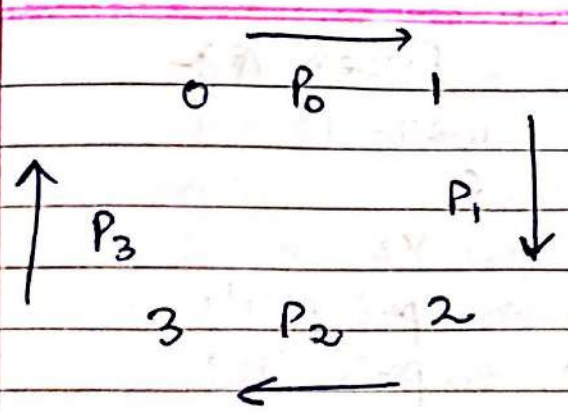
$P_3$ : P($m_3$) P($m_0$)

$P_0$ : P($m_0$) | $P_1$ : P($m_1$) | $P_2$ : P($m_2$) |

$P_3$ : P($m_0$) | $P_0$ : P($m_1$) |

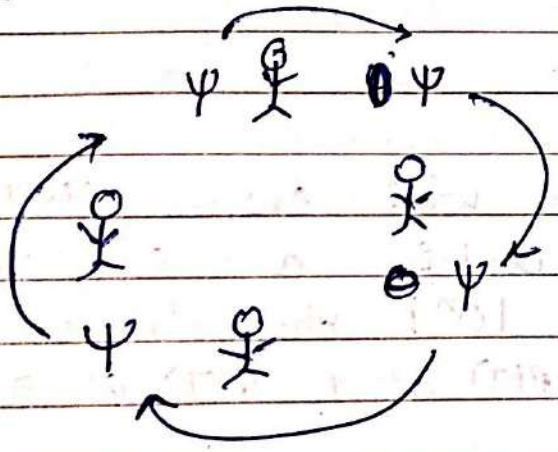
$P_1$ : P($m_2$) | $P_2$ : P($m_3$) CS

No deadlock




Change order of just one process to remove the deadlock.

Dining philosopher's problem :-



Same problem of above process. If all philosopher's take the spoons in same order, they all will have one spoon each.

Solution :- Change order of taking of spoons of just one philosopher → Deadlock breaked.

Ques 03. :- S and T are mutexes synchronization statements can be inserted only at w, x, y and z

Q1 Which of the following will lead to an o/p starting with 0011 0011 0011... ?

a) P(S) at w, V(S) at X, P(T) at Y, V(T) at Z, S & T init '1'.
b) P(S) at w, V(T) at X, P(T) at Y, V(S) at Z, S at '1', T at '0'.
c) P(S) at w, V(T) at X, P(T) at Y, V(S) at Z, S & T at '1'
d) P(S) at w, V(S) at X, P(T) at Y, V(T) at Z, S=1, T=0.

Process P :-

```
while (1)
{
  w:
  print '0';
  print '0';
  x:
}
```

Process Q :-

```
while (1)
{
  y:
  print '1';
  print '1';
  z:
}
```

Ans. (B)

Q.2 which of the following will ensure that output will never contain a substring of form $01^n0$ or $10^n1$ where 'n' is odd.

(a) P(S) at w, V(S) at x, P(T) at y, V(T) at z
S & T initially 1.

(b) P(S) at w, V(T) at x, P(T) at y, V(S) at z
S & T initially 1.

(c) P(S) at w, V(S) at x, P(S) at y, V(S) at z
S initially 1.

(d) V(S) at w, V(T) at x, P(S) at y, P(T) at z
S & T initially 1.

Ans :- (c)

Gate06

```
void barrier
{
  1. P(S);
  2. process_arrived ++;
  3. V(S);
```

```
4. while (process_arrived != 3);
5.  P(s);
6.  process_left ++;
7.  if (process_left == 3)
8.  {   process_arrived = 0;
9.      process_left = 0;   }
10. V(s);
    }
```

Q.1 This implementation of barrier is incorrect. Which of the following is true?

a) Barrier implementation is wrong due to the use of binary semaphore 's'.

b) The barrier imp. may lead to deadlock if two barrier invocations are used in immediate succession.

c) Lines 6 to 10 need to be inside a CS.

d)

Ans:- (B)

Q.2 which of the following rectifies the problem in implementation

(A) Lines 6-10 are simply replaced by process_arrived--

(B) At the begin of the barrier, the 1st process to enter has to wait until process_arrived becomes '0' before P(s).

(C) Context switch is disabled at the beginning of the barrier and reanabled at end.

(D) The variable process_left is made private instead to shared

Ans:- (B)

Gate 08. The P and V operations on counting semaphores, where 'S' is a counting semaphore are defined as follows :-

P(S) : S = S - 1;
if S < 0 then wait;

V(S) : S = S + 1;
if S ≤ 0 then wake up a process waiting on S;

Assume $P_b$ and $V_b$ are wait & signal operations on binary semaphores provided. Two binary semaphores $X_b$ and $Y_b$ are used to implement the semaphore operations P(S) and V(S) as follows :-

```
P(S) : Pb(Xb);
       S = S - 1;
       if (S<0) {
         Vb(Xb);
         Pb(Yb);
       }
       else Vb(Xb);
```

```
V(S) : Pb(Xb);
       S = S + 1;
       if (S<=0) Vb(Yb);
       Vb(Xb);
```

The initial values of $X_b$ and $Y_b$ are :-

(a) 0, 0 (b) 0, 1 (c) 1, 0 (d) 1, 1

Ans:- (c)

Gate10 The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0

| Process P0 | Process P1 | Process P2 |
|---|---|---|
| while (1) {<br>wait (S0);<br>print '0';<br>release (S1);<br>release (S2);<br>} | wait (S1);<br>release (S0); | wait (S2);<br>release (S0); |

How many times will P0 print '0'?

a) Atleast twice
b) Exactly twice
c) Exactly thrice
d) Exactly once

Sol:-

X, Ø, Ø
Ø X X 1
X Ø Ø
0 1

000..

A

~~Ø~~ X, Ø, Ø
Ø X X
X Ø Ø
0 1 1

Gate13 Q.34

S = 2
X = 0

| W | X | Y | Z |
|---|---|---|---|
| P(S) | P(S) | P(S) | P(S) |
| X = X + 1 | X = X + 1 | X = X - 2 | X = X - 2 |
| V(S) | V(S) | V(S) | V(S) |

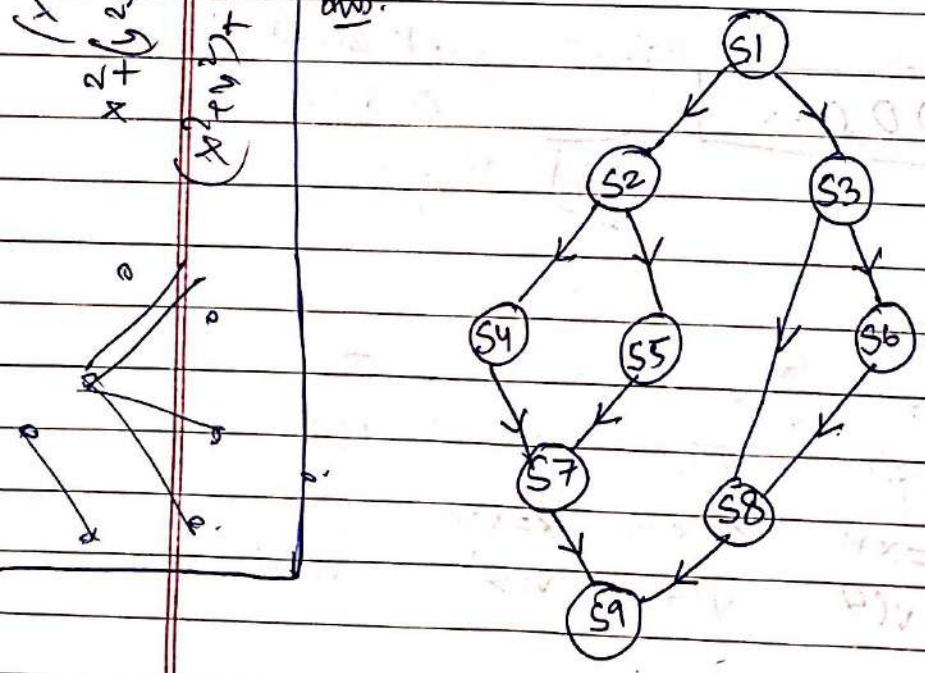
2

B

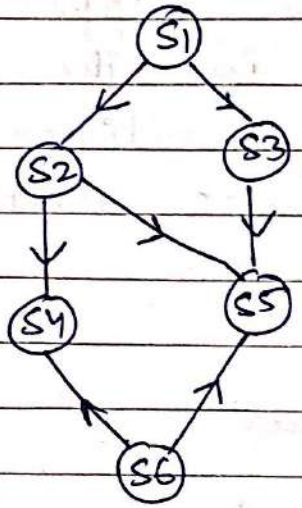
Gate95. Draw the precedence graph for statements S1 to S9

```
var : a,b,c,d,e,f,g,h,i,j,k : semaphore;
begin
cobegin
    begin : S1; V(a) ; V(b) end
    begin : P(a) ; S2 ; V(c); V(d); end
    begin : P(c) ; S4 ; V(e) ; end
    begin : P(d) ; S5 ; V(f) ; end
    begin : P(e) ; P(f) ; S7 ; V(k); end
    begin : P(b) ; S3 ; V(g) ; V(h) end
    begin : P(g) ; S6 ; V(i); end
    begin : P(h) ; P(i) S8 V(j) end.
    begin : P(j) ; P(k) ; S9 end
coend
end
```

two:-



ate93 Write the program for this precedence graph




```
begin
cobegin
begin :     S1 V(a) V(b) end
begin :   P(a) S2 V(c) V(d) end
begin :   P(b) S3 V(e) end
begin :   P(c) P(d) S4 end
begin :   ~~P(d) P(e) S5 end~~ P(d) P(e) P(f) S5 end
begin :     S6 V(f) V(g) end
coend
end
```

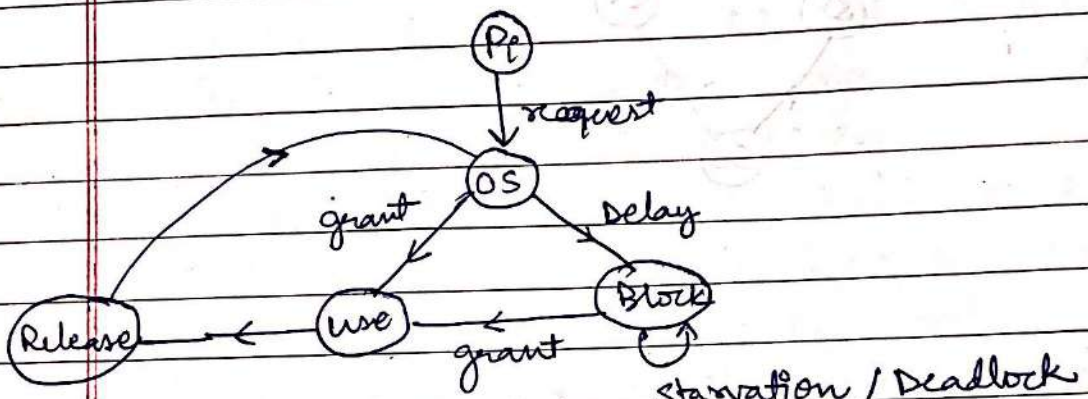
# 3. Deadlocks

Deadlock :-

→ A set of processes are said to be in deadlock, if they wait for happening of an event caused by others in the same set.

→ Starvation is long waiting.

→ Deadlock is infinite waiting.



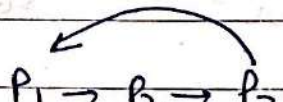

Necessary conditions for DL :-

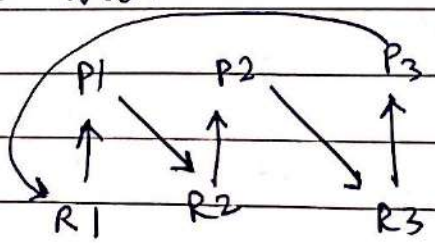
→ Mutual exclusion

→ Hold and wait

→ No preemption

→ Circular wait




eg. A system is having 3 user processes each requiring 2 units of resource 'R'. The minimum number of units of 'R' such that no deadlock occur is :-

a) 3 b) 5 c) 4 d) 6

Date Page 57

Sol:- $P_1$ $P_2$ $P_3$

"2" 1 1 Deadlock

"3" 1 1 1 Deadlock

"4" 1 1 1 (1) No deadlock [C]

Finish

eg. $P_1$ $P_2$ $P_3$ # Res. req no deadlock?

#Res. 2 3 4

Sol. $P_1$ $P_2$ $P_3$

1 1 1 (1) [7]

1 1

1

eg. $P_1$ $P_2$ ... $P_n$ # Res req no deadlock?

$x_1$ $x_2$ $x_n$

Sol. $P_1$ $P_2$ ... $P_3$

$x_1 - 1$ $x_2 - 1$ $x_n - 1$

$$\sum_{i=1}^{n} x_i - n + 1$$

eg. There are 6 instances of a resource, and every process $P_i$ requires 2 instances of that resource to finish its execution, then how many process can be present at maximum so that there will be no deadlock.

Sol.

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | | → deadlock |

5 Processes :- No deadlock

eg. 6 instances, $P_i$ = 3 instances, max # P = ?
(no deadlock)

Sol.

| $P_1$ | $P_2$ | $P_3$ | |
|---|---|---|---|
| 1 | 1 | 1 | → deadlock |
| 1 | 1 | 1 | |

2 process :- No deadlock.

eg. 100 instances, $P_i$ = 2 instances, max # P = ?

Ans :- 99

eg. 100 instances, $P_i$ = 3 instances, max # P = ?

Sol. $n \times 2 = 100$ $n = 50$

49

eg: 100 instances, $P_i$ = 4 instances, max # P = ?

Sol: $P_1$ $P_2$ ... $P_{33}$
3 3 3 1 → 33

eg: 10 process, each process requires 3 instances, min. # resources such that no deadlock?

Sol: $P_1$ $P_2$ ... $P_{10}$
2 2 2 → 21

Gate 92: A computer system has 6 tape drives, with n processes competing for them. Each process needs 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock free is :-

a) 2 b) 3 c) 4 d) 1

Sol: $P_1$ $P_2$ $P_3$
2 2 2 → 2 → (A).

Gate 93: Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of 'm' deadlock will not occur?

a) 7 b) 9 c) 10 d) 13

Sol: A B C
2 3 5 → 10 → min 11 → D

Gate 05 Suppose 'n' processes $P_1, P_2, \dots P_n$ share 'm' identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process $P_i$ is $S_i$ where $S_i > 0$. Which one of the following is a sufficient for ensuring that deadlock does not occur:-

a) $\forall i,\ S_i < m$

b) $\forall i,\ S_i < n$

c) $\sum_{i=1}^{n} S_i < (m+n)$

d) $\sum_{i=1}^{n} S_i < (m \times n)$

Sol:

| $P_1$ | $P_2$ | ... | $P_n$ | |
|---|---|---|---|---|
| $S_1-1$ | $S_2-1$ | | $S_n-1$ | 1 |

$$m \geq \sum S_i - n$$

$$m+n > \sum S_i$$

(c)

Gate 06. Consider the following snapshot of a system running 'n' processes. Process 'i' is holding $x_i$ instances of a resource 'R' for $1 \leq i \leq n$ currently all instances of 'R' are occupied. Further, for all 'i', process 'i' has placed a request for an additional $y_i$ instances it already has. There are exactly two processes 'p' and 'q' such that $y_p = y_q = 0$. Which of the following can serve as necessary condition to guarantee that the system is not approaching a deadlock?

a) $\min(x_p, x_q) < \max_{k \neq p,q} y_k$

b) $x_p + x_q \geq \min_{k \neq p,q} y_k$

c) $\max(x_p, x_q) > 1$ d) $\min(x_p, x_q) > 1$

Sol:

| $P_1$ | $P_2$ | ... | $P_i$ | ... | $P_p$ | ... | $P_q$ |
|---|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | | $x_i$ | | $x_p$ | | $x_q$ |
| $y_1$ | $y_2$ | | $y_p$ | | 0 | | 0 |

$$x_p + x_q \geq \min_{k \neq p,q} y_k$$ [B]

## Strategies for handling Deadlock :-

** 1) Deadlock ignorance (Ostrich approach)
2) Deadlock prevention
3) Deadlock avoidance
* 4) Deadlock detection and recovery

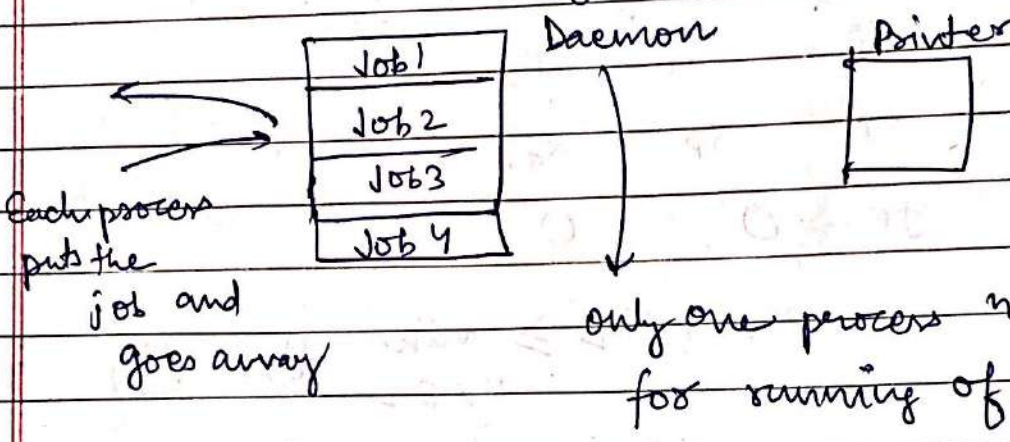
## Deadlock prevention :-

| Condition | Approach |
|---|---|
| Mutual exclusion | Spool everything |
| Hold and wait | Request all resources initially |
| No preemption | Take resources away |
| Circular wait | order resources numerically |

- Spool everything :- Allowing all the resources to give permission to all process concurrently may cause chaos. eg. printer

  ↳ one possible solution :-

  Spooling




problem :- Even this is not a feasible solution for all the resources. (Processes will not compete for this printer, instead will compete for this memory).

- Prevent Hold & wait :- Request all resources initially.

  !(Hold and wait) = !Hold or !wait

  ↙ Before requesting a new resource, release all previously held resources.

  ↓ Request all resources initially

Problem :- Process cannot say before execution that what are resources it might need before execution.

- No preemption :- Take resources away

Problem :- If we pull out printer away during execution, work done by previous process will be wasted.

- Solution to circular wait :- Order resources numerically
  > Ask for resources always in ascending order

$R_1$ $R_2$ $R_3$ $R_4$ ...

$P_1$ then next $P_1 \rightarrow R_1$ not possible
$P_1 \rightarrow R_4$ or $R_3$ possible

If every process ask for the resources in increasing order, no deadlock will occur.

> Only solution which can be implemented practically

## Deadlock Avoidance :-

[Safe & Unsafe Deadlock Avoidance] :-

$ Banker's Algorithm :-

Consider the following system configuration

(At some instance of time, a system is having these many processes) :-

eg.

| Process | Tape drivers | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| A | 3 | 0 | 1 | 1 |
| B | 0 | 1 | 0 | 0 |
| C | 1 | 1 | 1 | 0 |
| D | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 |

Resources assigned

| Process | Tape drivers | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 1 | 2 |
| C | 3 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 0 |
| E | 2 | 1 | 1 | 0 |

Resources still needed.

E = (6 3 4 2) Resources available in total
P = (5 3 2 2) Resources assigned currently
A = (1 0 2 0) Resources available currently

- Safe state:- A state is said to be safe if there exist a sequence in which you could satisfy the need of all processes in some order without getting involved in deadlock.

Step 1:- available 1 0 2 0
↑
need of A: 1 1 0 0 ✗ not possible

need of D: 0 0 1 0 ✓ can be satisfied.
↳ D will release all resources it was holding

Step 2:- available A = 1 0 2 0
(D) 1 1 0 1
A = 2 1 2 1
(A) 3 0 1 1
A = 5 1 3 2

A = 5 1 3 2
(B) 0 1 0 0
A = 5 2 3 2
(C) 1 1 1 0
A = 6 3 4 2

Last Step:-

| | | | | |
|---|---|---|---|---|
| A = | 6 | 3 | 4 | 2 |
| (E) | 0 | 0 | 0 | 0 |
| A = | 6 | 3 | 4 | 2 |

D A B C E → in this order needs of all process are satisfied without deadlock

Thus it is a "safe state"

eg. State is given in previous eg, now OS has get a new request from B = (0010), Now OS will check if the state after allocating these resources is safe OR NOT :

Sol: New resource assigned & resource needed matrix :-

| | | | | |
|---|---|---|---|---|
| A | 3 | 0 | 1 | 1 |
| B | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 1 | 0 |
| D | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 |

res. and.

| | | | | |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 0 | 2 |
| C | 3 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 0 |
| E | 2 | 1 | 1 | 0 |

res. needed.

E = (6 3 4 2) P = (5 3 3 2) A = (1 0 1 0)

D: 1 0 0 1 0

| | | | | |
|---|---|---|---|---|
| | 1 | 0 | 1 | 0 |
| (D) | 1 | 1 | 0 | 1 |
| | 2 | 1 | 1 | 1 |
| (A) | 1 | 1 | 0 | 0 |
| | 3 | 2 | 1 | 1 |
| (C) | 1 | 1 | 1 | 0 |
| | 4 | 3 | 2 | 1 |

| | | | | |
|---|---|---|---|---|
| | 1 | 0 | 1 | 0 |
| (D) | 1 | 1 | 0 | 1 |
| | 2 | 1 | 1 | 1 |
| (A) | 3 | 0 | 1 | 1 |
| | 5 | 1 | 2 | 2 |
| (B) | 0 | 1 | 1 | 0 |
| | 5 | 2 | 3 | 2 |

| | | | | |
|---|---|---|---|---|
| | 5 | 2 | 3 | 2 |
| (C) | 1 | 1 | 1 | 0 |
| | 6 | 3 | 4 | 2 |
| (E) | 0 | 0 | 0 | 0 |
| | 6 | 3 | 4 | 2 |

D A B C E → safe state ✓

After allocating the resource to (B), still we are in safe state. Thus OS will allocate the resource.

eg: In the state of previous eg, E request for (0 0 1 0). check whether OS will grant the resources or not.

Sol: New matrices:-

| | | | | |
|---|---|---|---|---|
| A | 3 | 0 | 1 | 1 |
| B | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 1 | 0 |
| D | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 1 | 0 |

resource allocated

| | | | | |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 0 | 2 |
| C | 3 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 0 |
| E | 2 | 1 | 0 | 0 |

resources still needed

E = (6 3 4 2) P = (5 3 4 2) A = (1 0 0 0)

1 0 0 0

No need can be completed

Thus, ~~NOT~~ UNSAFE state

∴ OS will NOT allocate resource to "E".

Gate07

| | X | Y | Z |
|---|---|---|---|
| $P_0$ | 1 | 2 | 1 |
| $P_1$ | 2 | 0 | 1 |
| $P_2$ | 2 | 2 | 1 |

alloc.

| | X | Y | Z |
|---|---|---|---|
| $P_0$ | 1 | 0 | 3 |
| $P_1$ | 0 | 1 | 2 |
| $P_2$ | 1 | 2 | 0 |

request

There are 5 units of each resource type.

Sol. $T = (5\ 5\ 5)$

$Al = (5\ 4\ 3)$

$Av = (0\ 1\ 2)$

| | 0 | 1 | 2 |
|---|---|---|---|
| $P_1$ | 2 | 0 | 1 |
| | 2 | 1 | 3 |
| $P_0$ | 1 | 2 | 1 |
| | 3 | 3 | 4 |

| | 3 | 3 | 4 |
|---|---|---|---|
| $P_2$ | 2 | 2 | 1 |
| | 5 | 5 | 5 |

$P_1\ P_0\ P_2$

"Safe state"

Gate 11.

| | Allocated | maximum | Future need |
|---|---|---|---|
| PA | 1 0 2 1 1 | 1 1 2 1 3 | 0 1 0 0 2 |
| PB | 2 0 1 1 0 | 2 2 2 1 0 | 0 2 1 0 0 |
| PC | 1 1 0 1 1 | 2 1 3 1 1 | 1 0 3 0 0 |
| PD | 1 1 1 1 0 | 1 1 2 2 0 | 0 0 1 1 0 ✓ |

AV = 00X11

Q. what is the smallest value of 'x' for which this is safe state?

Sol.

00X11

take X = 1

X [illegible] =

0 0 2 1 1

(B) 1 1 1 | 1 0

1 1 3 2 1

(C) 1 0 0 0 1 0 1

2 2 3 3 3

B, A ✓

$X_{min} = \boxed{2}$

Gate 14. (Set-D)

Q. 31

| | Alloc | | | Max | | | Future Need | | |
|---|---|---|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z | X | Y | Z |
| $P_0$ | 0 | 0 | 1 | 8 | 4 | 3 | 8 | 4 | 2 |
| $P_1$ | 3 | 2 | 0 | 6 | 2 | 0 | 3 | 0 | 0 |
| $P_2$ | 2 | 1 | 1 | 3 | 3 | 3 | 1 | 2 | 2 |

AV = 3 2 2

Req 1

| | Alloc | | | Future Need | | |
|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z |
| $P_0$ | 0 | 0 | 3 | 8 | 4 | 0 |
| $P_1$ | 3 | 2 | 0 | 3 | 0 | 0 |
| $P_2$ | 2 | 1 | 1 | 1 | 2 | 2 |

3 2 0

($P_2$) 2 1 1

5 3 3

(11) 3 2 0

8 5 3

($P_0$) 0 0 3

8 5 6

"REQ 1"

"Not" permitted ✓

Req 2

| | Alloc | | | Future Need | | |
|---|---|---|---|---|---|---|
| | X | Y | Z | X | Y | Z |
| P0 | 2 | 0 | 1 | 6 | 4 | 2 |
| P1 | 3 | 2 | 0 | 3 | 0 | 0 |
| P2 | 2 | 1 | 1 | 1 | 2 | 2 |

1 2 2
P2 2 1 1
3 3 3
P1 3 2 0
6 5 3
P0 2 0 1
8 5 4

"REQ 2" permitted ✓

B

Gate96.

| | current alloc | | | Max needed | | | Future need | | |
|---|---|---|---|---|---|---|---|---|---|
| | $R_0$ | $R_1$ | $R_2$ | $R_0$ | $R_1$ | $R_2$ | $R_0$ | $R_1$ | $R_2$ |
| $P_0$ | 1 | 0 | 2 | 4 | 1 | 2 | 3 | 1 | 0 |
| $P_1$ | 0 | 3 | 1 | 1 | 5 | 1 | 1 | 2 | 0 |
| $P_2$ | 1 | 0 | 2 | 1 | 2 | 3 | 0 | 2 | 1 |

Av = (2 2 0)

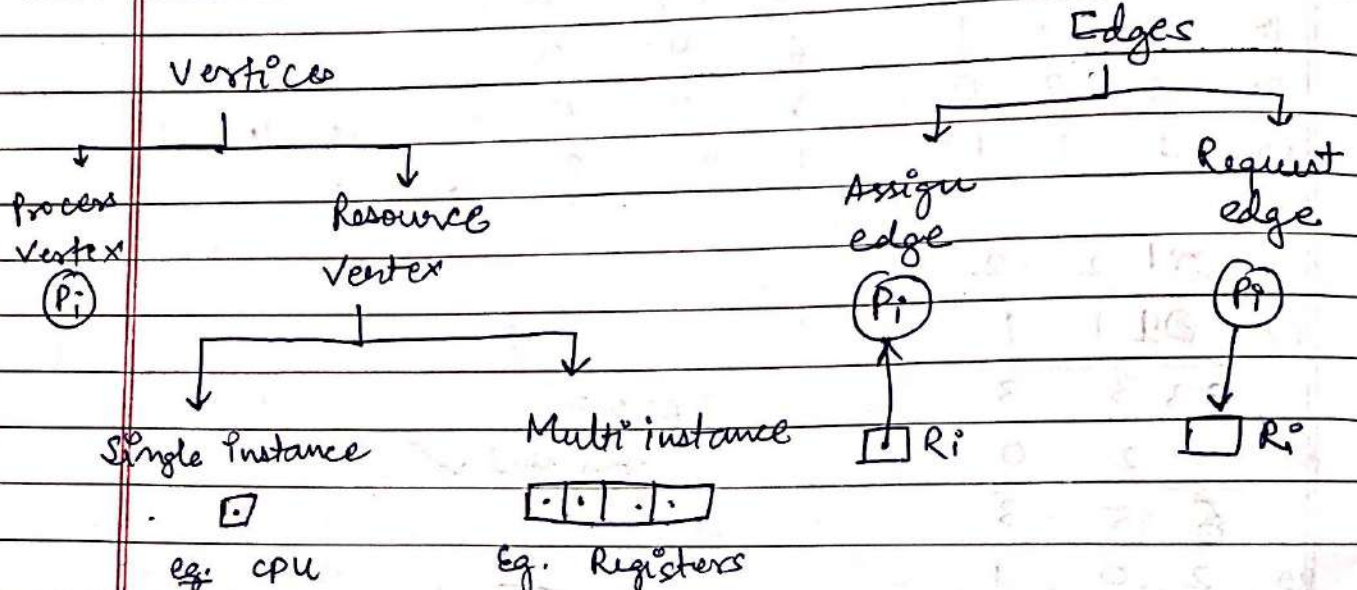
$Req_2 = P_0$: (0 1 0)

| | cur alloc | | | Future Need | | |
|---|---|---|---|---|---|---|
| | $R_0$ | $R_1$ | $R_2$ | $R_0$ | $R_1$ | $R_2$ |
| $P_0$ | 1 | 1 | 2 | 3 | 0 | 0 |
| $P_1$ | 0 | 3 | 1 | 1 | 2 | 0 |
| $P_2$ | 1 | 0 | 2 | 0 | 2 | 1 |

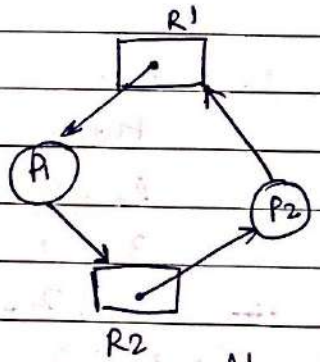
Av = 2 1 0

"unsafe"

# Resource Allocation Graph




eg.




Single instance resource type Graphs :- If there is a cycle, there will be "deadlock" surely. (Sufficient condition)

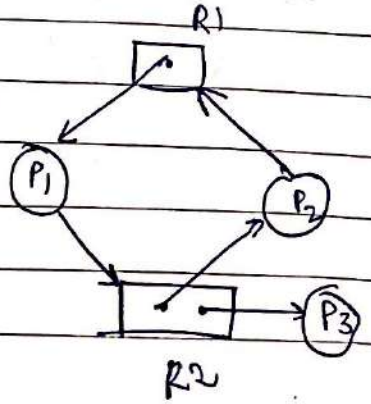
| | alloc | | request | |
|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_1$ | $R_2$ |
| $P_1$ | 1 | 0 | 0 | 1 |
| $P_2$ | 0 | 1 | 1 | 0 |

avail = (0 0)

Thus "unsafe"

∴ deadlock ✓

eg.




- Multi instance resource type graph.

| | alloc | | request | |
|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_1$ | $R_2$ |
| P1 | 1 | 0 | 0 | 1 |
| P2 | 0 | 1 | 1 | 0 |
| P3 | 0 | 1 | 0 | 0 |

$P_3 \quad P_1 \quad P_2$ →

"Safe state"

∴ Deadlock free

av = (0 0)

(P3) 0 1

0 1

(P1) 1 0

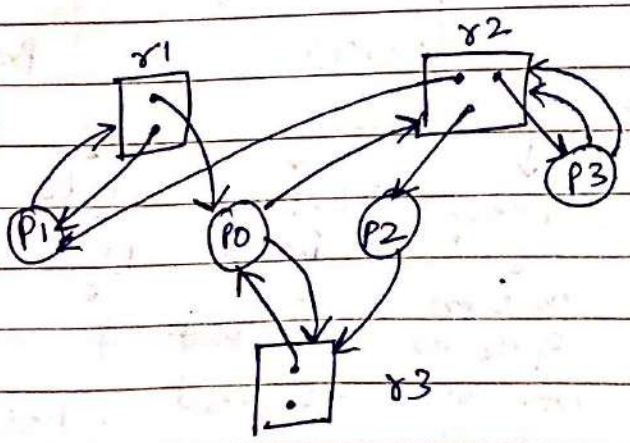
1 1

(P2) 0 1

1 2

- Even though there is a cycle, there is NO deadlock, ∴ in multi-instance resource type graph, a cycle need not necessarily mean deadlock.
- But still, if there has to be a deadlock, then there has to be a cycle. Thus cycle is a "necessary condition" in multi instance resource type graph but NOT "sufficient condition".
- Instead, cycle is a "necessary & sufficient" condition for single instance resource type graph.

Ora te 9y.



| | alloc | | | 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 | 0 | 2 | 0 |

av :- (0 0 1)

(P2) 0 1 0 → 0 1 1

(P0) 1 0 1 → 1 1 2

(P1) 1 1 0 → 2 2 2

(P3) 0 1 0 → 2 3 2

$P_2$ $P_0$ $P_1$ $P_3$ →

"safe"

and NO deadlock

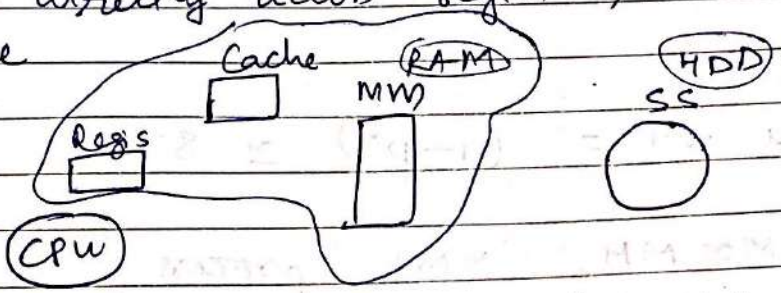
## Deadlock ~~Allocation~~ & Recovery
Detection

- Detect
  - Single instance resource type → Detect cycle (use DFS)
  - Multi instance resource type → Safety algorithm
- Recovery
  - Resources
    - Preempt → pull one of the res from user & give it to other • manual intervention
    - Roll Back → checkpoint required
  - Processes
    - Kill a process → if lucky → deadlock resolved
    - kill all processes & restart them

# 4. Memory Management

## Need of Multiprogramming

CPU can directly access register, main memory or cache




Problem:- All these components are costly

eg. 4MB MM, 4MB process

1 process - fraction of 'p' time doing I/O

CPU util = ?

CPU util = $(1-p)$

i.e. $p = 80\%$, CPU util = 20% (very low)

eg. 16MB MM, 4MB process

Then # process = 4 1 process - p doing I/O

Probability that all doing I/O during same time $= p^4$

CPU util = $(1-p^4)$

# process ↑, util ↑

If $p = 80\%$, CPU util = 60%

eg. 32 MB MM, 4MB process
8 process

CPU util = $(1-p^8) \simeq 83\%$ for $p = 80\%$

eg. 48 MB MM, 4MB process, 12 process

CPU util = $(1-p^{12}) \simeq 93\%$

- CPU util = $1-(p)^n$

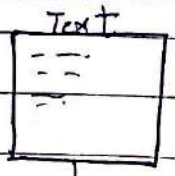
  where $n$ is the # processes that can be present is memory at same time.

- CPU util $\propto$ degree of multiprogramming 'n'

  As $n \to \infty$, CPU util $\to 1$

  ∴ For CPU util to be 100%, $n$ should be $\infty$
  For this, MM size should be $\infty$ (very very costly). → Use the memory in such a way

## Object Code Relocation & Linker




↓ Compiler

↓ assembly code

↓ Assembler

↓ Obj code

Date
Page 75

## object code

| Header |
|---|
| Text segment |
| Data Segment |
| Relocation Info |
| Symbol Table |
| Debugging Info |

absolute address ← relocatable address

relocation

| min() | 100 |
|---|---|
| printf() | — |
| scanf() | — |

resolution is done by linker

| Object code: A |
|---|
| Call: B, C, D |

| Object code: B |
|---|
| Call: D, E |
| call: printf |

C, D, E = system library

| library |
|---|
| C |
| D |
| E |
| F |
| : |

linker

2 passes

I phase
(i) Segment table
(ii) Symbol table

~~Executable code~~
Object code

II phase
resolve undefined references

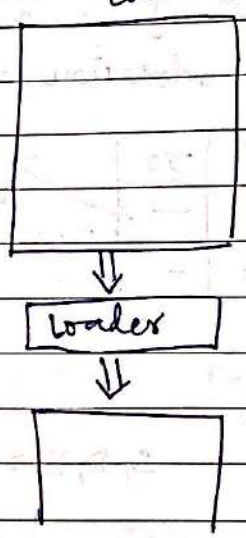
| A |
|---|
| B |
| C |
| D |
| E |
| printf() |

Stub (or) glue code

Responsibilities

(i) Relocation
(ii) Symbol resolution

Loader :-




(i) Program loading
Programs HDD → memory to run

(ii) Relocation

(iii) Symbol resolution

Linker :-
Symbol resolution + Relocation

loader :-
Program loading + Relocation.

Compile time : Symbol constants → relocatable address

Link time : Relocatable code → Relocatable addresses
ext ref.

load time :- Relocatable addr. → Absolute addresses

Run time :- Dynamic link libraries are linked using OS support

Gate 95:- A linker is given object modules for a set of programs that were compiled ~~successfully~~ separately. What information need not be included in an object module?

a) Object code
b) Relocation bits
c) Names & locations of all external symbols defined in the object module.
✓d) Absolute addresses of internal symbols.

Ans:- (D)

Gate 98 In a resident OS computer, which of the following system s/w must reside in the main memory under all situation

a) assembler
b) linker
✓c) loader
d) compiler

Ans:- (C)

Gate 01 The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned addresses is called

a) assembly
b) parsing
✓c) Relocation
d) Symbol resolution

Ans:- (C)

Gate04 Consider a program 'p' that consists of two source modules $M_1$ and $M_2$ contained in two different files. If $M_1$ contains a reference to a function defined in $M_2$, the reference will be resolved at

a) Edit time
b) Compile time
c) link time
d) load time

Ans. (C)

Gate02 Dynamic linking can cause security concerns because :-

a) Security is dynamic.
b) the path for searching dynamic libraries is not known till run time.
c) linking is insecure
d) Cryptographic procedures are not available for dynamic linking.

Ans. (B)

Gate03 which of the following is NOT an advantage of using shared, dynamically linked libraries as opposed to using statically linked libraries?

a) Smaller sizes of executable file
b) lesser overall page fault rate in the system.
c) Faster program setup
d) existing programs need not be relinked to take advantage of newer versions of libraries.

Ans: (B)

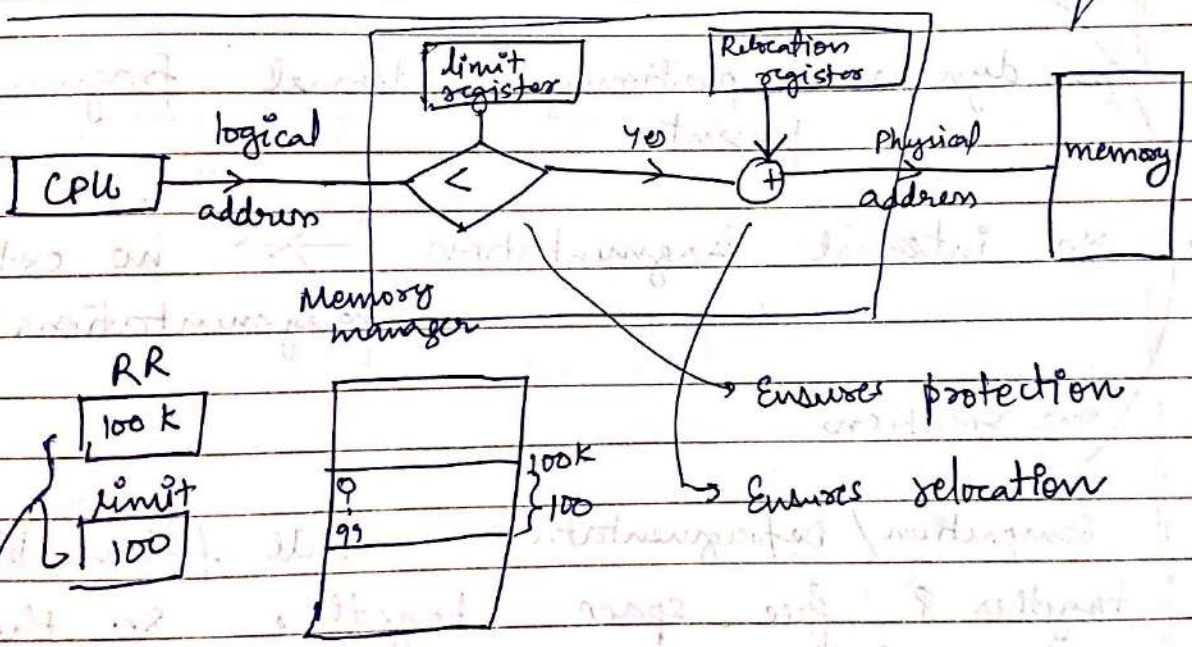
## Fixed Partioning

Contigous memory allocation


Bcoz Problem :-

- Internal Fragmentation.
- Limitation process size.
- External Fragmentation.
- Degree of multiprogramming limited.

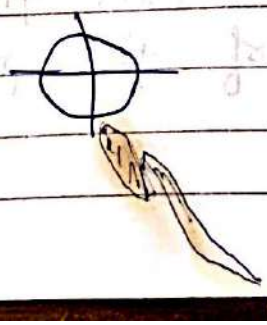
Relocation & Protection in H/W



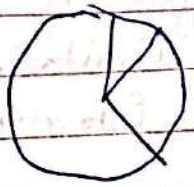

→ under the control of OS. (Privilege registers)

## Dynamic Partioning

Fixed

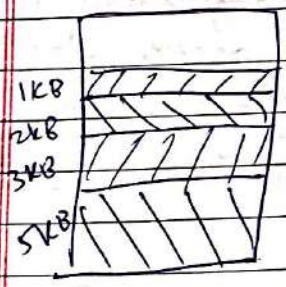


Dynamic



In dynamic partioning - no internal fragmentation

- If there is internal fragmentation, there is definitely external fragmentation.




Now 1KB space & 3 KB space is freed up then :- we have 4 KB free space but is not contigous so we cannot give it to anyone - External fragmentation.

In dynamic partioning - external fragmentation is present

- No internal fragmentation $\not\rightarrow$ no external fragmentation

one solution

Compaction / Defragmentation :- Pull filled blocks together & free space together so that entire free space comes together as a big block.

Advantages :-

(i) Degree of MP is dynamic (not fixed)
(ii) No limitation on size of the process
(iii) No internal fragmentation.

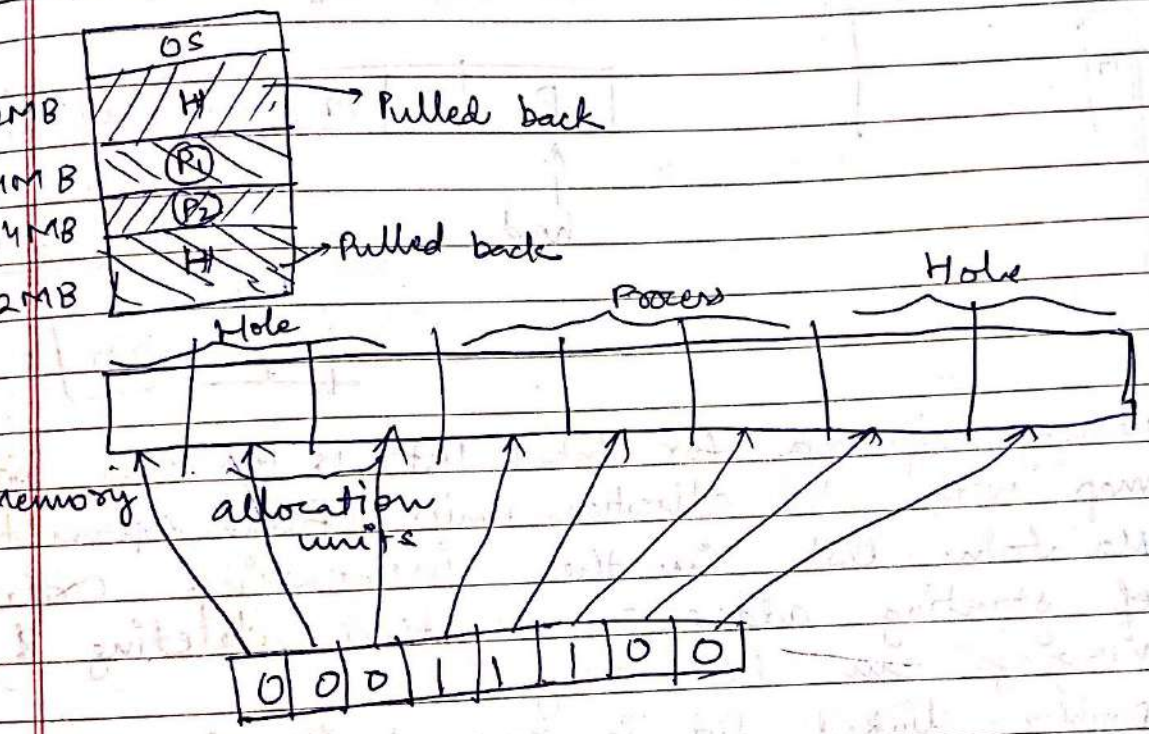
Data structures for partitioning

Bit-Map | Linked-list

Disadvantages :-

(i) Allocation & Deallocation of memory is complex
(ii) External fragmentation

Bit Map for Dynamic Partioning :-

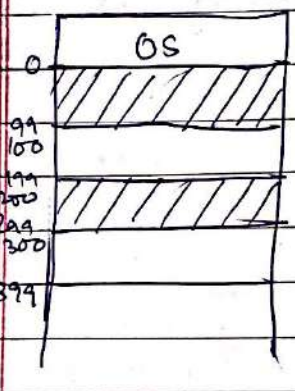


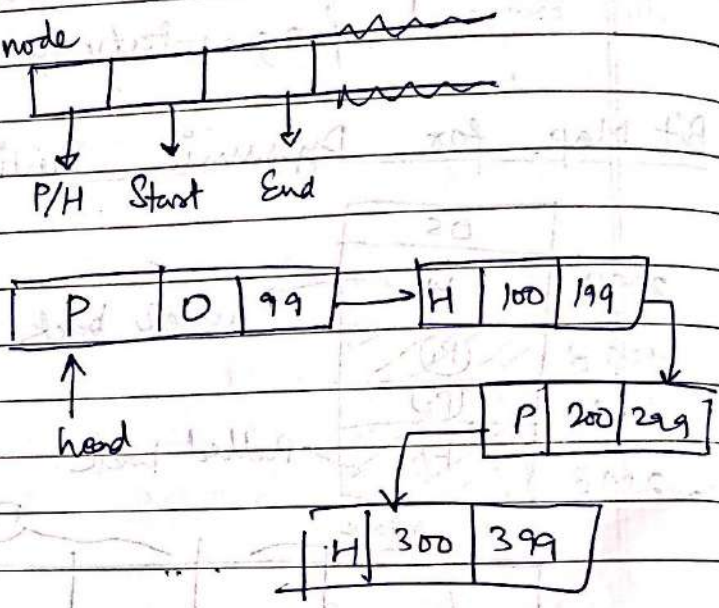

| Allocation unit Size | fraction of memory used by bit map |
|---|---|
| 1 bit | 1/2 |
| 4 bytes | 1/32 |

used widely

Allocation unit ↑ space wastage ↑ (→ static part.)
↓ memory wastage ↑ (fraction ↑)

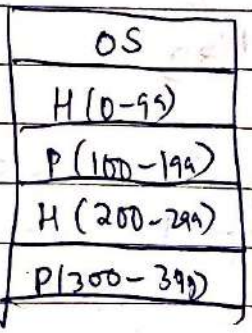
## Linked List for Dynamic Partitioning

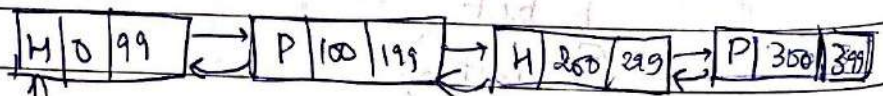



- Memory required for linked list is less than bit-map with 4B allocation units (found experimentally)
- Maintain list in the increasing order of starting address so that deleting & merging can be easy.
- Doubly linked list is more benificial.

### First fit



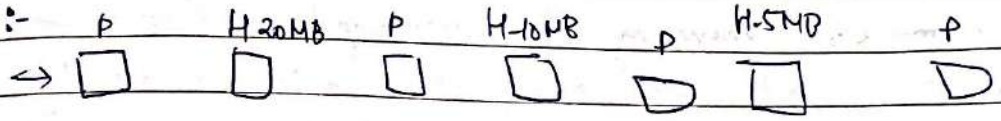

Head

P = 50 bytes

### Next fit

Hole → Scan from here

**Best Fit :-**

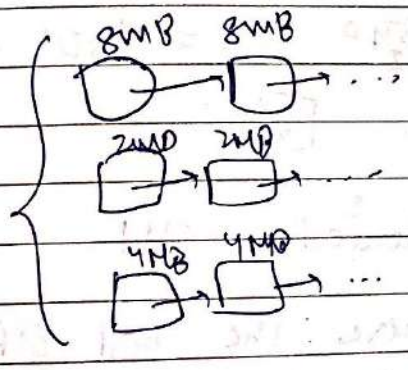



Smallest hole greater than or equal to demand. Not good approach. smaller portions left will not be used.

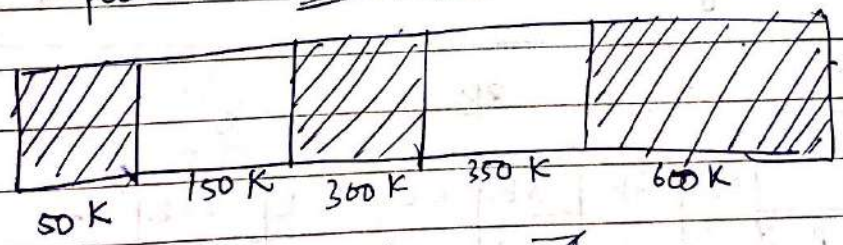
**Worst Fit :-**

largest hole will be allocated.

**Quick Fit :-**

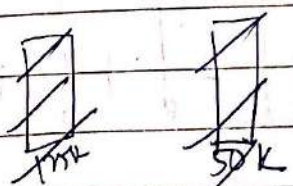



**Gate 94** Process requests 300K, 25K, 125K, 50K.




a) Either FF or BF

b) FF but not BF



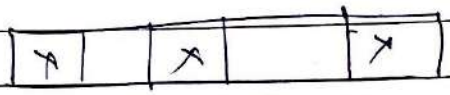

c) BF but not FF


d) None

Gate96 A 1000 KB memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 KB & 260 KB respectively. The smallest allocation request in Kbytes that could be denied is for

a) 151 b) 181 c) 231 d) 541

Sol-



$X + 200 + X + 260 + X = 1000$

$3X = 1000 - 460 = 540$, $X = 180$

(B)

also,

largest pr. size that could be denied = 541

Gate98 In a computer system where the 'best-fit' algorithm is used for allocating 'jobs' to 'memory partitions', the following situation was encountered

| Partitions Size in KB | 4K | 8K | 20K | 2K |
|---|---|---|---|---|

| Jobs | 2K | 14K | 3K | 6K | 10K | 20K | 2K |
|---|---|---|---|---|---|---|---|
| Time for execution | 4 | 10 | 2 | 1 | 1 | 8 | 6 |

When will the 20K job complete?

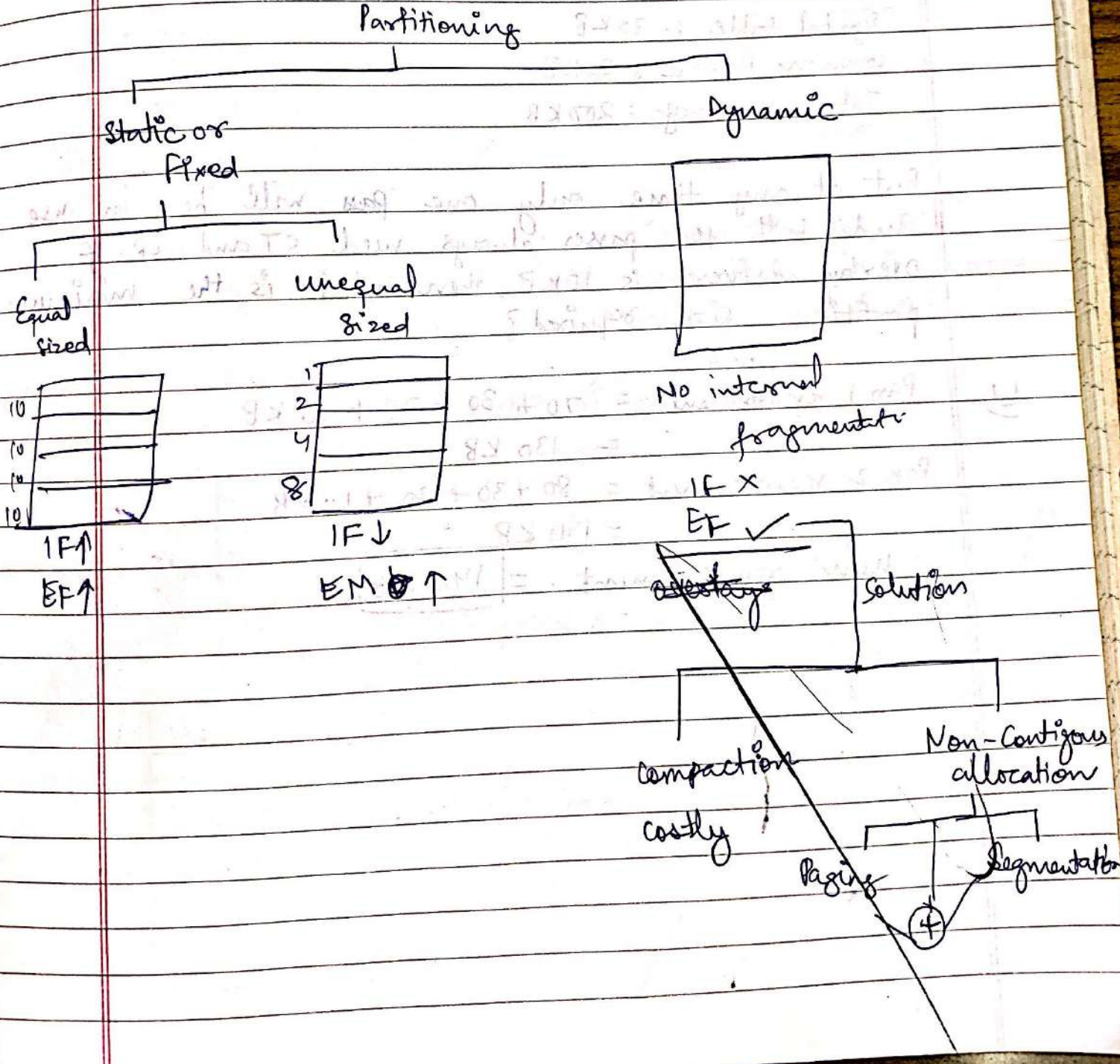
Sol:

| 3K | 6K | 14K | 2K |
|---|---|---|---|
| 4K | 8K | 20K | 2K |

| | | | |
|---|---|---|---|
| 4K | 3K (0-2) | | |
| 8K | 6K (0-1) | 2K (1-7) | |
| 20K | 14K (0-10) | 10K (10-11) | 20K (11-19) |
| 2K | 2K (0-4) | | |

19

## Summary on Partitioning



## Overlays

Size of program > size of biggest partition

Eg Consider a 2 pass assembler:-

pass 1 : 70 KB
pass 2 : 80 KB
Symbol table : 30 KB
Common Routine : 20 KB
Total memory : 200 KB

But at any time only one pass will be in use and both the passes always need ST and CR. If overlay driver is 10 KB, then what is the minimum partition size required?

Sol Pass 1 requirement = 70 + 30 + 20 + 10 KB
= 130 KB
Pass 2 requirement = 80 + 30 + 20 + 10 KB
= 140 KB
Min requirement = **140 KB**

150 - 64
200 - 62
250 - 61
300 - 60

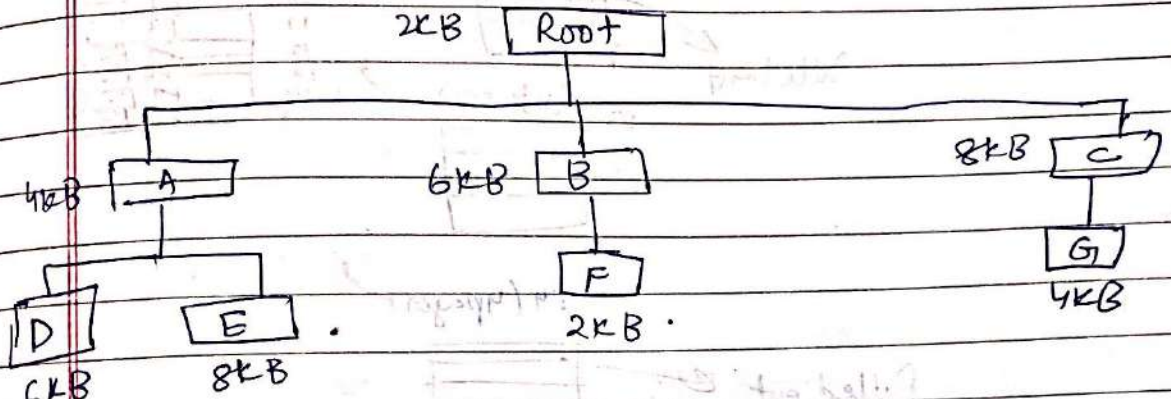
1 - 84
50 - 71
100 - 67

80+ = 5


125 - 66

3hr

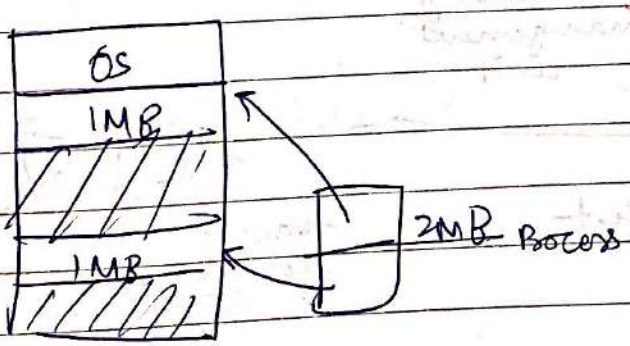
Gate98 The overlay tree for a program is shown below:

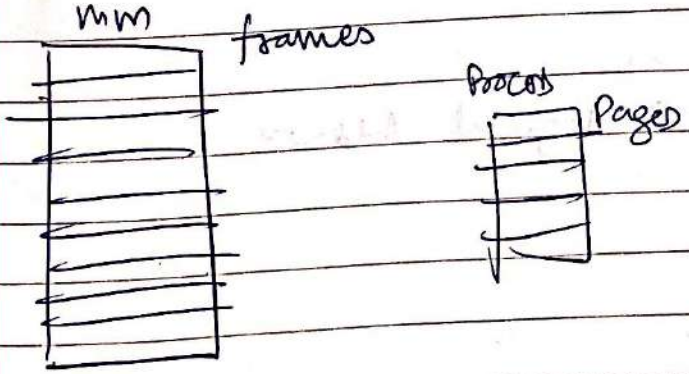



What will be the size of partition (in physical memory) required to load (and run) this program?

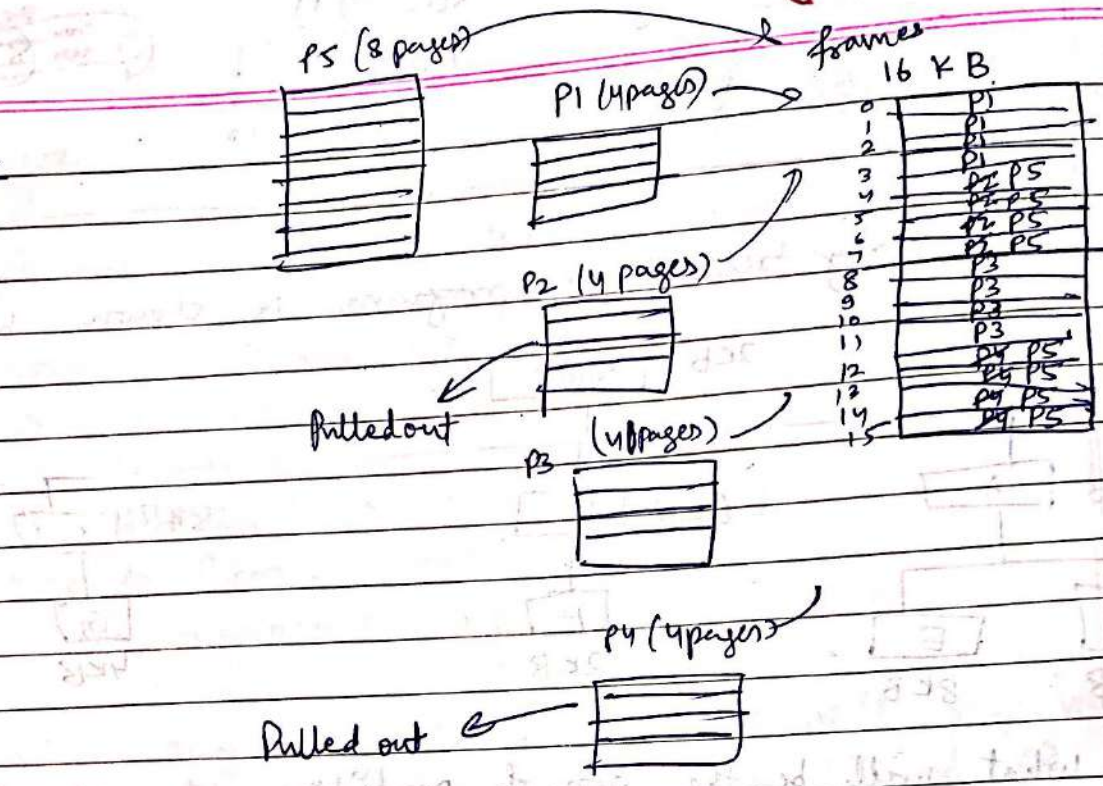
A) 12KB  B) 14KB  C) 10KB  D) 8KB

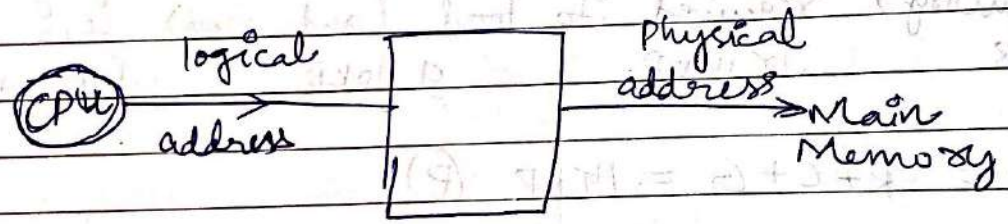
Sol. Max = R+C+G = 14KB (B)

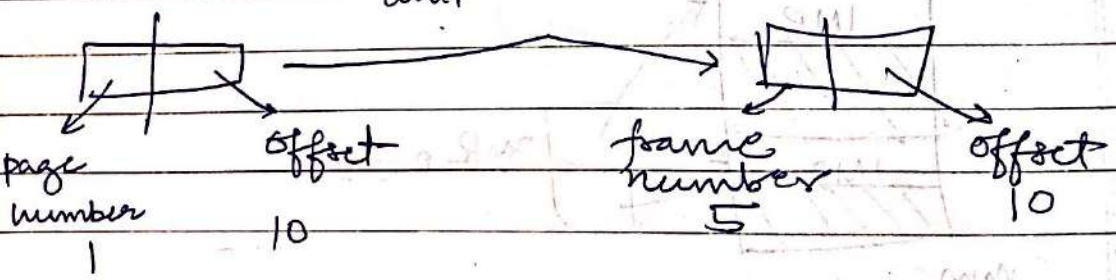


Paging




- Logical address → Physical Address


eg. Memory = 64 B

$2^6 = 64$ → 6 bits

Frame = 4 B

# Frames = $\frac{64}{4} = 16$

Process = 16 B

Page Size = 4 B

# Pages in Process = $\frac{16}{4} = 4$

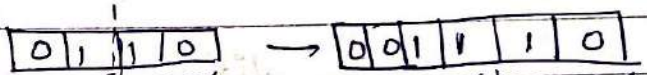
| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 |
| 3 | 12 | 13 | 14 | 15 |

Process P1

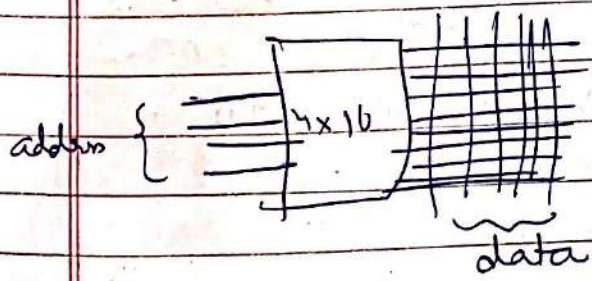
| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | X |
| 1 | 4 | 5 | 8 | 7 | X |
| 2 | 8 | 9 | 10 | 11 | P1 |
| 3 | 12 | 13 | 14 | 15 | P2 |
| 4 | 16 | 17 | 18 | 19 | P3 |
| 5 | 20 | 21 | 22 | 23 | P1 |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |


Page table

| | |
|---|---|
| 0 | f2 |
| 1 | f3 |
| 2 | f4 |
| 3 | f5 |




Word :- Smallest addressable unit in a computer.




Physical address space = Size of main memory

eg. PAS = 128 KB, WS = 4B

$= 2^{15}$ words

∴ PA = 15 bits

eg. PAS = M words, PA = $\lceil \log_2 M \rceil$ bits.

Logical address space = size of a process

eg. LAS = 256 MB, WS = 4 B

= $2^{28}$ B

or = $2^{26}$ words

Logical addres. LA = 26 bits.

Virtual Memory :- Whenever the size of process is larger than the size of main memory we put only a part of process in MM & be able to run it.

- Nowadays, virtual memory is also used to run smaller processes (to increase degree of multiprogram)

• Page size = Frame size = P words.

# bits required (Page offset)

= $\lceil \log_2 P \rceil$ bits.

eg. Page size = 4 KB, Word size = 4B

Words / page = 1K words

= $2^{10}$ words

# bits = 10 (offset)

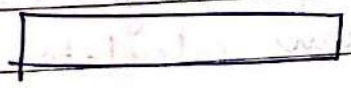
## Page Table

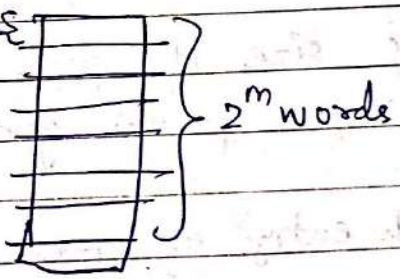
PAS = Main Memory Size = M words
LAS = Process size = L words
Page size = P words
PA = $\lceil \log_2 M \rceil$ ~~words~~ bits = m bits

LA = $\lceil \log_2 L \rceil$ = l bits
(offset)
Page size = $\lceil \log_2 P \rceil$ = p bits

eg ① LAS = 128 MB = $2^{27}$ B } virtual address has to be used
PAS = 1 MB = $2^{20}$ B
PS = 4 KB = $2^{12}$ B

LA = 27, PA = 20 bits, Page offset = 12 bits

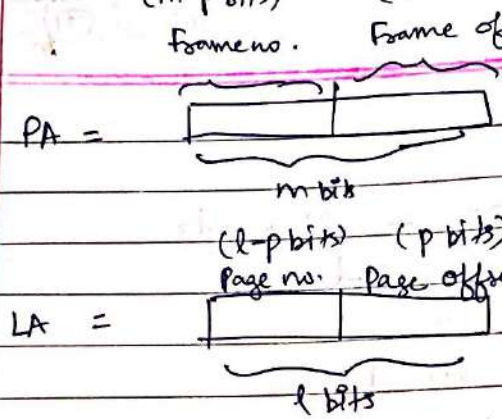
PA = 

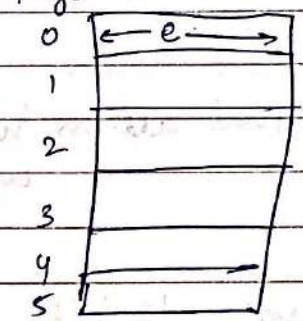

frame = $2^p$ words
PAS = $2^m$ words

# frames = $2^{m-p}$ frames
LAS = $2^l$ words PS = $2^p$ words

# pages = $2^{l-p}$ pages




Page table will contain $2^{l-p}$ entries (# pages)




No of entries = $2^{l-p}$

Each entry size = 'e' bytes

∴ Size of page table

$= e \cdot 2^{l-p}$ bytes

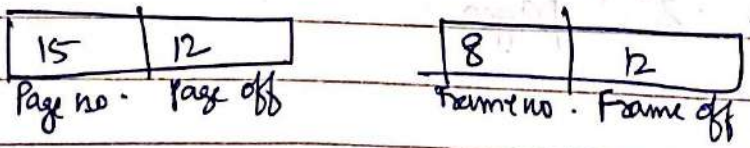
If size of page table entry is given - Fine, take it, if not given - we can calculate it

Each entry of page table contains frame number,

no. of bits for frame identification = m-p bits

∴ Size of page table = $(2^{l-p}) \cdot (m-p)$ bits.

eg. (1) contd: LA = 27 bits, PA = 20 bits, $P_{off}$ = 12 bits




Size of page table = $2^{15} \times 8$ bits = 32 KB.

Page size = 4k B , Page Table size = 32 kB

↓

"Cannot fit in one page"

↓

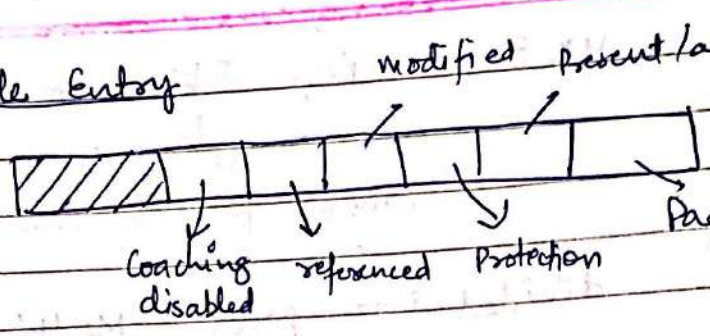
divided into pages (Multi level paging)

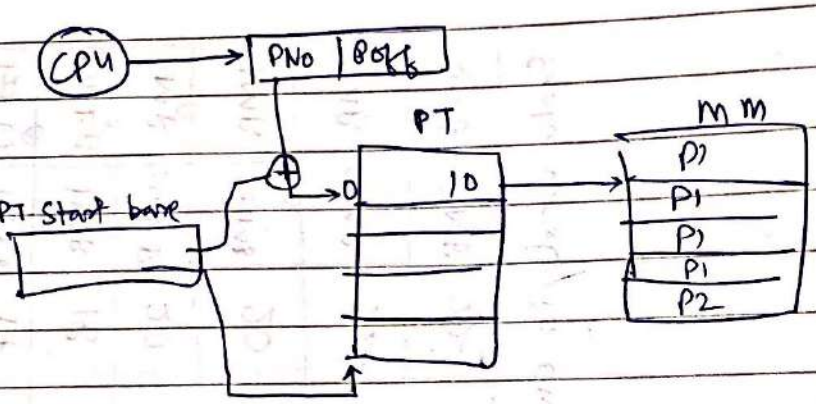
eg:

| LAS | PAS | LA | PA | Page size | Page offset | Pages | Frames | PTE | PT Size |
|---|---|---|---|---|---|---|---|---|---|
| 128 KB | 128 KB | 17 | 17 | 4 KB | 12 | 32 | 32 | 5 bits | 5 × 32 bits |
| 256 KB | 1 MB | 18 | 20 | 4 KB | 12 | 64 | 256 | 2B | 128 B |
| 1 MB | 256 KB | 20 | 18 | 1 KB | 10 | $2^{10}$ | 256 | 1B | 1 KB |
| 1 MB | 512 MB | 20 | 29 | $2^{12}$ | 12 | 256 | $2^{17}$ | 17 bits | 32 × 17 B |
| 4 MB | 2 MB | 22 | 21 | $2^{12}$ | 12 | $2^{10}$ | $2^{9}$ | 9 bits | 128 × 9 B |
| 4 MB | 4 MB | 22 | 22 | $2^{14}$ | 14 | $2^{8}$ | 256 | 1B | 256 B |

Each word is one byte (system Ps byte addressable)

Page table Entry




Gate 04. In a virtual memory system, size of virtual address is 32 bit, size of physical address is 30 bit, page size is 4KB, and size of each page table entry is 32-bit. The main is byte addressable which of the following is maximum number of bits that can be used for storing protection & other information in each page table entry

a) 2 b) 10 c) 12 d) 14

Sol:- ⓐ # frames $= 2^{30-12} = 2^{18}$

18 $\quad$ 32 − 18

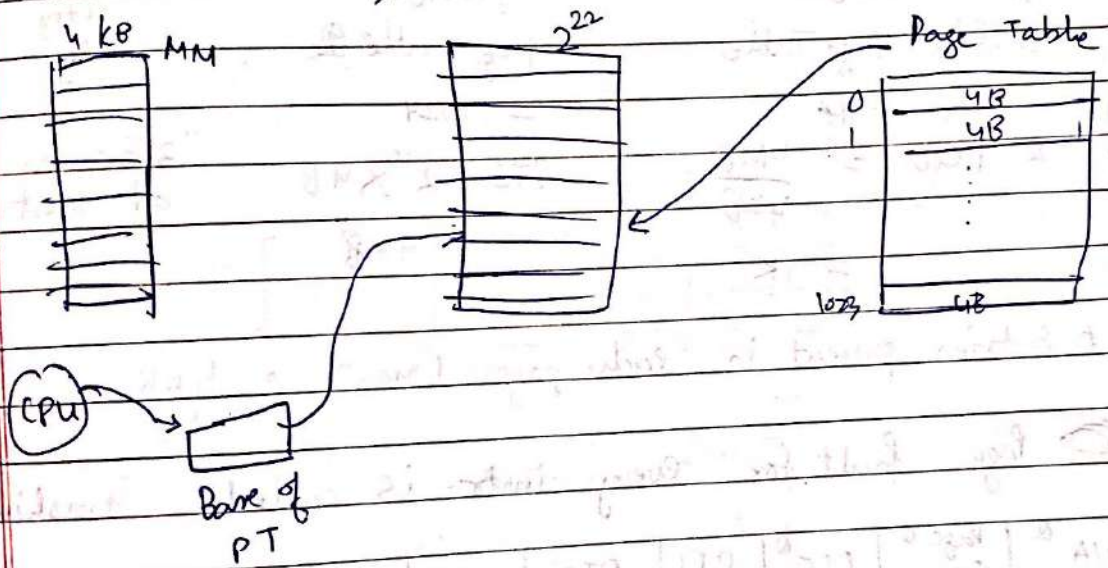
$= 14$ (D)

## Multi-level Paging

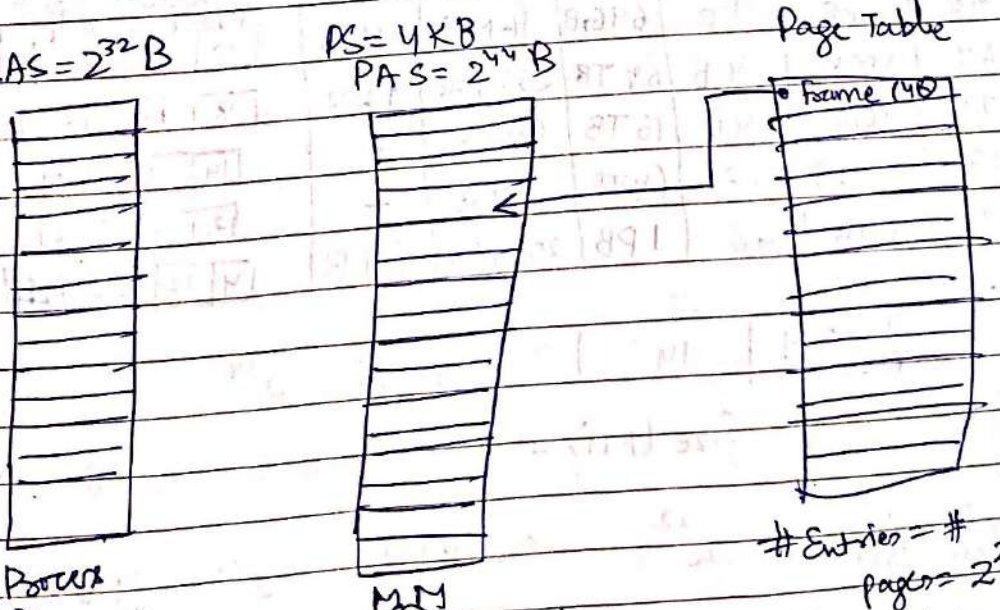
eg. LA = 22 bits
LAS = $2^{22}$ B = 4MB

Page size = 4kB
offset = 12 bits
lucky

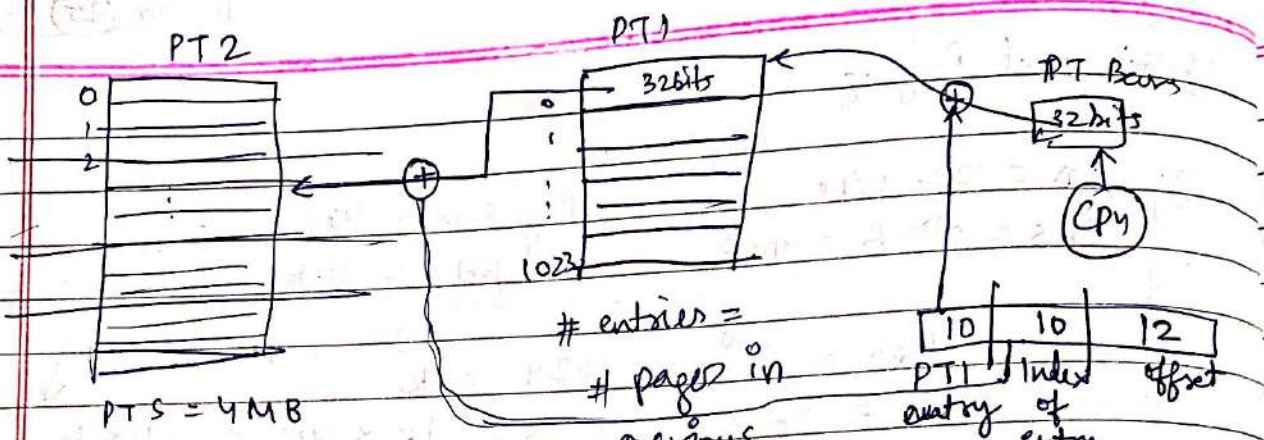
# Pages = $2^{10}$ = 1024 = 1K
PTE = 4B
PTS = 1K × 4B = 4KB




eg. LAS = $2^{32}$ B
PS = 4KB
PAS = $2^{44}$ B




# Entries = # pages = $2^{20}$
PTS = $(2^{20} \times 4)$ B
= 4MB




PTS = 4MB

PS = 4KB

divide Page Table into Pages

# Pages = $\frac{4MB}{4KB}$ = 1K

# entries = # pages in previous page table = 1024

PTS = $2^{10} \times 4B$ = 4KB

"3 Frames at least"

# Entries present in each page (max) = $\frac{4KB}{4B}$ = 1K

Page fault for every instr. is called "trashing"

Eg:

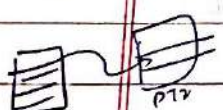
| | VA | Page size | PTE | PT1 | PT2 | PT3 | address split (PT3 \| PT2 \| PT1 \| Page offset) |
|---|---|---|---|---|---|---|---|
| ① | 48 | 16KB | 4B | 64GB | 16MB | 4KB | 10 \| 12 \| 12 \| 14 |
| ② | 64 | 1MB | 4B | 64TB | 256MB | 1KB | 8 \| 18 \| 18 \| 20 |
| ③ | 72 | 1GB | 4B | 16TB | 64KB | – | 14 \| 28 \| 30 |
| ④ | 72 | 256MB | 4B | 64TB | 1MB | – | 18 \| 26 \| 28 |
| ⑤ | 72 | 16MB | 4B | 1PB | 256MB | 64B | 4 \| 22 \| 22 \| 24 |

Sol ①

| 34 | 14 |
|---|---|

#pages = $2^{34}$

Size (PT1) = $2^{34} \times 4B = 2^{36} B = 64GB$




(#Pages)$_{PT1}$ $\frac{2^{36}}{2^{14}} = 2^{22}$

Size (PT2) = $2^{22} \times 4B = 2^{24} B$ = 16MB

(#Pages)$_{PT2}$ = $\frac{2^{24}}{2^{14}} = 2^{10}$

Size (PT3) = $2^{10} \times 4B$ = 4KB

Neel Kamal
Date
Page 97

② | 44 | 20 |

# pages = $2^{44}$

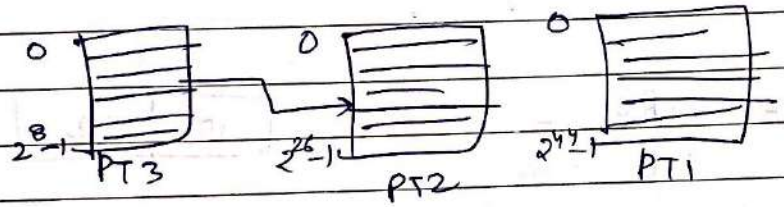
Size (PT1) = $2^{44} \times 4B = 2^{46} B$

# Pages (PT1) = $\frac{2^{46}}{2^{20}} = 2^{26}$

Size (PT2) = $2^{26} \times 4B = 2^{28} B$

# Pages (PT2) = $\frac{2^{28}}{2^{20}} = 2^{8}$

Size (PT3) = $2^{8} \times 4B = 2^{10} B$




③ | 42 | 30 |

# pages = $2^{42}$

Size (PT1) = $2^{42} \times 4B = 2^{44} B$

# Pages (PT1) = $\frac{2^{44}}{2^{30}} = 2^{14}$

Size (PT2) = $2^{14} \times 4B = 2^{16} B$

④ | 44 | 28 |

# pages = $2^{44}$

Size (PT1) = $2^{44} \times 2^{2} = 2^{46}$

# Pages (PT1) = $\frac{2^{46}}{2^{28}} = 2^{18}$

Size (PT2) = $2^{18} \times 4B = 2^{20} B$

⑤ | 48 | 24 |

# pages = $2^{48}$

Size (PT1) = $2^{48} \times 2^{2} = 2^{50}$

# Pages (PT1) = $\frac{2^{50}}{2^{24}} = 2^{26}$

Size (PT2) = $2^{26} \times 2^{2} = 2^{28}$

# Pages (PT2) = $\frac{2^{28}}{2^{24}} = 2^{4}$

Size (PT3) = $2^{4} \times 2^{2} = 2^{6}$

| eg. | Page size | PTE | Outer page table size | levels of paging | virtual address spaces |
|---|---|---|---|---|---|
| ① | 4KB | 4B | 4KB | 1 | 22 bits (4MB) |
| ② | 4KB | 4B | 4KB | 2 | 32bits (4GB) |
| ③ | 4KB | 4B | 4KB | 3 | 42 bits (4TB) |
| ④ | 4KB | 4B | 256 B | 1 | 256 KB (18 bits) |
| ⑤ | 4KB | 4B | 256 B | 2 | 256 MB (28 bits) |
| ⑥ | 4KB | 4B | 256 B | 3 | 256 GB (38 bits) |

Assume that except the outer level page table, all the pages of inner level PT are completely full.

Sol. ① $\#\text{pages} = \frac{4KB}{4B} = 2^{10}$

| 10 | 12 |
|---|---|

② $(\#\text{Pages})_{PT1} = 2^{10}$

| 10 | 10 | 12 |
|---|---|---|

③

| 10 | 10 | 10 | 12 |
|---|---|---|---|

$2^{42}$

④

| 6 | 12 |
|---|---|

$2^{18}$

⑤

| 6 | 10 | 12 |
|---|---|---|

$2^{28}$

⑥

| 6 | 10 | 10 | 12 |
|---|---|---|---|

$2^{38}$

eg. Fill in the blanks in such a way that the page table will fit in one page in single level paging

| | VAS | PS | PTE |
|---|---|---|---|
| ① | 4MB | 4KB | 4B |
| ② | 4GB | 128KB | 4B |
| ③ | 128TB | 32MB | 8B |
| ④ | 256MB | 32KB | 4B |
| ⑤ | 512KB | 1KB | 2B |
| ⑥ | 16GB | 256KB | 4B |

Sol: ① | 10 | 12 | (22)

$\frac{2^{12}}{2^{10}} = 2^2$

② | 15 | 17 | (32)

$\frac{2^{17}}{2^{15}} = 2^2$

③ | 22 | 25 | (47)

$\frac{2^{25}}{2^{22}} = 2^3$

④ | y | x | (28)

$x - y = 2$

$x + y = 28$

$2x = 30$

$x = 15 \quad y = 13$

⑤ $x + y = 19$

$x - y = 1$

$2x = 20 \quad x = 10$

⑥ $x + y = 34$

$x - y = 2$

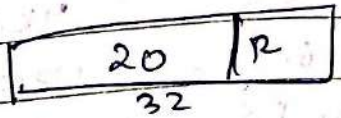
$2x = 36 \quad x = 18$

Gate01 Consider a m/c with 64MB physical memory and 32-bit virtual address space. If the page size is 4KB, what is the approximate size of PT?

a) 16MB b) 8MB c) 2MB d) 24MB

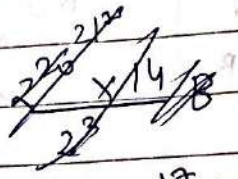
Sol. PA 26 bits VA 32 bits

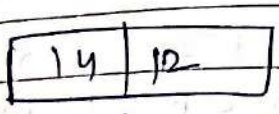
PS = 4KB = 12 bits




# pages = $2^{20}$

# frames = $2^{14}$




$14 \times 2^{20}$ bits = $2^{17} \times 14$ B

$\simeq$ 2MB (c)

Gate 99 Which of the following is/are advantages of virtual memory?

x a) Faster access to memory on an average.

x b) Processes can be given protected address spaces.

x c) Linker can assign addresses independent of where the program will be loaded in the physical memory.

✓ d) Programs larger than the physical memory size can be run.

Ans. (d)

Gate 95 In a virtual memory system, the address specified by the address lines of CPU must be ____ than the physical memory size and ____ than the secondary memory size.

Sol: more, less

Gate 97 Dirty bit for a page in Page table
✓a) Helps avoid unnecessary writes on paging device.
b) Helps maintain LRU information
c) Allows only read on a page
d) None

Ans: (A)

Gate 06 A computer system supports 32-bit virtual addresses as well as 32 bit physical addresses. Since the VAS is of same size as PAS, the OS designers decide to get rid of VM entirely. Which of the following is true?
✓a) Efficient implementation of multi user support is no longer possible.
✗b) The processor cache organization can be made more efficient now.
✗c) H/w support for memory management is no longer needed
✗d) CPU scheduling can be made more efficient now.

Ans: (A)

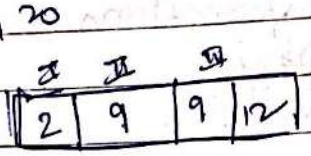
Gate 08 PA = 36 bits, VA = 32 bits, PS = 4 KB, PTE = 4 B. A three level PT is used. VA is divided as follows:-
→ Bits 30-31 are used to index I level PT.
→ Bits 21-29 are used to index II level PT.
→ Bits 12-20 are used to index III level PT.
→ Bits 0-11 are used as offset within page.

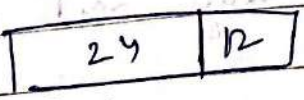
(* ~~extra~~ bits in PTE for frame no)

Then PTE sizes in I, II, III level PT are :-

a) 20, 20, 20
b) 24, 24, 24
c) 24, 24, 20
d) 25, 25, 24

Sol.




32 - 12 = 20

PS = 4kB, PTE = 4B

$$\frac{2^{20} \times 2^{12}}{2^9}$$

24 bits

$$2^{20} \times 4B = \frac{2^{22} B}{4KB} = 2^{10}$$

(B)

Gate09 A multilevel PT is preffered in comparision to single level PT for translating VA to PA because :-

(A) It reduces the memory access time to read or write a memory location.

(B) It helps to reduce the size of PT needed to implement the virtual address space of a process.

(C) It is required by the translation lookaside buffer.

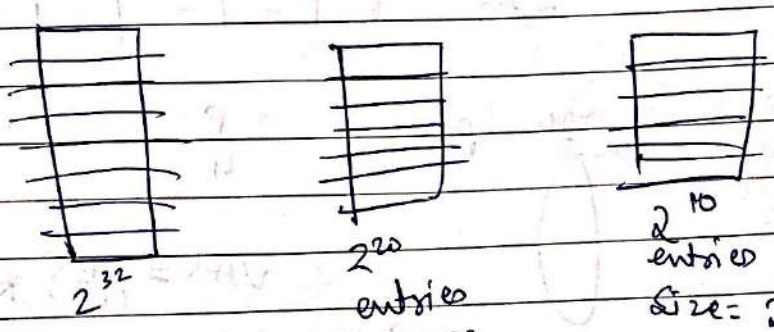
(D) It helps to reduce the number of page faults in page replacement algorithms.

Ans. (B)

Gate02 VA = 32, PA = 32 Byte addressable. PS = 4kB. Two level P.T., PTE = 4 B.

a) Diagram showing VA to PA conversion.
b) Page table entries in each page.
c) How many bits are available for storing protection & other information in each PTE.

Sol: (a)




# pages = $2^{20}$ Size = $2^{22}$ B

# pages = $2^{10}$

| VA = | PT II | PT I | offset |
|---|---|---|---|
| | 10 | 10 | 12 |

(b) # PTE = $2^{10}$

(c) # frames = $2^{20}$, PTE = 4B = 32

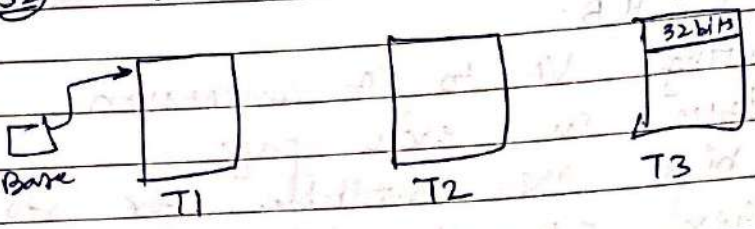
# bits (extra) = 32 − 20 = 12

Gate08 IT

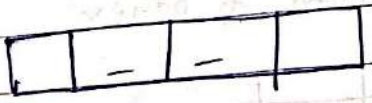
| | |
|---|---|
| 1) Dirty | a) Page initialization |
| 2) R/w | b) write - back policy |
| 3) Reference | c) Page protection |
| 4) Valid | d) Page replacement policy |

Ans: 1 → B
2 → C
3 → D
4 → A

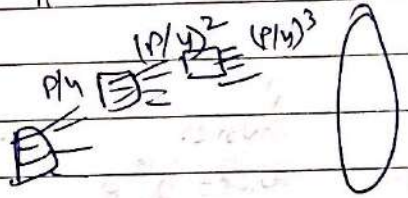
Gate B (52) VA = 46 PA = 32




~~4B~~ let size = p bytes



# entries in one page = $\frac{p}{4}$




# page = $\left(\frac{p}{4}\right)^3$

$VAS = \left(\frac{p}{4}\right)^3 * p$

$= 2^{46}$

$\Rightarrow p^4 = 2^{5-2}$

$\Rightarrow p = 2^{13} = 8KB$

## Finding optimal page size

- PS ↑ PTS ↓

overhead

Average page wastage = $\left(\frac{1}{2}\right)$ Page

Say, page size = p bytes, PTE = e bytes

on average, VAS = S bytes

Overhead = $\left(\frac{p}{2}\right) + \left(\frac{S}{p}\right) * e$

↑ page wasted on avg.

↑ # pages (a waste)

minimize the overhead :-

$$\frac{dov}{dp} = 0 \rightarrow \frac{1}{2} - \frac{se}{p^2} = 0$$

$$\Rightarrow p = \sqrt{2se}$$

- For avg. 'virtual address space (s) and page table entry (e), the value of page size so that overhead is minimized, $\boxed{p = \sqrt{2se}}$

eg.

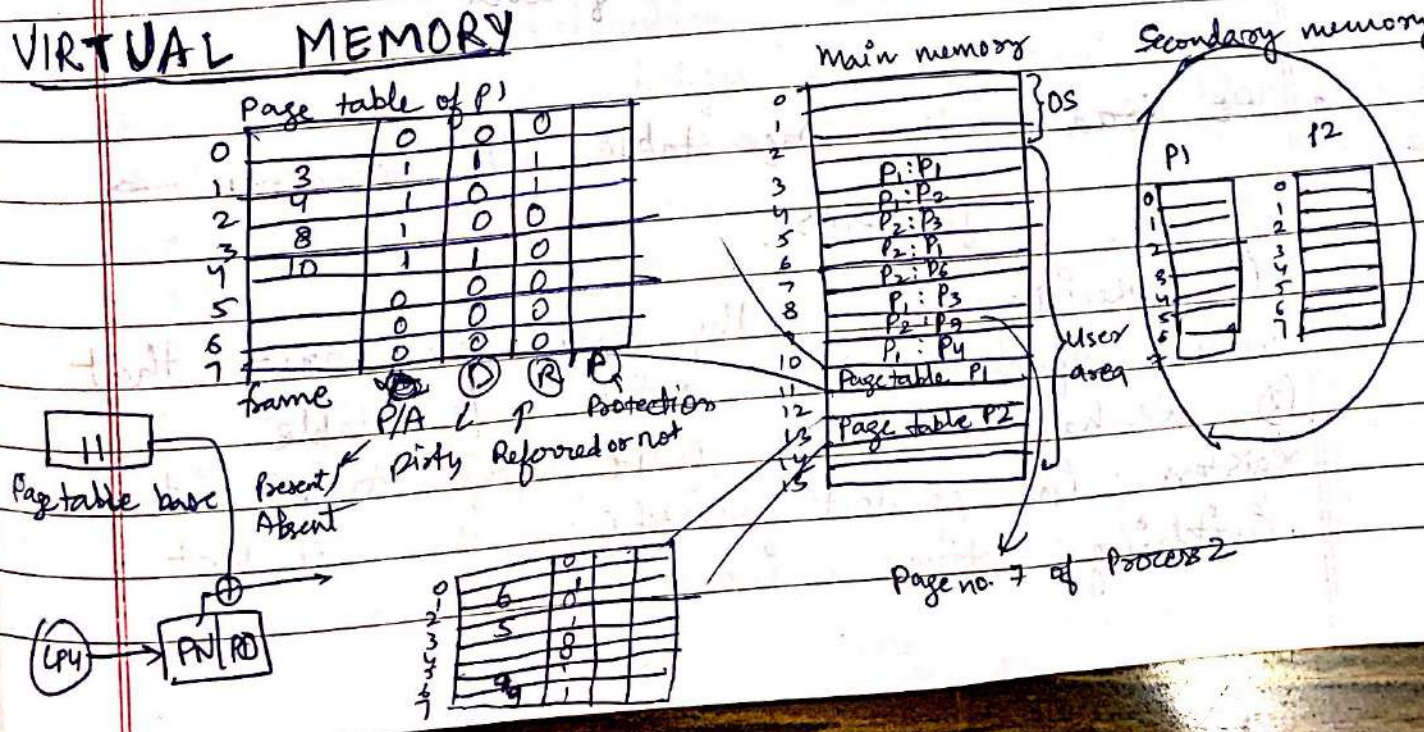
| | s | e | → p |
|---|---|---|---|
| ① | 4KB | 8 B | 256 B |
| ② | 16 MB | 2 B | 8KB |
| ③ | 256 GB | 32 B | 4 MB |

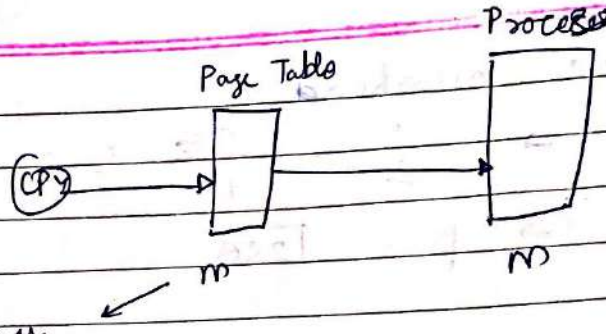
① $p = \sqrt{2 \times 2^{12} \times 2^3}$
$= 2^8 B$

② $p = \sqrt{2 \times 2^{24} \times 2}$
$= \sqrt{2^{26}} = 2^{13} B$

③ $p = \sqrt{2^{38} \times 2^5 \times 2}$
$= \sqrt{2^{44}} = 2^{22} B$

# VIRTUAL MEMORY

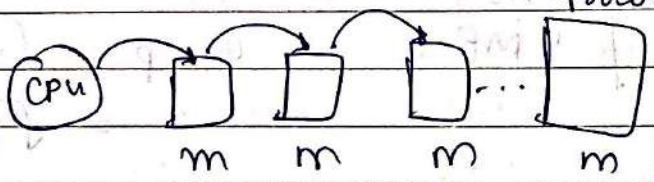





memory access time

Effective access time = 2 × m (due to paging)

- And by using multilevel page tables, the effective access time increases even more.




EAT = (L+1) × m

Solution

no. of levels

- We load entire page table into registers →

↓Problems

① Registers are costly and don't have that much space to load entire page table

② We have to load entire page table into registers in context switching, thus context switching time increases.

↓ Solution

load page table in some point in main memory and load base address of page table in one register.

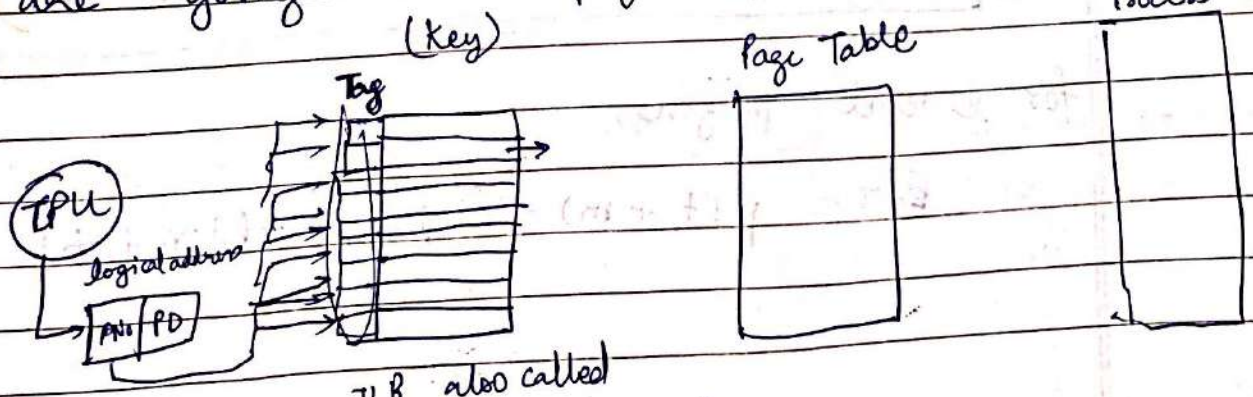
Final Solution →

We will not use main memory and also not use registers, we will use some kind of memory which is faster compared to main memory but cheaper compared to registers - "CACHE".

Also called "TLB" (Translation Lookaside Buffer).

Locality of Reference :- A process will access only few pages for a long time; and these few pages will keep on changing but, at any point of time, a process will not access all the pages available; it needs only few pages.

Concept (of TLB) :- whenever you need any page, then the particular page table entry will be referred, and whenever a particular page table entry is referred for the first time, we are going the page table entry inside TLB




TLB also called "associative memory".

- TLB only contains frequently used page table entries.

eg. t (TLB access time) = 20 ns
m (main memory access time) = 100 ns

If page you are searching is present in TLB,
EAT = 20 ns + 100 ns = 120 ns.

If the entry is NOT present in TLB then,
EAT = 20 ns + 100 ns + 100 ns = 220 ns.

→ called "TLB hit".

- Now if 80% result in TLB hit & 20% miss.

$$EAT = 0.8 \times 120 + 0.2 \times 220$$
$$= 96 + 44 = \boxed{140 \text{ ns}}$$

(without TLB at all, EAT = 200 ns).

- $p$ = TLB hit rate

$$\boxed{EAT = p(t+m) + (1-p)(t+2m)}$$

for $l$ level paging,

$$EAT = p(t+m) + (1-p)((1+l)m + t)$$

Gate 08 A paging scheme uses TLB. A TLB access takes 10 ns and main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% & there is no page fault?

a) 54 b) 60 c) 65 d) 75

Sol: $0.9(10+50) + 0.1(10+100)$
$= 0.9 \times 60 + 0.1 \times 110$
$= 54 + 11 = 65$ [C]

eg.

| | TLBA | MA | LTBH | PT levels | EMAT |
|---|---|---|---|---|---|
| ① | 20 ns | 100 ns | 80% | 1 | 140 ns |
| ② | 20 ns | 100 ns | 80% | 2 | 160 ns |
| ③ | 20 ns | 100 ns | 80% | 3 | 180 ns |
| ④ | 20 ns | 100 ns | 90% | 1 | 130 ns |
| ⑤ | 20 ns | 100 ns | 60% | 1 | 160 ns |
| ⑥ | 20 ns | 100 ns | 50% | 1 | 170 ns |

Assume no page faults

Sol: ① $0.8 \times 120 + 0.2 \times 220$
$= 96 + 44 = 140$ ns

② $0.8 \times 120 + 0.2 \times 320$
$= 96 + 64 = 160$ ns

③ $0.8 \times 120 + 0.2 \times 420$
$= 96 + 84 = 180$ ns

④ $130 = x \times 120 + (1-x) 220$
$\Rightarrow 100x = 220 - 130 = 90$

⑤ $160 = 0.6 \times (100+x) + 0.4 \times (200+x)$
$x = 160 - 60 - 80 = 20$

⑥ $2 \times 170 = x + 20 + 2x + 20$
$x = \frac{340 - 40}{3} = 100$

## Summary on TLB

- TLB access = t  main memory AT = m
  TLB miss rate = p

EMAT = p(t+m+m) + (1-p)(t+m)

Effective memory access time

= p(t+2m) + (1-p)(t+m)

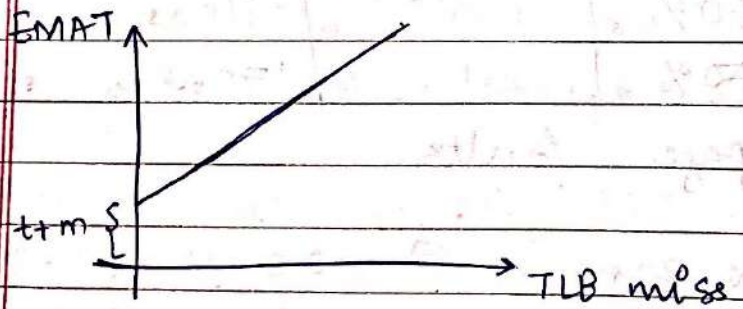
= ~~pt~~ + 2pm + t + m - ~~pt~~ - pm

= pm + t + m

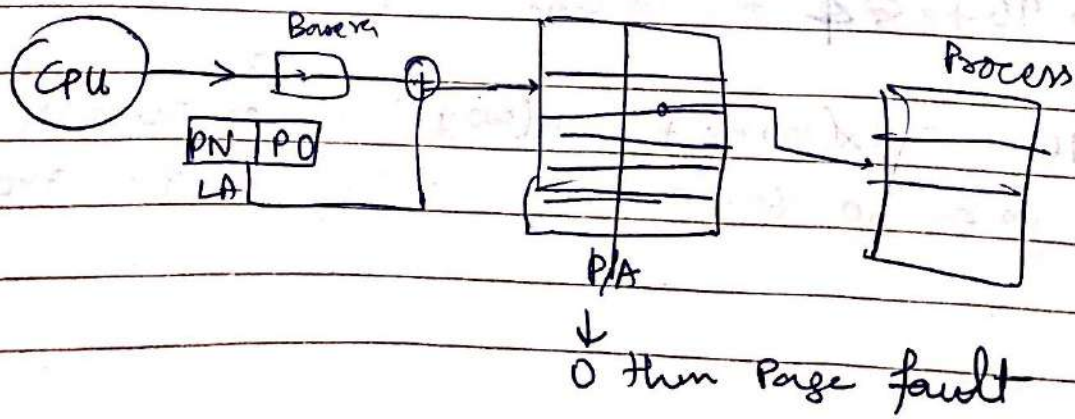
EMAT = t + m + pm

TLB time + memory fetch + miss rate × memory access

0 y = (c) + xa




Demand paging :- We do not load the page until it is required.




↓

0 then Page fault

Page fault :-

When page fault occurs, page is to be loaded to MM from HDD.

memory access time - (ns)

Context switching time - (HS)

Copy (HDD → MM) - (ms)

Context switch - (HS)

Thrashing :-

When lots of page faults ( # page faults = # memory references) i.e. for every memory reference, there is a page fault.

To eliminate thrashing, come up with a better guess ~~what~~ what is the requirement in future

EMAT = p (Service time + MAT) + (1-p) (MAT)

- p : Page fault rate
- Service time : Page fault time
- (1-p) : No page fault
- MAT : memory access time

Gate 11 Let the page fault service time be 10 ms in a computer with avg. MAT being 20 ns. If one page fault is generated for every $10^6$ memory accesses, what is effective access time for the memory

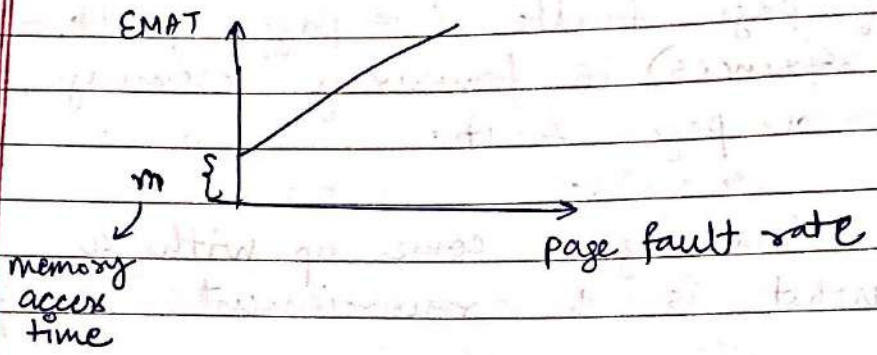
A) 21 ns B) 30 ns C) 23 ns D) 35 ns

Sol. $\frac{1}{10^6} \times 10 \times 10^6 \text{ ns} + 20 \text{ ns}$

$\simeq 30 \text{ ns}$ (B)

- EMAT = $p(PS + m) + (1-p)(m)$
  $= p(PS) + pm + m - mp$
  EMAT $= p(PS) + m$

  $y = mx + c$




Gate 98 If an instruction takes 'i' microseconds and a page fault takes an additional 'j' μs, the effective instruction time if on the average a page fault occurs every k instructions is:-

A) $i + (j/k)$ B) $i + j * k$

C) $(i+j)k$ D) $(i+j) * k$

Sol. $i + \frac{1}{k}(j)$ (A)

Gate 00 Suppose the time to service a page fault is on average 10 ms, while a memory access takes 1μs. Then a 99.99% hit ratio results in average memory access time of

A) 1.09999 ms B) 1ms C) 9.9999 μs D) 1.999 μs

Sol. $99.99$ $\left|\begin{matrix}+\\10\end{matrix}\right|$ $\frac{0.01}{100} \times 10 \times 10^3$ μs

$10 \times \frac{0.01}{100} \times 10^3$ μs 1μs (2μs)

IT

Gate.07 A demand paging system takes 100 time units to service a page fault and 300 time units to replace a dirty page. Memory access time is 1 time unit. The probability of a page fault is 'p'. In case of page fault the probability of page being dirty is also 'p'. It is observed that the average access time is 3 time units. Then value of p is

A) 0.194 B) 0.233 C) 0.514 D) 0.981

Sol: $3 = (1-p) \times 1 + p(1-p) \times 100 + p^2 \times 400$

$\Rightarrow 3 = 1 - p + 100p - 100p^2 + 400p^2$

$\Rightarrow 300p^2 + 99p - 2 = 0$

$\Rightarrow p = \frac{-99 + \sqrt{99^2 + 8 \times 300}}{2 \times 300}$

$= \frac{-99 + 110.46}{600} \simeq 0.01909$

$\simeq 1.91\%$

PS $= (1-p) \times 100 + p \times 300$

$= 100 + 200p$

EMAT $= 1 + 100p + 200p^2 = 3$

$\Rightarrow 200p^2 + 100p - 2 = 0$

$\Rightarrow 100p^2 + 50p - 1 = 0$

$\Rightarrow p = \frac{-50 + \sqrt{2500 + 400}}{200}$

$= \frac{3.85}{200} = 0.019 \rightarrow$ (A)

wrong answer in options

Gate 03 CS

A processor uses 2 level paging. VA = PA = 32 bits.
Byte addressable

VA

| 10 | 10 | 12 |
|---|---|---|

PTE = 4 Bytes

TLB has hit ratio of 96%. Cache has hit ratio of 90%. Main memory access time is 10 ns, cache access time is 1ns and TLB access time is 1ns.

Q.1 Assuming no page faults, the average time taken to access a VA is approx. (to nearest 0.5ns)

A) 1.5ns B) 2ns C) 3ns D) 4ns ✓

Sol: $0.96 \times 1 + 0.04 \times 0.9 \times (10+1) + 0.04 \times 0.1 \times (1+1+10)$

$0.96 \times 1 + 0.04 \times 0.96 \times 2 + 0.04 \times 0.04 \times 0.1 \times$

Avg. time taken to access VA = (VA → PA) + (Fetch the word from process)

$= t + (1-p_t)(k * m) + c + (1-p_c)\, m$

$= 1 + 0.04 \times (2 \times 10) + 1 + 0.1 \times 10$

$= 1 + 0.8 + 1 + 1$

$= 3.8$ ns (D)

Q.2: Suppose a process has the following ~~data~~ pages in virtual address space: two contigous code pages starting at virtual address 0X00000000, two contigous data pages starting at virtual address

0x00400000 and a stack page at virtual address 0x FFFFF000. The amount of memory required for storing page tables for this process:-

A) 8 KB  B) 12 KB  C) 16 KB  D) 20 KB

Sol. Size of page = 4 KB  # pages = $2^{20}$

PTE = 4 B, PTS = 4 MB

2 levels of paging

# pages in MM = 1 + 1 + 1 = 4
(Outer 1 of data, 2 of code, 1 of stack)

2 pages: 0x00000000 to 0x00000FFF; 0x00001000 to 0x00001FFF

in one page (some ∓ 10 bits).

Total mem. req = 4 × 4 KB = 16 KB (C)

Gate04 Consider a system with 2-level paging scheme in which a regular memory access takes 150 ns and servicing a page fault takes 8 ms. An average instruction takes 100 ns of CPU time and two memory accesses. The TLB hit rate is 90% and page fault rate is one in every 10,000 instructions. What is the effective average instruction exec. time?

A) 645 ns  B) 1050 ns  C) 1215 ns  D) 1230 ns

Sol. ~~EAT~~

EMAT = ~~0.6*1*~~ (VA → PA) + (Access the byte from PA)

$= t + p_{tm}(k \times m) + m + p_{pf}(S_J)$

$= 0 + 0.1 \times 2 \times 150 + 150 + \frac{1}{10000} \times 8 \times 1000000$ ns

$= 30 + 150 + 800$

$= 980$ ns

Avg. inst time = $100 + 2 \times 980$ ns

$= 2060$ ns

Gate 14 Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in physical memory. It takes 10 ms to search the TLB and 80 ms to access physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in ms) is (122)

Sol. ~~80 + 0.1*~~ $10 + 80 + 0.6 \times 80$ ms

$= 90 + 48$ ms $= 138$

$10 + 80 + 0.4 \times 80$ ms

$= 90 + 32 = \boxed{122}$ ms

## Inverted Page Table

let If there are 10 processes in system, for each process we should have 10 page tables in memory.

↓ Problem

Takes up much of space

↓ Solution

# pages in system at any time = # frames in main memory

(Inverted page table)

PT of P1

| | |
|---|---|
| P0 | x |
| 1 | f1 |
| 2 | x |
| 3 | f2 |
| 4 | x |
| 5 | f3 |
| 6 | x |

PT of P2

| | |
|---|---|
| P0 | x |
| P1 | x |
| P2 | f4 |
| P3 | f5 |
| P4 | f6 |
| P5 | x |
| P6 | x |

Inverted PT

| | |
|---|---|
| f0 | |
| 1 | P1 ⇐ P1 |
| 2 | P3 ⇐ P1 |
| 3 | P5 ⇐ P1 |
| 4 | P2 ⇐ P2 |
| 5 | P3 ⇐ P2 |
| 6 | P4 ⇐ P2 |
| 7 | |
| ⋮ | ⋮ |

Eg PA = 32 bits, Page Size = 4kB, IPTE = 4B

Size of IPT ?

Sol: # frames = $2^{32-12} = 2^{20}$

Size (IPT) = $2^{20} \times 4B$

= 4 MB

# Importance of frame allocation & Page Replacement

virtual memory implementation

## Frame allocation

min frame :-

1) Code Segment
2) 2p Data Seg.
3) Stack page

4 pages (example only)

If we don't allocate frames appropiately, "trashing" occurs.

max. req :- Size of process itself.

Strategies

### Equal allocation

30 Frames
3 process

$\frac{30}{3} = 10$ f/proc

### Weighted allocation

10 frames
3 processes
$P_1$ = 20 pages
$P_2$ = 30 pages
$P_3$ = 50 pages

$F_1 = \left(\frac{20}{100}\right) \times 10F$

$= 2F$

$F_2 = 3F$

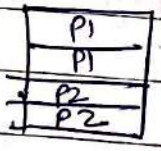
$F_3 = 5F$

Static methods

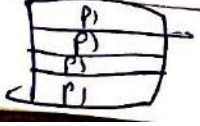
### Dynamic allocation

"Priorities"

In start

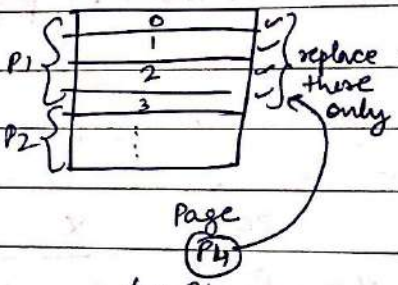



Now if P1 wants more frames & has higher priority it can expand




## Page Replacement

### Local (For static frame allocation)




Replace from frames allocated to that process only

### Global

It considers all pages for replacement even if it is not allocated for that process or NOT.

**Demand Paging :-** We do not load any page in the frame until it is required.

Reference string :- Requests of pages by a process in need order. eg 0 1 4 5 7 1 6 7 ...

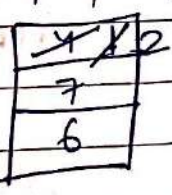
Page numbers (order of need)

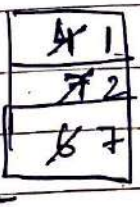
## Page Replacement Algorithms

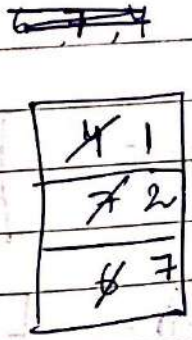
- Optimal page Replacement algorithm :- Replace the page which will not be referred for longest. (Not practical to implement) (used for benchmarking only).
- LRU :- Replace the least recently used page (page which has not been referred for a long time).
- FIFO :- First one added to main memory is replaced

eg: Gate 14. Demand paging is used, find the page faults :-

Gate 14 4, 7, 6, 1, 7, 6, 1, 2, 7, 2 , F = 3




optimal :- # PF = 5

LRU :- # PF = 6

FIFO :- # Pf = 6

Gate 95: 0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370

100 records per page with 1 free main memory frame.

Sol: Ref. string :- 01, 02, 04, 04, 05, 05, 05, 01, 02, 02, 02, 03, 03

- Since there is only 1 free frame, no matter whatever algorithm used, # page faults will be same.

| 1 | 2 | 4 | 5 | 1 | 3 |
|---|---|---|---|---|---|

∴ # page faults = 7

$$\text{miss rate or page fault rate} = \frac{\#\text{ page faults}}{\#\text{ total references}} = \frac{7}{13}$$

$$\text{also, hit rate} = 1 - \text{miss rate} = \frac{6}{13}$~~Gate 07~~

Gate09.

| 1 | 2 | 3 | 2 | 4 | 1 | 3 | 2 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |

F = 3

optimal :-

| 1 | |
|---|---|
| 2 | 4 |
| 3 | 2 |

# faults = 5 PFR = 5/10

LRU :-

| 1 | 4 | 2 |
|---|---|---|
| 2 | 3 | 1 |
| 3 | 1 | 4 |

# faults = 9 PFR = 9/10

FIFO :-

| 1 | 4 |
|---|---|
| 2 | 1 |
| 3 | 2 |

# faults = 6 PFR = 6/10

Next Kanal
Date
Page 124

Gate 07

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

F = 3

optimal :-

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 6 |
| 3 | 3 | 3 | 4 | 5 | 6 | 6 |

# faults = 7

LRU :-

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 7 | 4 | 5 | 6 | 6 | 6 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 |

# faults = 9

FIFO :-

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 7 | 4 | 5 | 6 | 6 | 6 |
| 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 1 |
| 3 | 3 | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 |

# faults = 9

Gate 14

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

F = 3

optimal :-

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 4 | 4 | 4 | 5 | 3 | 3 | 4 | 4 |

# faults = 7

LRU :-

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 3 | 3 | 4 | 4 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6 |
| 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 2 | 2 | 2 |

# faults = 10

FIFO :-

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 4 | 4 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 6 |
| 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 2 | 2 | 2 |

# faults = 10

# Belady's Anamoly

Q: 0 1 2 3 0 1 4 0 1 2 3 4

- **Optimal :-**

① 3 frames

| | |
|---|---|
| ~~0~~ | 2 |
| ~~1~~ | 3 |
| ~~2~~ | ~~3~~ 4 |

# faults = 7

② 4 frames

| | |
|---|---|
| ~~0~~ | 3 |
| 1 | |
| 2 | |
| ~~3~~ | 4 |

# faults = 6

# frames ↑ ⇒ # faults ↓

- **LRU :-**

① 3 frames

| | |
|---|---|
| ~~0~~ | ~~3~~ ~~4~~ 2 |
| ~~1~~ | 0 3 |
| ~~2~~ | 1 4 |

# faults = 10

② 4 frames

| | |
|---|---|
| ~~0~~ | 4 |
| 1 | |
| ~~2~~ | ~~4~~ 3 |
| ~~3~~ | 2 |

# faults = 8

# frames ↑ ⇒ # faults ↓

- **FIFO :-**

① 3 frames

| | |
|---|---|
| ~~0~~ | ~~3~~ 4 |
| ~~1~~ | 0 2 |
| ~~2~~ | 1 3 |

# faults = 9

(2) 4 frames

| | |
|---|---|
| 0 | 4 3 |
| 1 | 0 4 |
| 2 | 1 |
| 3 | 2 |

# faults = 10

Here, # frames ↑ ⇏ # faults ↓

instead it increases

↓

Called "Belady's anamoly".

Algorithms which show this behaviour are called "stack algorithms".

- Optimal & LRU will never show belady's anamoly. For FIFO, for some examples, this is shown.

## Stack Algorithms

| Reference string | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Using optimal (3 frames) | | | | | | | | | | | | |
| Memory | 0 | 0 1 | 0 1 2 | 0 1 3 | 0 1 3 | 0 1 3 | 0 1 4 | 0 1 4 | 0 1 4 | 2 1 4 | 3 1 4 | 3 1 4 |

For this algorithm, always, the pages present when we had 3 frames are also present when we have 4 frames.

Pages (3F) ⊆ Pages (4F)

This property is called "stack property".

Pages (m) ⊆ Pages (m+1)

- optimal algorithm follows stack property.

- Whenever a algorithm is "stack algorithm", it does not fall into belady's anamoly.

Using FIFO:-
(3 & 4 frames)

| Ref string | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 0 | 0 0 | 0 0 | 3 0 | 3 0 | 3 0 | 4 4 | 4 4 | 4 4 | 4 4 | 4 3 | 4 3 |
| | | 1 1 | 1 1 | 1 1 | 0 1 | 0 1 | 0 1 | 0 0 | 0 0 | 2 0 | 2 0 | 2 4 |
| | | | 2 2 | 2 2 | 2 2 | 1 2 | 1 2 | 1 2 | 1 1 | 1 1 | 3 1 | 3 1 |
| | | | | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |

Does not follow stack property

- Thus FIFO is not stack algorithm and hence can show belady's anamoly.

Using LRU:-
(3 & 4 frames)

| Ref String | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 0 | 0 0 | 0 0 | 3 0 | 3 0 | 3 0 | 4 0 | 4 0 | 4 0 | 2 0 | 2 0 | 2 4 |
| | | 1 1 | 1 1 | 1 1 | 0 1 | 0 1 | 0 1 | 0 1 | 0 1 | 0 1 | 3 1 | 3 1 |
| | | | 2 2 | 2 2 | 2 2 | 1 2 | 1 4 | 1 4 | 1 4 | 1 4 | 1 3 | 4 3 |
| | | | | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |

- LRU is stack algorithm & thus does not show belady's anamoly.

Gate 94 A memory page containing a heavily used variable that was initialized very early and is in constant use is removed when ___ is used

a) LRU b) FIFO c) LFU d) None.

Ans (B)

Gate 07 Consider the following statements.

P: Increasing the number of page frames sometimes increase page faults in FIFO.

Q: Some programs do not exhibit the locality of reference.

Sol. P ✓ Q ✓ correct reason X

Gate 10 A system uses FIFO policy for page replacement. It has 4 page frames with no pages loaded to begin with. The system first accesses 100 distinct pages in some order then accesses the same 100 pages but now in reverse order. How many page faults will occur?

a) 196 b) 192 c) 197 d) 195

Sol.

| | |
|---|---|
| 0 | 4 |
| 1 | 5 |
| 2 | 6 |
| 3 | 7 |

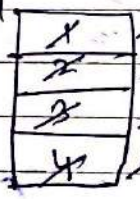
... 96 95 ... 3
... 97 94 ... 2
... 98 93 ... 1
... 99 92 ... 0

100 + 96 = 196 (A) A

## Some interesting behaviour of optimal page replacement

eg. 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6

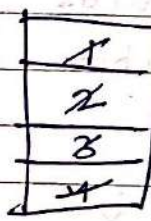
optimal



All the times, most recently used page has been replaced, violating the principle of locality of reference.

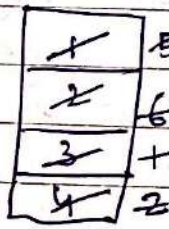
# pf = 12

MRU algorithm



Most recently used strategy works same as the optimal solution.

LRU



Worst possible strategy

# PF = 24

"causing Trashing"

also, FIFO ~~always~~ also works same as LRU (worst possible)

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

optimal

MRU behaves as worst case (causes trashing)

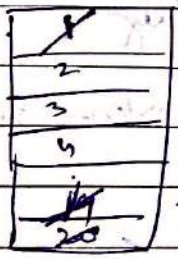
behaving like LRU algorithm

also FIFO will behave same

Gate14 A computer has 20 physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, ... 100 in that order, and repeats the access sequence thrice. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacent for this program?

a) Least recently used
b) FIFO
c) Last-in-first-out
d) MRU

Sol.



# PF = 100 + (100 - 19) + (100 - 19)

~~MRU~~ ~~(D)~~

# PF for all options = 300

MRU & LIFO is answer if pure "demand paging"

(C) & (D) → Q. wrong

## Working Set Algorithm

- The nearest future is a closed approximation of the recent past.

eg: 1 2 3 1 2 4 1 4 2 1 5 1 2 4 2 1

A = 4 (window size)

Working Set

W = {1, 2, 3}

W = {1, 2, 3}

W = {1}

W = {1, 2}

# frames ↑ or ↓

window is max # frames given to a process.

1 2 3 1 2 4 1 4 2 1 5 1 2 4 2 1

A = 4

1. w = {1}
2. w = {1, 2}
3. w = {1, 2, 3}
3. w = {1, 2, 3}
3. w = {1, 2, 3}
4. w = {1, 2, 3, 4}
3. w = {1, 2, 4}
3. w = {1, 2, 4}
3. w = {1, 2, 4}
3. w = {1, 2, 4}
4. w = {1, 2, 4, 5}
3. w = {1, 2, 5}
3. w = {1, 2, 5}
4. w = {1, 2, 4, 5}
3. w = {1, 2, 4}
3. w = {1, 2, 4}

avg frame req. = $\frac{48}{16} = 3$

Finding window size ? → Experimentally

PF dec. Algo :- when PF ↓, ↓ the working set (window)

eg

Gradeage | c c d b c e c e a d

w = {a, d, e}

A = 4

w = {a, d, e} 3
w = {a, d, e, c} 4.
w = {d, e, c} 3
w = {d, e, c} 3.
w = {b, c, d} 3.
w = {c, b, d} 3.
w = {b, c, d, e} 4
w = {b, c, e} 3.
w = {c, e} 2.
w = {a, c, e} 3.
w = {a, e, c, d} 4.

avg frame req. = $\frac{32}{10} = 3.2$

# Page Replacement Algorithms Implementation

Optimal:- Cannot be implemented (We cannot look into future).

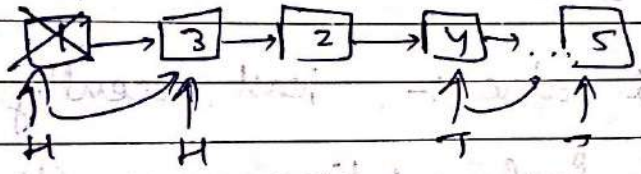
Not Recently Used :-

Reffered bit (R)

clock period.

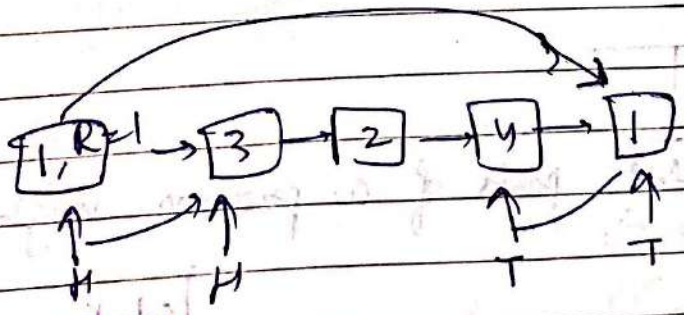
↳ If a page is reffered in this clock, R = 1

FIFO :- Maintain a queue (linked list)




Second chance :- Advancement to FIFO

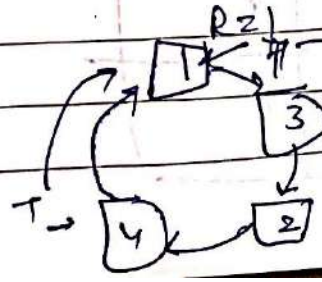
- Follow FIFO but check R bit, if R=1, give it one more chance (make R=0) and do not replace it



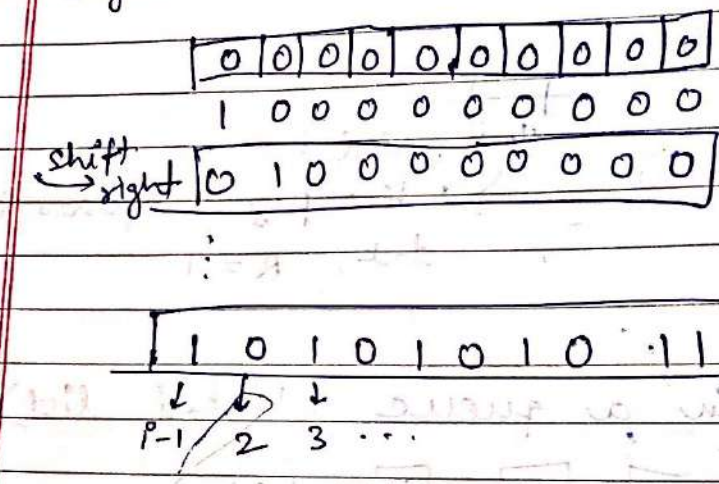

Clock Page Replacement :- Modification to second chance

- maintain circular list

Just move Head & Tail.



LRU :- Using 1 bit (R) we can only say that the page has been referred in last cycle OR Not, but for LRU, we should have multiple bits, say 10 bits




Small value :- least recently used & replace it

- Most of implementations use 64 bits to represent this

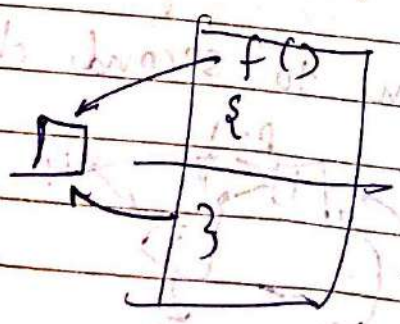
LFU :- associate a counter with every page.
Whenever page reference occurs, ↑ the counter.
The one with least counter → LFU. & replace it.

## Segmentation

- Two related parts of a process might fall in different pages

↓ solution

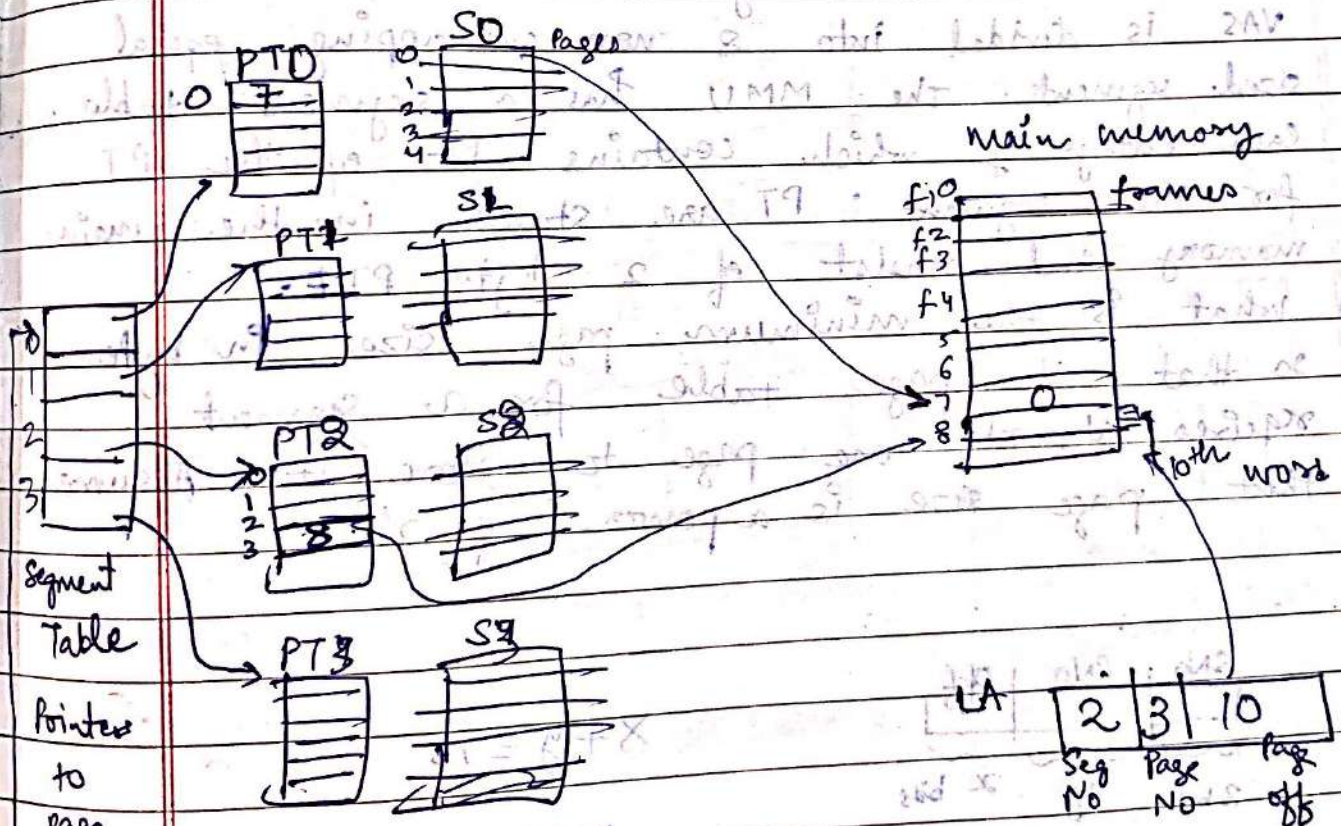
divided into segments

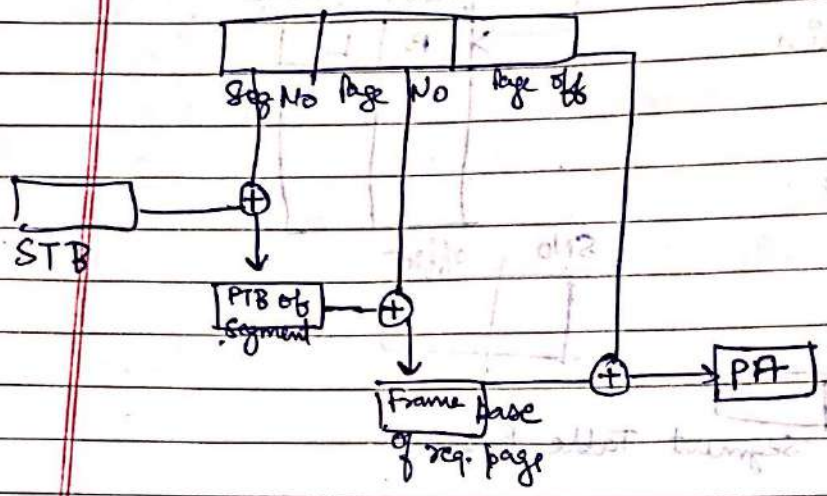






## Segmented Paging

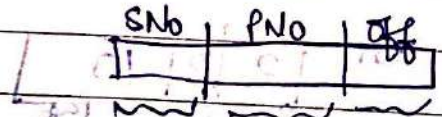





Gate99 PAS = VAS = $2^{16}$ B. Byte addressable.
VAS is divided into 8 non-overlapping equal sized segments. The MMU has a segment table, each entry of which contains PA of the PT for the segment. PT are stored in the main memory and consist of 2 byte PTE.

(a) What is the minimum page size in bytes so that the page table for a segment requires at most one page to store it. Assume that page size is a power of '2'.

Sol.




$2^y$ pages $2^x$ ~~part~~

page size

$x + y = 13$

PTS = No of entry × Size of PTE

$2^x = 2^y \times 2$

$x = y + 1$

$x = 7, \; y = 6$

page size = $2^7$ B = **128 B**

# 5. File Systems ; IO & Protection



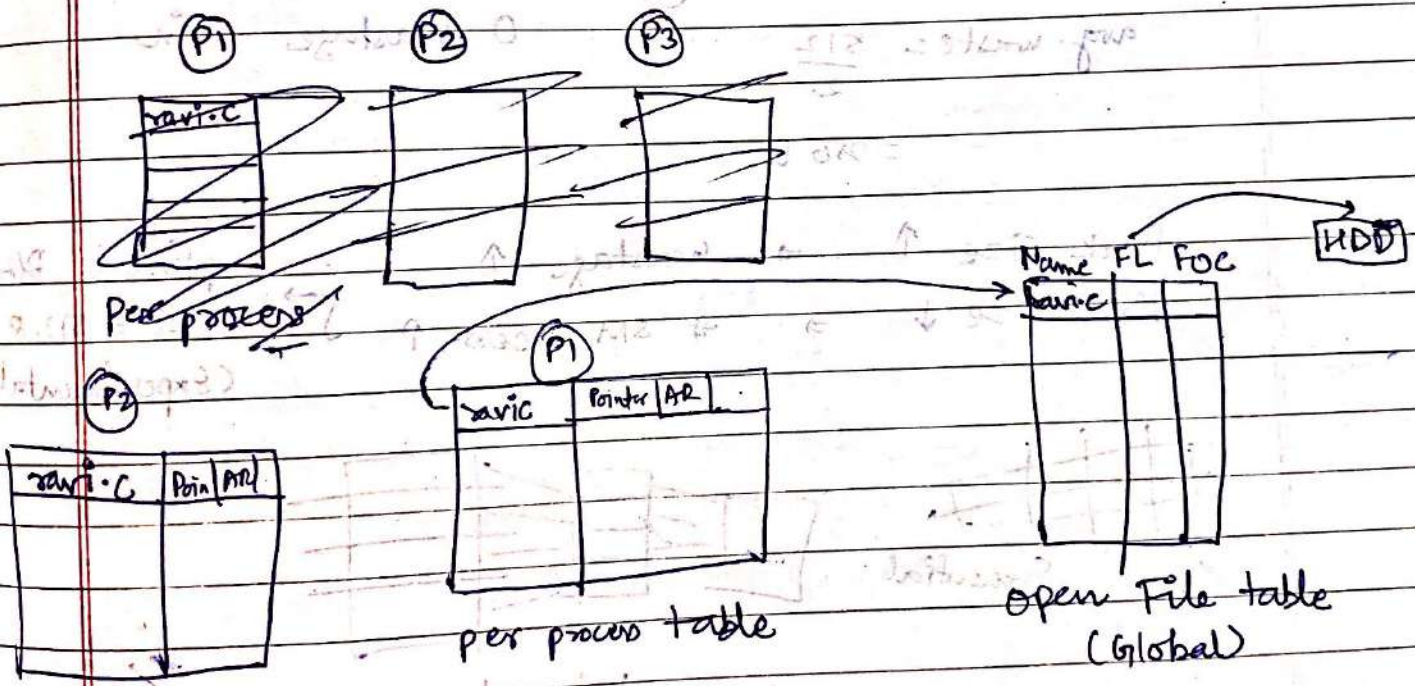
## File attributes

- Name
- Identifier
- Type
- Location
- Size
- Protection
- Time & date.

## File operations

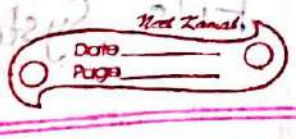
- Creating
- Writing
- Reading
- Repositioning
- Deleting
- Truncating.

## Open File Tables




## Information required for accessing a file :-

1. File pointer
2. File open count
3. Disk location of file
4. Access Rights




## Accessing Files

Access Methods

→ Sequential access
→ Direct access
→ Indexed access

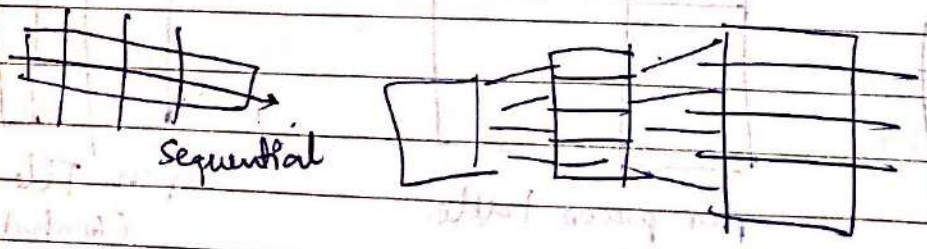
512 B block

512
512
512
1 → 511 wastage max
0 wastage min.

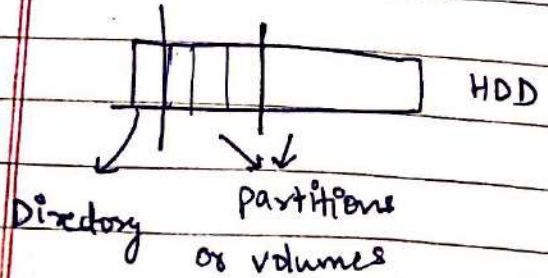
avg. waste = $\frac{512}{2}$

= 256 B

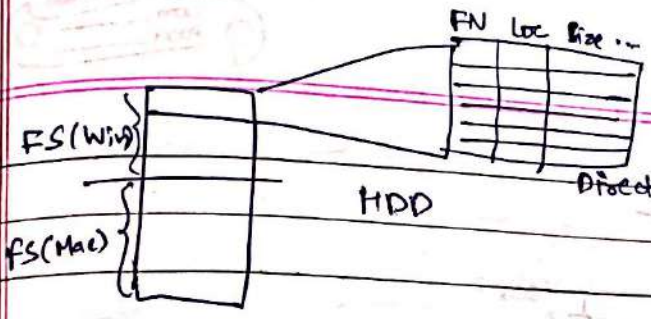
Block size ↑ ⇒ wastage ↑
Block size ↓ ⇒ # SM access ↑
} → optimal Block Size = 512 B (Experimentally)




## Directory Structure

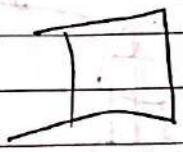



Directory :- Translation tables translates file name into all the info required to access it




- Every file system is supposed to have at least one directory (containing info about all other files in that file system)

operations a directory should support :-



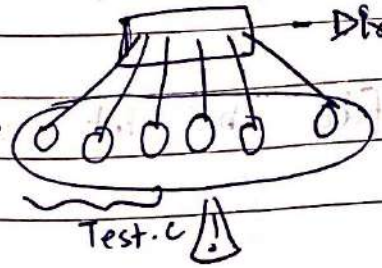
- Search
- Create
- Delete
- List
- Traverse
- Rename

- When millions of files are there, it is difficult to keep information about all the files in a single directory

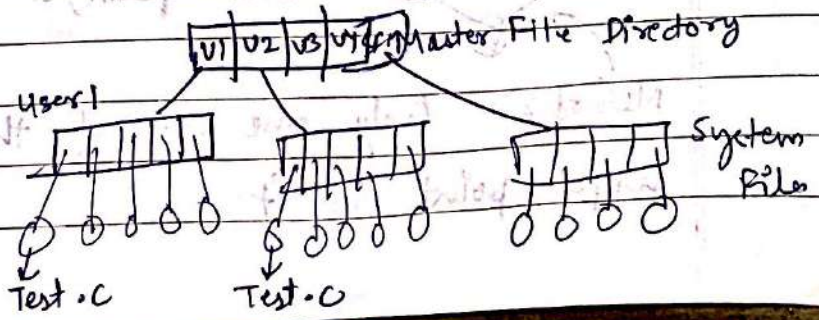
↓ solution

- Multi-level directory
  - 2 level
  - Tree
- Acyclic graph
- Graphs

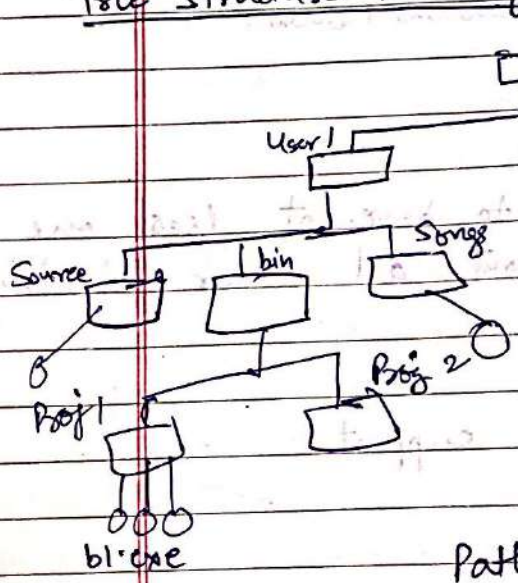
Single Level




Two level



## Tree Structure Directory




Path :- File is present

Path Name (b1.exe) :-

root/user1/bin/proj1/b1.exe

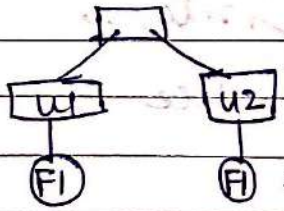
Path Name
- Absolute (write everything)
- Relative (wrt something else)

Relative :-

pwd :- root/user1

↳ Then user does not specify these two while opening the file.

## Acyclic Graph Structured Directory

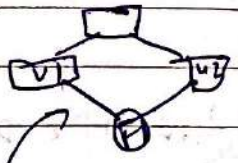



← Share F1 to both users

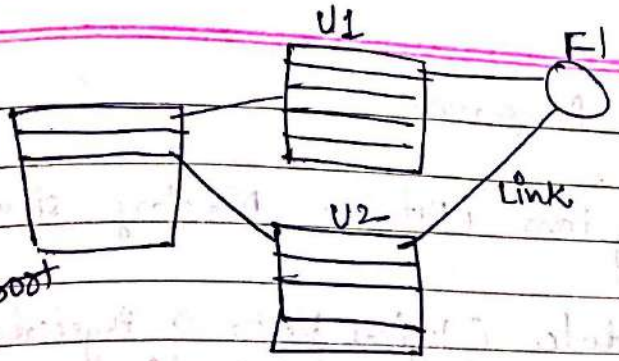
Method 1 :- Copy to both of their directory

↓ Problem

When update, check for copy & update that too.




Method 2 :- Only one copy of the file but let many people point to it.




Problem :- ① File deletion (Dangling Pointer) → solution maintain a count of # references
② Different path names

## File Systems

File system provides the mechanism for on-line storage and access to file contents., including data and programs.

File system deals with following issues :-
→ File structure
→ To allocate diskspace
→ Recovering freed space
→ To track the locations of data
→ Interface other parts of operating system to secondary storage

### File System Structure

→ File systems provide efficient and convenient access to the disk by allowing data to be stored, located and retrieved easily.

Application Programs

⇓

Logical File System (Metadata, Directory structure)

⇓

File Organization module (logical Block ⇒ Physical Block)
(Free space Management)

⇓

Basic File System (Commands to the I/O control buffering)

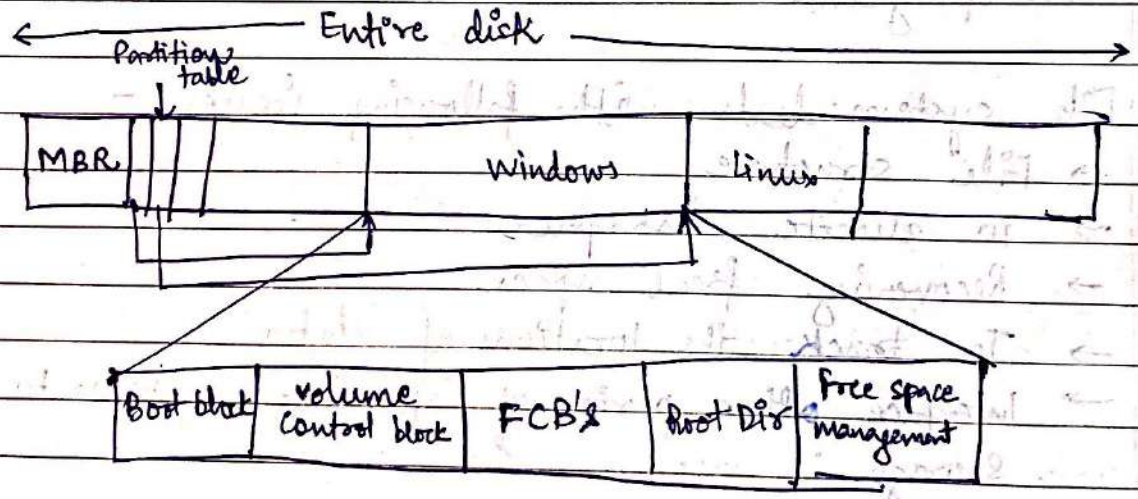
⇓

I/O control (Consists device drivers, Interrupt handlers)

⇓

Devices

Master Boot record (MBR):-




BIOS:- 1st block of ROM contains MBR

On disk data structure used in File System Implementation

- Several in memory & on disk structures are used to Implement a file system. These structures

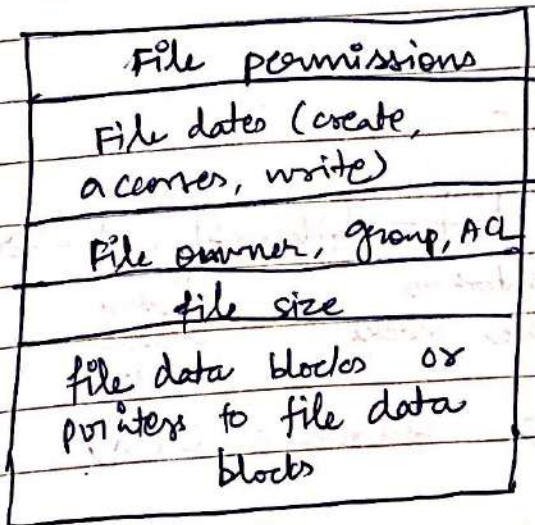
vary depending upon the OS and the file system; but general principles apply.

Boot Control Block :- (Per volume):- It contains information needed by the system to boot an OS from that volume. In UNIX file system it is called the boot block. In NTFS it is called the partition block sector.

Volume Control Block :- It contains volume details such as the number of blocks in the partitions, size of each block
In UFS, it is called super block.
In NTFS, this informations stored in master file table.

Directory structure per file system :- It is used to organize the files
In UFS. this includes file names & associated inode numbers.

File Control Block (FCB) :- It contains details about the file




FCB

## In Memory Data Structures used in File System Implementation

The in-memory is used for both the file system management & performance improvement via caching. This data loaded at mount time and discarded at dismount.

① In memory mount table :- It contains the information about each mounted volume.

② In memory directory structure cache :- It holds the directory information of recently accessed directory.

③ System-wide open file table :- It contains the FCB of each open file.

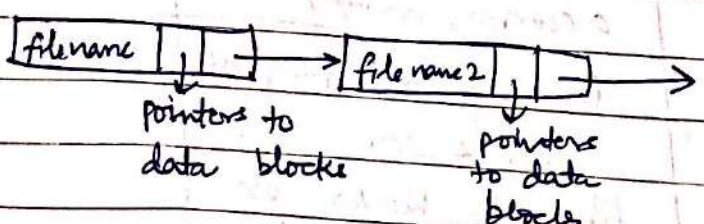
④ Per process open file table :- It contains the pointer to the appropiate entry in the system wide open file table.

## Directory Implementation

→ The selection of directory allocation & directory management algorithms significantly affects the performance and reliability of file system.

Algorithms :-

1. Linear List :-




Problem :- Search = O(n)

2. Hash Table :-

H (Name) = pointer in linked list

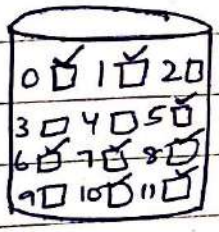
Search or open = O(1)

Disadv :- Fixed size, collisions, etc...

## Allocation Methods

→ How to allocate space to the files so that disk space is utilized efficiently and files can be accessed quickly.

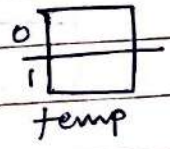
### Contigous allocation :-



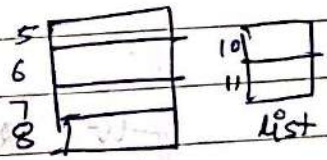

HDD

directory

| file | start | leng |
|---|---|---|
| temp | 0 | 2 |
| tr | 5 | 4 |
| list | 10 | 2 |




temp

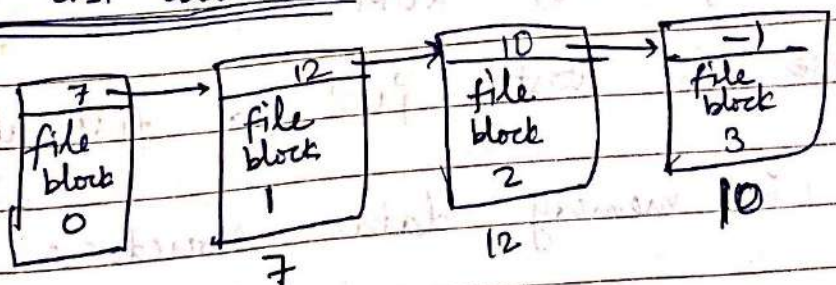



tr list

Advantages :- Simple to implement, read performance is excellent.

Disadvantages :- This disk becomes fragmented. (Both external & internal)

### Linked list allocation :-



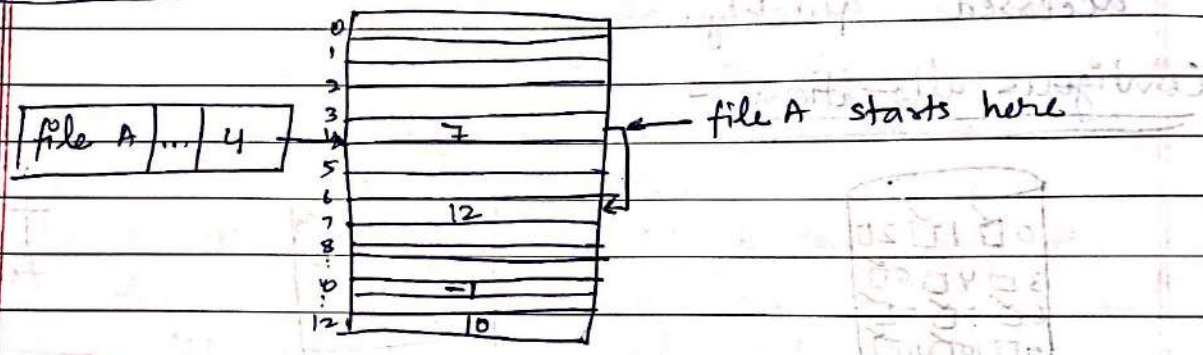

Physical Block

File A

Adv :- → Unlike contigous allocation, every disk block can be used.
→ No space lost due to disk fragmentation (except for internal fragmentation in the last block of each file).

Disadv :- → Random access is very slow.
→ Pointer takes up few bytes.

File allocation table :-




File allocation Table

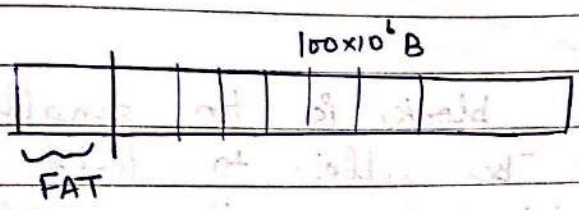
Adv :- Random access is easier
Disadv :- Size might be too big that it could consume lot of memory.

eg Capacity = 20 GB, Block size = 1 KB
the # blocks = 20 M → require 20 M entries,
if each entry = 4 B, 20 M × 4 B = 80 MB
80 MB space is wasted just for file allocation tbl.

• This is a in-memory data structure.

Gate14. ~~too~~ A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4B in size. Given a $100 \times 10^6$ B disk on which the file system is stored and data block size is $10^3$ B, the max. size of the file that can be stored on this disk in units of $10^6$ Bytes is _____.

Sol.




$$4x + 10^3 \times x = 100 \times 10^6$$

$$x = \frac{100 \times 10^6}{(10^3+4)} = \frac{100 \times 10^3}{} = 10^5$$

$$FAT_{size} = 0.4 \times 10^6 B$$

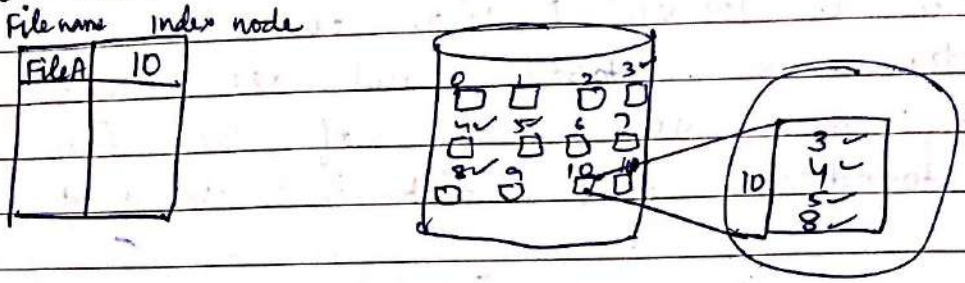
$$\text{Space available}_{\text{without FAT}} = (100 - 0.4) \times 10^6$$

$$= 99.6 \times 16^6 B$$

## Indexed allocation:-

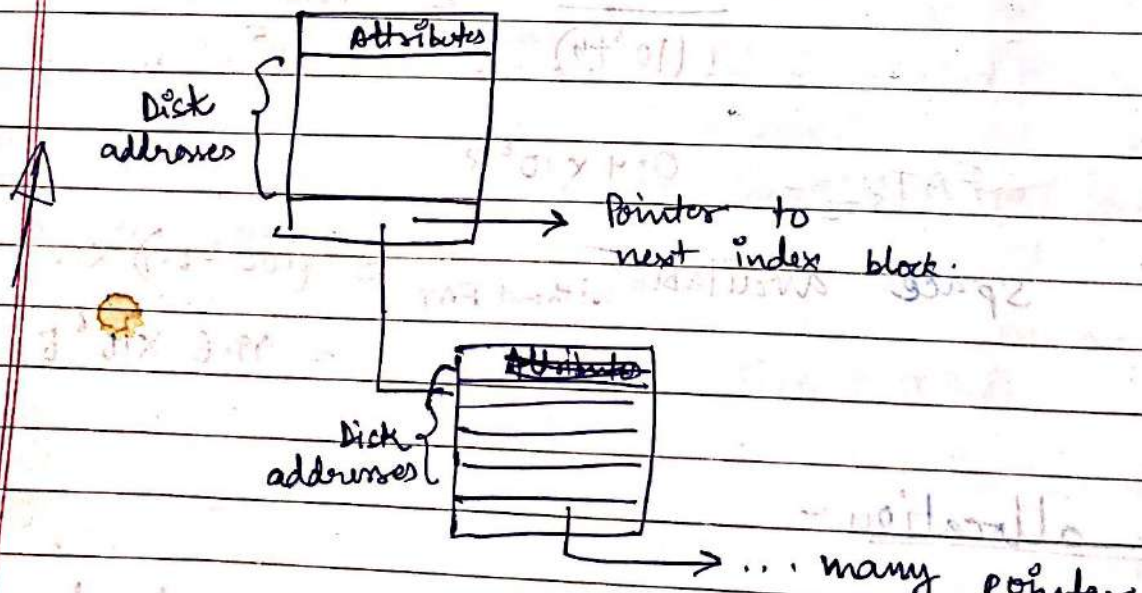
- Linked allocation solves the external fragmentation & size declaration problem of contigous allocation.
- In the absence of FAT, linked allocation cannot support efficient direct access.

→ Indexed allocation solves the problem by bringing all the pointers together into one location: the index block.

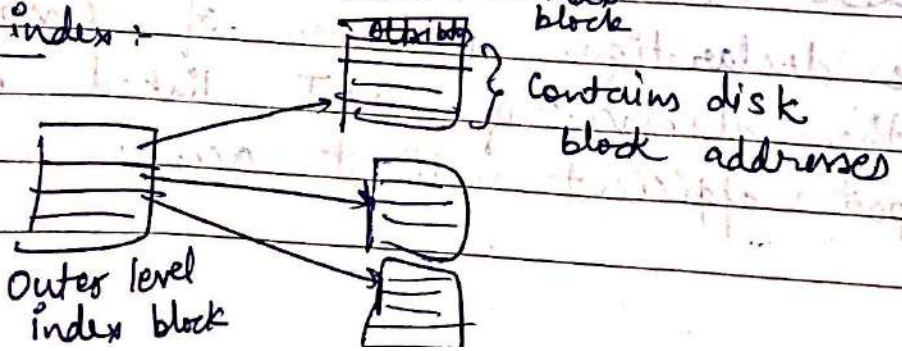



→ If the indexed block is too small, however it will not be able to hold enough pointers to hold for a large files.

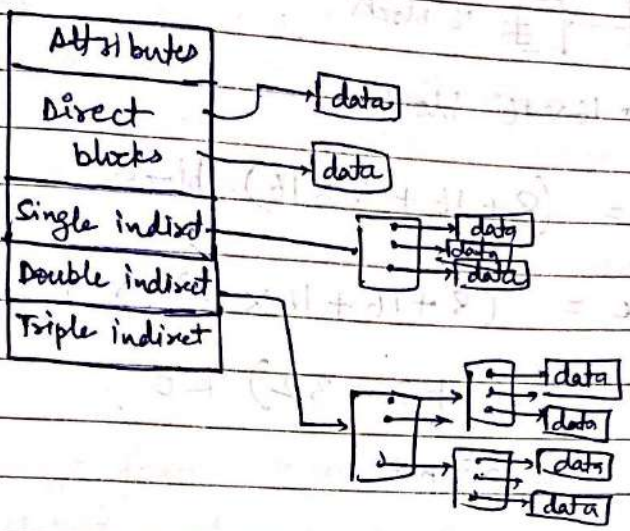
Linked schema :- We can link several index blocks.




Multi-level index :-



## Combined scheme :-




Gate CS 2004 A unix style i-node has 10 direct pointers and one single, one double and one triple indirect pointers. Disk block size is 1KB, disk block address is 32 bits and 48-bit integers are used. What is the maximum possible file size ?

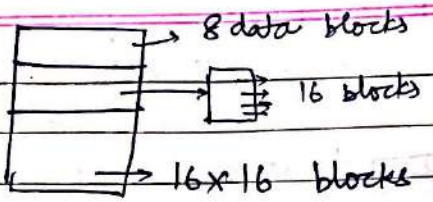
Sol. $\frac{1024\ B}{4\ B} = 170$ pointers

10, 170, 170×170, 170×170×170

$(10 + 170 + 170 \times 170 + 170 \times 170 \times 170) \times 1024\ B$

Gate 12 A file system with 300 GB uses a file descriptor with 8 direct block addres. One indirect block address & one double indirect block address. The size of each disk block is 128 B and the size of each disk block address is 8 B. Max. possible file size in system is ?

Sol:




$$\frac{128\ B}{8\ B} = 16$$

# blocks = $(8 + 16 + 16 \times 16)$ blocks.

Size file = $(8 + 16 + 16 \times 16) \times 128\ B$

$= (1 + 2 + 32)\ KB$

$= 35\ KB$

## FREE SPACE MANAGEMENT

1. Bit Vector :-
   - → The Free space list is implemented as a bit map (or) bit vector.
   - → Each block is represented by 1 bit.
   - → If the ~~bit~~ block is free bit is 1 else it is 0

2. Linked list :-
   - → Another approach to free space management is to link together the free disk blocks, keeping a pointer to the first block in a special location on the disk and caching it in memory.

## Disk Scheduling

→ File systems must be accessed in efficient manner, especially with hard drives, which are slowest part of computer.

→ As a computer deals with multiple processes over a period of time, a list of requests to access the disk build up. For efficiency purposes all requests (from all processes) are aggregated together.
→ The technique that the OS uses to determine which request to satisfy next is called disk scheduling.

FCFS scheduling:-

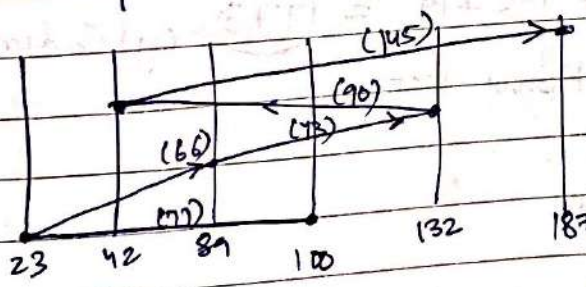
→ Simplest; performs operations in order requested
→ No ordering of work queue.
→ No starvation :- Every request is serviced.

eg: A disk queue with requests for I/O to blocks on cylinders :

23, 89, 132, 42, 187.

with disk head initially at 100; r/w head movement (# cylinders crossed while processing all these requests) = ?

Sol:




77 + 66 + 43 + 90 + 145 = (421) cylinders

• seek time ∝ # cylinders crossed

# Cylinders crossed ↑ ⇒ seek time ↑

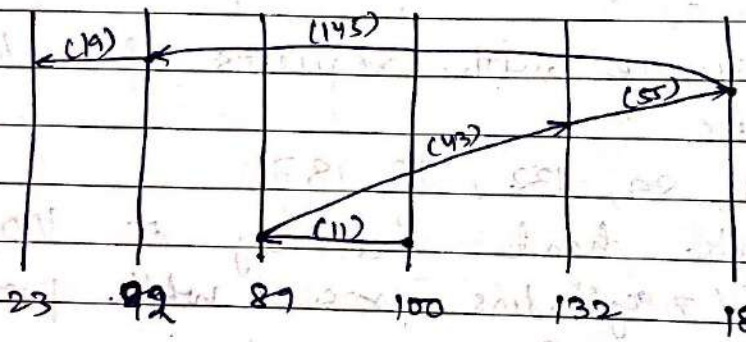
## SSTF scheduling :-

→ Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position regardless of direction.
→ Reduces total seek time compared to FCFS.

Disadv:-

→ Starvation is possible.
→ Switching directions slows things down.
→ Not the most optimal.

eg: 23, 89, 132, 42, 187

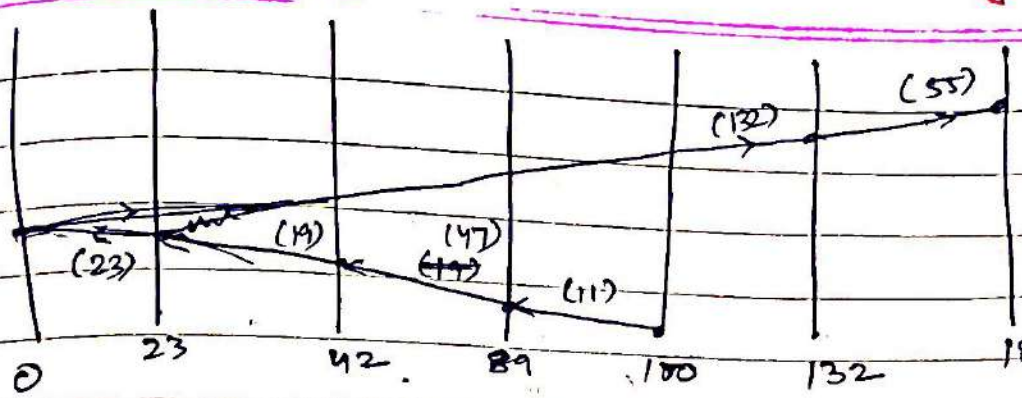



11 + 43 + 55 + 145 + 19 = (273) cylinders
(better than FCFS)

## SCAN (Elevator algorithm)

→ Go from the outside to inside servicing requests and then back from inside to ouside servicing requests.

eg. 23, 89, 132, 42, 187

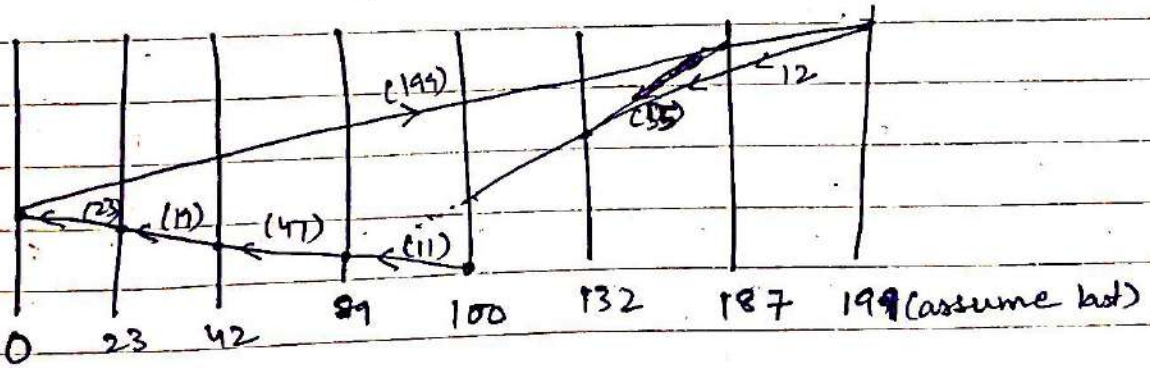



11 + 47 + 19 + 23 + 132 + 55 = (287) cylinders

C-SCAN :-

→ Moves inwards servicing requests until it reaches the innermost cylinder ; then jumps to the outside cylinder of the disk without servicing any requests.

eg. 23, 89, 132, 42, 187

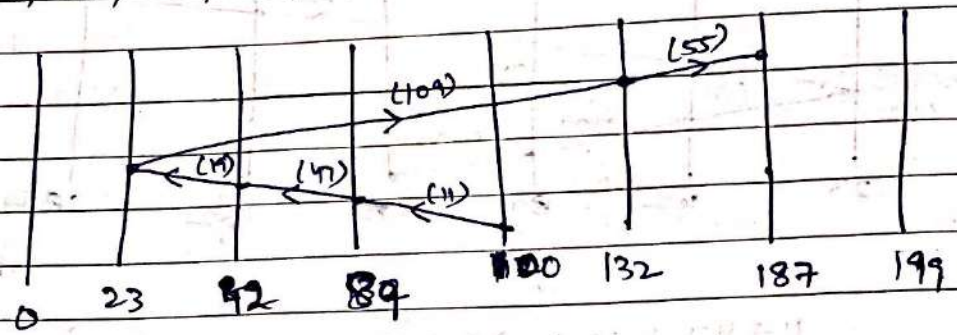



11 + 47 + 19 + 23 + 199 + 12 + 55 = (366) cylinders

LOOK :-

→ Like SCAN but stops moving inwards (or outwards) when no more requests in that direction exists.

eg: 23, 89, 132, 42, 187

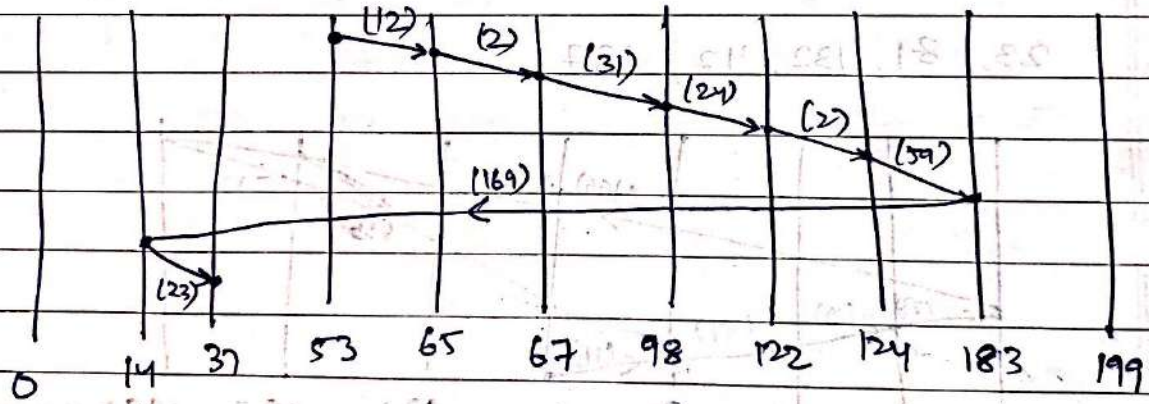



11 + 47 + 19 + 109 + 55 = (241) cylinders

## C-LOOK

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

head starts at 53




12 + 2 + 31 + 24 + 2 + 59 + 169 + 23 = (322) cylinders

→ In C-Look scheduling, the arm goes only as far as the final request in each direction

→ Then reverse direction immediately without going all the way to the end of the disk.

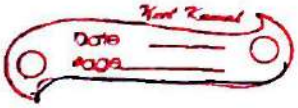
→ When head reaches the other end
- It immediately returns to the lowest cylinder request and without servicing any requests on the return trip.

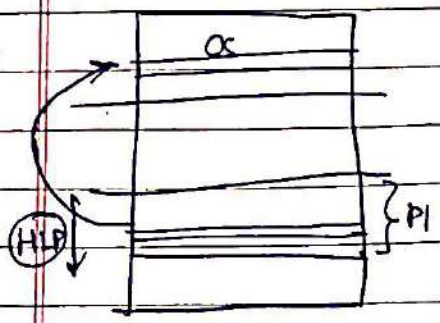
Gate16 Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling is used. The head is initially at cylinder 63, moving to larger on its service pass. Cylinders are numbered 0 to 199. To head movement (# cylinders) = _____

Sol. $(191-63) + (191-10) + (47-10)$

$= \boxed{346}$

# 6. Threads & System Calls






## Process Control System Calls

- End, abort
- load, execute
- Create process, terminate process
- Get process attributes, set process attributes
- Wait for time
- wait event, signal event
- allocate and free memory.

## File manipulation System calls:-

- Create file, Delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes.

## Device manipulation system calls:-

- request device, release device
- Read, write, reposition
- get device attributes, set device attributes
- logically attach or detach # devices

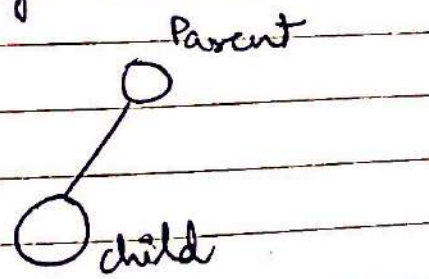
## Information maintenance System Calls

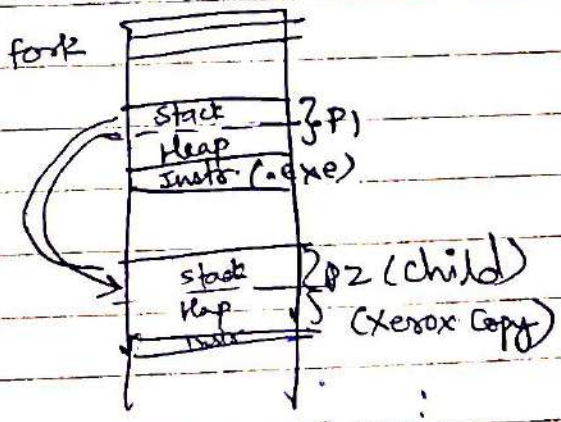
→ Get time or date, set time or date
→ Get system data, set system data
→ Get process, file or device data
→ Set process, file or device data

## Communications related system calls

→ Create, delete communication connection
→ Send, receive messages
→ Tranfer status information
→ Attach or detach remote devices

## Fork system Call (Fork / execve)

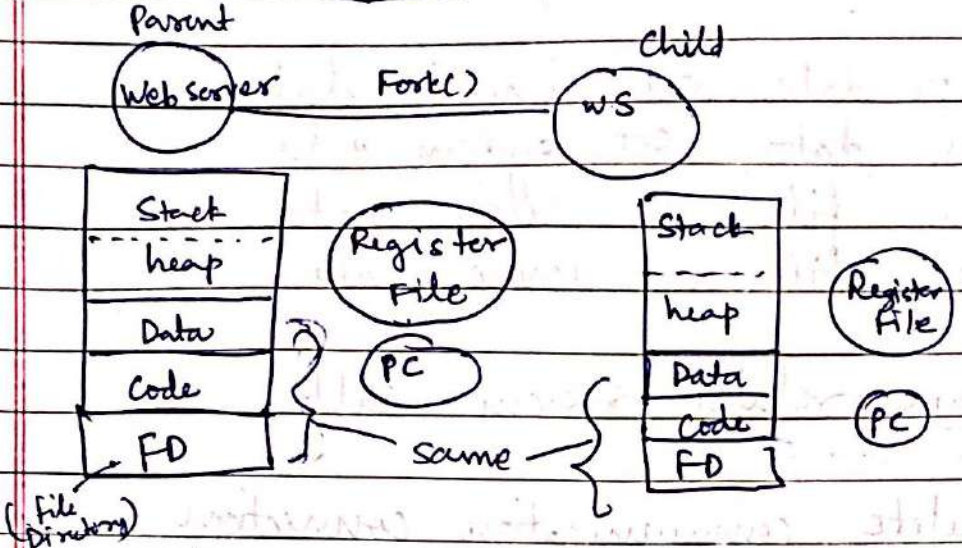


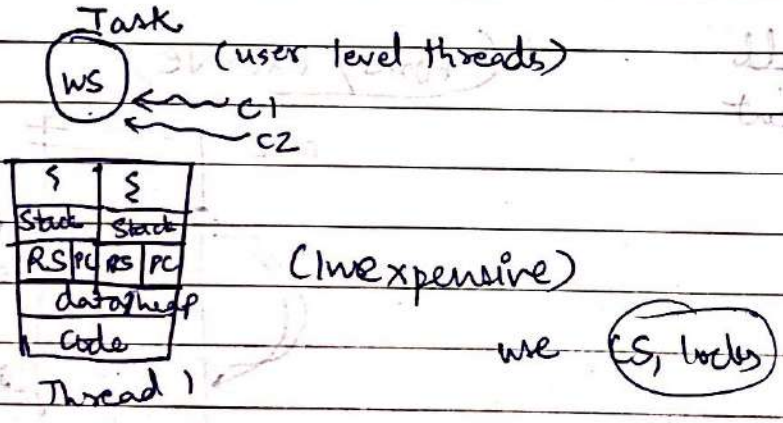

```
int i = fork();
if (i==0)
{
    child process
}
if (i!=0)
{
    parent process
}
}
```

0 → child process
pid of child → parent process


## Process vs. Threads




CPU wise expensive - Context switch required
Memory wise expensive - unecessirily wasted (copied)

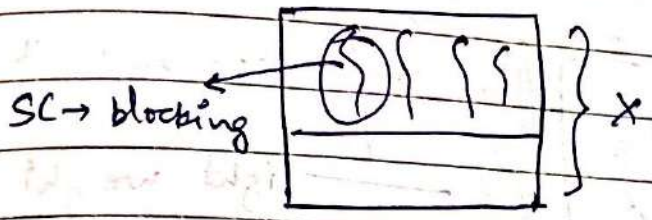



Thread :- New, Ready, Run, Block, Terminated

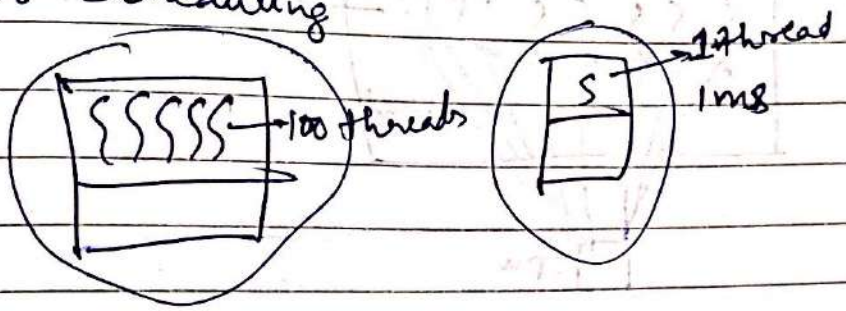
| Process (Fork) | Threads (user level) |
|---|---|
| → System Calls are involved | → No system call |
| → Context switching required | → Register set switch only |
| → Expensive | → Cheaper |
| → Different copies of code & data | → Same copies of code & data |

## Problems with user level threads :-

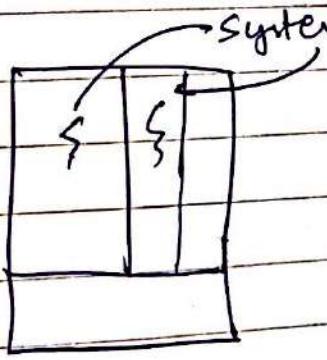
① Blocking system calls will block the whole task.




② Unfair scheduling




## Kernel level threads :-




Kernel is involved in thread creation and it knows about all threads

> Both problems of user level threads solved

Disadv :-

① Expensive as compared to user level threads

② 8 context switch required.

Expenses

Process > Kernel level threads > User level threads

## Hybrid Threads

Solaris 2
(v. of UNIX) :-