Operating System:-

An operating system (OS) is a system software which is an interface between computer Hardware & users. It is responsible for all basic tasks like :- process scheduling, process synchronisation, memory management, Disk management file management etc.

Types of Operating System:-

Batch Operating System:-

In batch OS, users do not interact with computer directly. Instead, each user writes the program on devices like punch cards, then it is submitted to the operator.
Jobs will be sorted by operator acco. to the requirements & executed completely in a group of similar kinds. Outputs are also written in devices like punch cards.
eg:- Bank statements, bulk salary credited system.

Time-sharing OS:-

In time-sharing OS, each process gets a specified time to execute on CPU as here are multiple processes waiting to execute since only 1 CPU is present.
Specified time allotted to a process is known as time quantums, generally denoted by tq. These type of OS are

Advantage:-
① Easy to handle repeated work.
② batch systems can be shared by multiple users.
③ Generally used for jobs which take very less time

Disadvantage:-
① A lot of time was wasted in reading / writing on punch cards.
② Other jobs has to wait for unknown time, if any job in execution is taking longer period.

Advantage:-
① Fair chance to get CPU for every task.
② CPU utilization is high as CPU will be idle for less amount of time.

Disadvantage:-
① security & Integrity is of the user programs
② Communication of da

Multi-processing OS:-

In Multi-processing OS, more than 1 CPU is present inside a single Computer.
Multiple processes can be executed at a time, Sharing of bus, clock, memory & I/O devices will be there in multi-processing OS.
by assigning them to diff-2 CPUs. Say we have dual-core processor then 2 processes can be executed at once. Similarly in Quad core processor, 4 processor can be executed at once & will be 4 times faster than uniprocessor.

Advantages

1. faster speed of computer & multiple processes are executing simultaneously.

Disadvantages

1. Data communication b/w processes is a difficult task.
2. Data security is also an important issue.

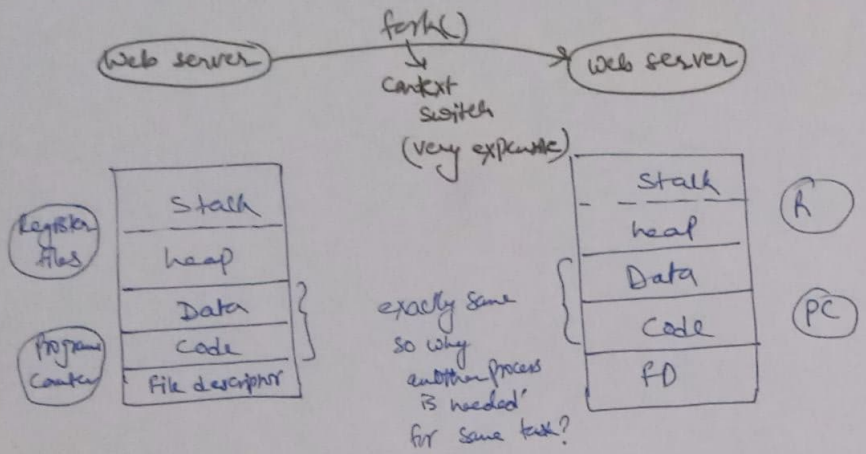
Time-Sharing OS:-

Multi-programmed OS:-

In real scenario, there are lot of processes waiting for execution i.e they are waiting for CPU to be allocated. While the main memory is not so large to accomodate all the waiting processes, so the process has to wait in job pool.

CPU selects one of the all processes kept in job pool using some job scheduling criteria & allocate it to CPU, then it starts executing. The process will execute until it needs to do I/O or interrupted by some external factor.

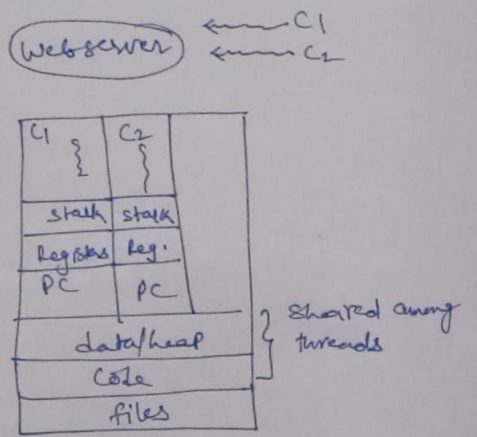
Process vs thread



Threads

User level thread

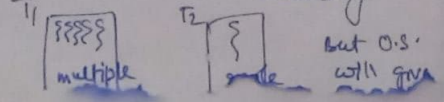
* User create the new thread. (O.S. is not involved).
* Switching b/w threads also O.S. is not involved.



Process (fork)	Threads (user level)
→ system calls are involved	→ NO system call
→ context switching is req.	→ NO context switching simply reg. set switch.
→ Diff copies of code & data	→ same copies of code & data.
→ C.S. & locks are req. for sync.	

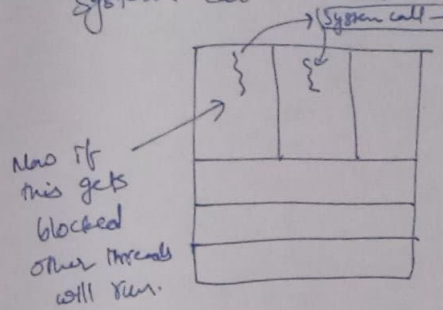
Disadvantages

* Even if 1 thread is blocked by O.S. then whole process (all threads) are blocked.
* Unfair Scheduling.



Kernel level threads :-

→ Now to create a new thread we have to make a system call to make kernel aware of diff. threads.

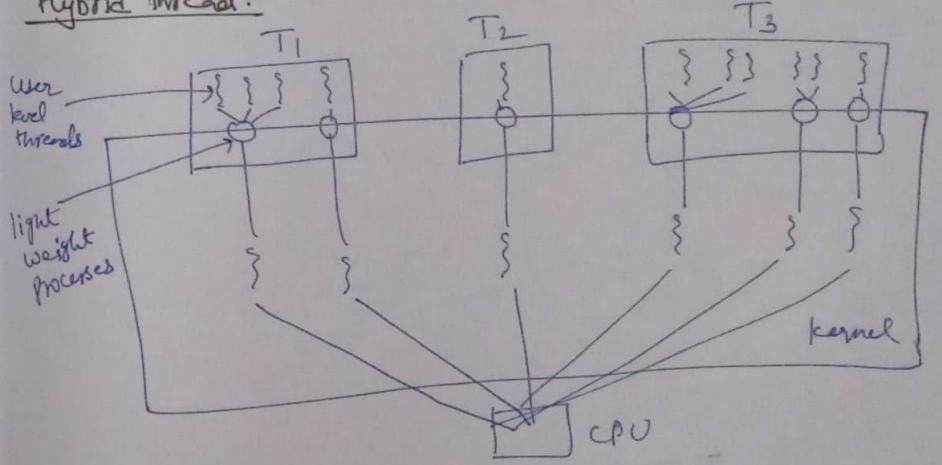


→ Appropriate scheduling will be done now.

Disadvantage

→ Expensive as compare to user level threads for creation.
→ Switching a thread req. system call (context switch).

Hybrid thread :-



CPU scheduling

Picking up one process from the Ready state & give it to the CPU for its execution.

Prameters for evaluation!-

① **CPU Utilisation!-** % of time CPU is busy doing useful work. CPU % = $\frac{\text{useful time}}{\text{total time}} \times 100$.

② **Avg. Response time!-**

Response time is the time a process has to wait in ready Q after being loaded.

Response time = process start time − Arrival time.

③ **Avg. waiting time!-**

It is the total time a process waits in Ready Queue.

waiting time = Turn around time − Burst time.

④ **Avg. turn Around time!-**

Total time of a process from its arrival till its completion.

Turn Around time = Completion time − Arrival time.

⑤ **Completion time!-**

Time at which a process completes or term

⑥ **Throughput!-**

Non-preemptive scheduling :-

If a process does it leaves CPU before its completion, it is non-preemptive scheduling.

Preemptive scheduling :-

If a process is forced to leave a CPU before its completion because of various reasons. it is preemptive sch.

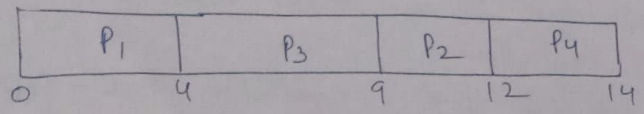
FCFS :- (Criteria is Arrival time)

FCFS is a Non-preemptive scheduling algo.

Eg:-

Process	Arrival time	Burst time	completion time	Turn around time	waiting time
P_1	0	4	4	4	0
P_2	2	3	12	10	7
P_3	1	5	9	8	3
P_4	4	2	14	10	8

Sol^n



	Response time	Start time
P_1	0	0
P_2	7	9

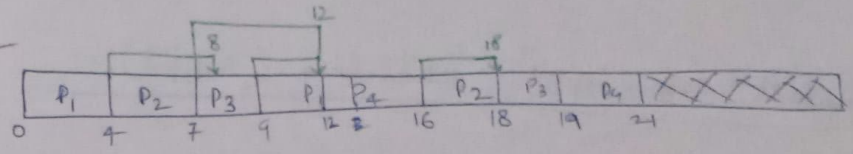
CPU % = $\frac{14}{14} \times 100 = 100\%$

Throughput = $\frac{14}{4}$

Q

Process	Burst Time	Arrival time	I/O Burst time
P_1	4, 3	0	4
P_2	3, 2	1	5
P_3	2, 1	4	3
P_4	4, 2	10	2

Sol^n



Process	Start time	Completion time	Turn around time	waiting time	Response time
P_1	0	12	12	1	0
P_2	4	18	17	7	3
P_3	7	19	15	4	3
P_4	12	21	11	3	2

Avg Turn around time = $\frac{12+17+15+11}{4} = \frac{55}{4}$

" waiting time = $\frac{1+7+9+3}{4} = \frac{20}{4}$

" Response time = $\frac{0+3+3+2}{4} = \frac{8}{4}$

CPU % = $\frac{21}{21} \times 100 = 100\%$

* FCFS scheduling algo observe convoy effect.

eg:-

Process	AT	BT
1	0	20
2	1	2
3	1	1

PNo.	AT	BT
1	1	20
2	0	2
3	0	1

Shortest job first scheduling (SJFS):-

Criteria is Burst time.

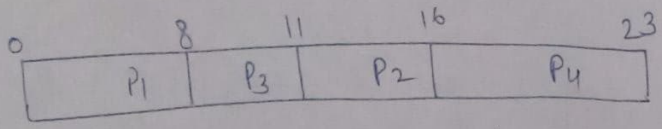
Mode: Non-preemptive & Preemptive both
↳ Shortest remaining time first sche.

* Note:- Tie of multiple same shortest Burst time processes is broken with FCFS criteria.

Q Mode: Non-preemptive

Process	Burst time	Arrival time
P_1	8	0
P_2	5	1
P_3	3	2
P_4	7	3

SJF



Process	Start time	Completion time	TAT	WT	RT
	0	8	8	0	0
	11	16	15	10	10

Avg TAT = $\frac{8+15+9+20}{4}$ =

Avg WT = $\frac{0+10+6+13}{4}$ =

Avg RT = $\frac{0+10+6+13}{4}$ =

CPU % = $\frac{23}{23} \times 100\%$ = 100%

Throughput = 4/23

* SJFS gives best throughput.
* SJFS suffers from starvation because if longer jobs may have to wait if shorter jobs keep coming.
* FCFS doesn't have starvation problem (assuming no process will run infinitely).
* It is very difficult to predict the CPU burst before actually running it.

Prediction Techniques

- Static
  1. process size
  2. process type
- Dynamic
  1. Simple averaging
  2. Exponential averaging

Process size:
if P_1 of 100KB took 10ms then we can assign

Process type
- OS process (5-6 units)
- user process
  - interactive

Simple average.

→ Given n-processes ($P_1 P_2 P_3 \dots P_n$)
→ t_i be the actual burst time.
& τ_i denotes the predicted burst time.

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

Exponential average method:-

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

where

t_n = Actual CPU burst for n^{th} proce
τ_n = Estimate of CPU burst for n^{th} time.
α = smoothing parameter (0–1)

$$\tau_2 = \alpha t_1 + (1-\alpha)\tau_1$$

$$\tau_3 = \alpha t_2 + (1-\alpha)\tau_2$$
$$= \alpha t_2 + (1-\alpha)[\alpha t_1 + (1-\alpha)\tau_1]$$
$$= \alpha t_2 + \alpha(1-\alpha)t_1 + (1-\alpha)^2\tau_1$$

$$\tau_4 = \alpha t_3 + (1-\alpha)\tau_3$$
$$= \alpha t_3 + (1-\alpha)(\alpha t_2 + \alpha(1-\alpha)t_1 + (1-\alpha)^2\tau_1$$
$$= \alpha t_3 + (1-\alpha)\alpha t_2 + \alpha(1-\alpha)^2 t_1 + (1-\alpha)^3\tau_1$$

qq:- $\alpha = 0.6$ $\tau_1 = 10$
actual burst time of (t_1, t_2, t_3) are (5, 9, 5)
then $\tau_4 = ?$

Sol:

$$\tau_2 = \alpha t_1 + (1-\alpha)\tau_1$$
$$= 0.6 \times 5 + 0.4 \times 10$$
$$= 7.0$$

$$\tau_3 = \alpha t_2 + (1-\alpha)\tau_2$$
$$= 0.6 \times 9 + 0.4 \times 7$$
$$= 6.4$$

$$\tau_4 = \alpha t_3 + (1-\alpha)\tau_3$$
$$= 0.6 \times 5 + 0.4 \times 6.4$$

$\tau_4 = 5.56$ Ans

Shortest Remaining time first:-

Criteria: Burst time.
Mode: preemptive.

ST	EX / CT	Process	Burst time	Arrival time	ATAT	WT	RT
0	13	P_1	7	0	13	6	0
1	7	P_2	4	1	6	2	0
2	4	P_3	2	2	2	0	0
13	19	P_4	6	3	16	10	10

0 1 2 4 7 13 19

P_1 P_2 P_3 P_2 P_1 P_4

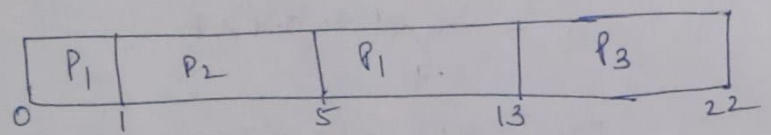
CPU % = 100%

Q what is Avg waiting time using SRTF? (Gate-2011)

Process	Arrival time	Burst time
P_1	0	9
P_2	1	4
P_3	2	9

a) 5.0 ms
b) 4.33 ms
c) 6.33 ms
d) 7.33 ms

Soln

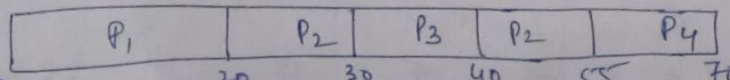


Avg waiting time = $\frac{4+0+11}{3} = \frac{15}{3} = 5.0$ ms

Q what is waiting time of P_2 using SRTF? (Gate-2001)

Process	AT	BT
P_1	0	20
P_2	15	25
P_3	30	10
P_4	45	15

Soln



Round Robin (RR):-

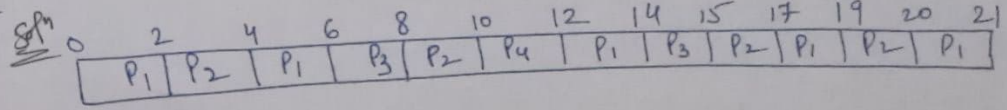
→ Each process is given a time Quanta tq.
→ If it is completed before it then we schedule next process immediately.
→ If it want more time then we will put it at the end of Ready Queue after tq time Quanta is over.

Eg:-

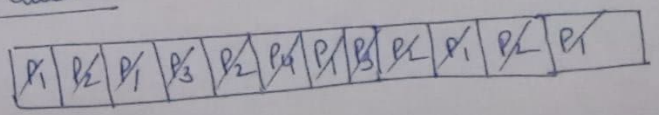
Process	Burst time	Arrival time
P_1	~~9 7 5 3 1~~	0
P_2	~~7 5 3 1~~	1
P_3	~~3 1 0~~	3
P_4	~~2 0~~	5

tq = 2 ms

Soln



Ready Queue



Process	Start time	Completion time	TAT	WT	RT
P_1	0	21	21	12	0
P_2	2	20	19	12	1
P_3	6	15	12	9	3

CPU % = 100 %
Throughput = 4/21

Eg

Process	Arrival time	Burst time
P_1	0	3
P_2	1	4
P_3	2	2
P_4	3	1
P_5	4	7
P_6	5	4

Sol^n Consider ① $T_q = 2ms$ $T_q = 4ms$.

Eg:-

Process	Arrival time	Burst time
P_1	6	7
P_2	5	6
P_3	4	5
P_4	2	2
P_5	3	9
P_6	1	3

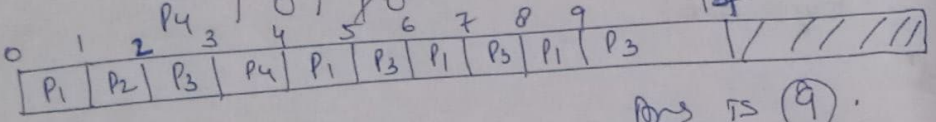
$tq = 3ms$:

Sol^n

Eg:- Consider 4 jobs P_1, P_2, P_3, P_4 arriving in ready Queue in the same order at time = 0. If Burst time req of these jobs are 4, 1, 8, 1 resp. what is completion time of P_1, assuming RR with $T_q = 1$.

Sol^n

Process	AT	BT	$T_q=1$
P_1	0	4 3 2 1 0	
P_2	0	1 0	
P_3	0	8 7 6 5	
P_4	0	1 0	



Ans is (9).

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

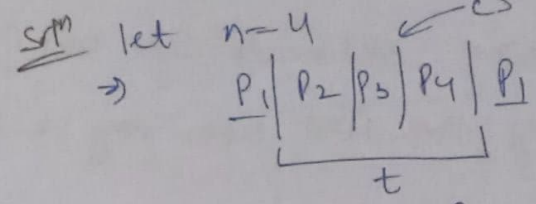
Eg:- Consider 'n' processes sharing the CPU in a RR-Fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

(Gate - 1998)

a) $q \leq \frac{t-ns}{n-1}$ b) $q \geq \frac{t-ns}{n-1}$

c) $q \leq \frac{t-ns}{n+1}$ d) $q \geq \frac{t-ns}{n+1}$

Sol^n let n = 4



⇒ Between P_1 & P_1 we have 4 CS & 3 tq.

$n \times s + (n-1)\, tq \leq t$

$\Rightarrow$

$\boxed{tq \leq \frac{t-ns}{n-1}}$

Important points!-

① RR is preemptive algo & doesn't suffers from starvation.

② RR gives a bound on response time for a process

max response time = $n \times tq$.

③ RR makes it useful for interactive system.

④ If Tq increases then Context switch decreases & Response time increases.

⑤ If Tq decreases then CS increases & RT decreases.

⑥ If tq = very large like ∞.

RR becomes FCFS.

⑦ If tq = very small.

CPU % becomes very less. because of time wastage in context switches.

⑧ If tq = ts (context switch time)

CPU % = $\frac{tq}{tq+ts} = \frac{tq}{2tq} = \frac{1}{2} \Rightarrow 50\%$.

(Gate-2010)

Q Which of the following statements are true?

I. Shortest remaining time first sche. may cause starvation.

II. Preemptive scheduling may cause starvation

III. RR is better than FCFS in terms of Response time.

a) I only b) I & II only c) II & III only d) All

(Gate-2012)

Q Conside the 3 processes P1, P2 & P3 shown in table.

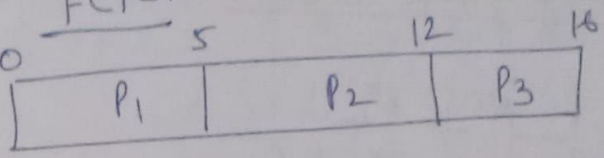
Process	Arrival time	Burst time
P1	0	5 3 1 0
P2	1	7 5 3 0
P3	3	4 2 0

The completion order of the 3 processes under the policies FCFS & RR2 (RR with tq = 2) are

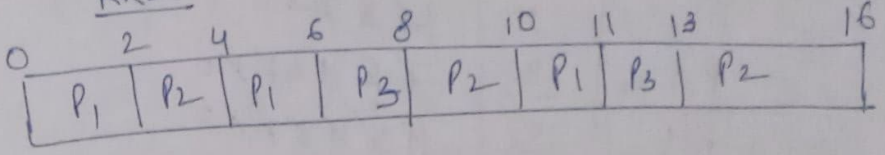
a) FCFS: P1 P2 P3 RR2: P1 P2 P3
b) FCFS: P1 P3 P2 RR2: P1 P3 P2
c) FCFS: P1 P2 P3 RR2: P1 P3 P2
d) FCFS: P1 P3 P2 RR2: P1 P2 P3

Sol'n

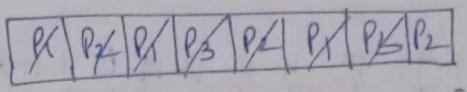
① FCFS:-



RR2:



RQ



⇒ order of completion is P1, P3, P2

⇒ FCFS: P1, P2, P3
RR2: P1, P3, P2

Longest first job algo

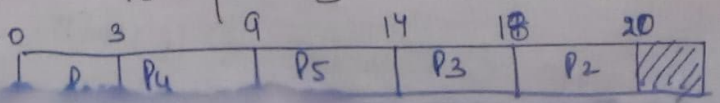
Criteria: Burst time.
Mode: non-preemptive.

Process having longest BT scheduled first.

Eg:-

Process	Burst time	Arrival time	CT	WT	RT	ST	TAT
P1	3	0	3	0	0	0	3
P2	2	1	20	17	17	18	19
P3	4	2	18	12	12	14	16
P4	6	3	9	0	0	3	6
P5	5	4	14	5	5	9	10

Sol'n



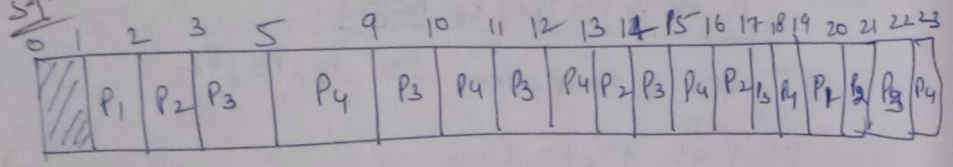
CPU % = 100 %
Throughput = 5/

Longest remaining time first!-

Eg:-

P.NO	AT	BT
P_1	1	2 1 0
P_2	2	4 3 2 1 0
P3	3	7 6 5 4 3 2 1 0
P4	5	9 8 7 6 5 4 3 2 1 0

Sol^n



P	St	CT	TAT	WT	RT
P_1	1	20	19	17	0
P_2	2	21	19	15	0
P_3	3	22	19	12	0
P_4	5	23	18	9	0

CPU% = $\frac{22}{23} \times 100$

Throughput = 4/23

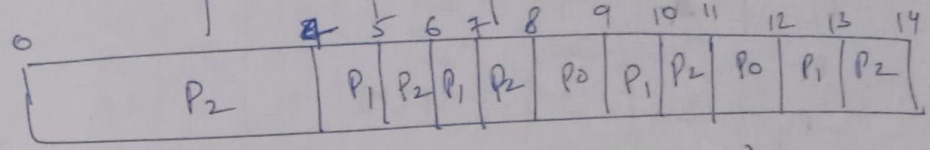
Que: (Gate-2006)

Consider 3 processes (0,1,2) with compute time burst 2, 4, 8 units. All processes arrive at time zero. Consider the LRTF algo. In LRTF ties are broken with lowest process id first. The avg. turn around time is

a) 13 units b) 14 units c) 15 units d) 16 units.

Process	Burst time	AT	CT	TAT
P_0	2	0	12	12
P_1	4 3 2	0	13	13
P_2	8 4 3 2	0	14	14

Avg TAT = $\frac{12+13+14}{3}$ = $\frac{39}{3}$ = 13 units.



Highest Response Ratio Next (HRRN)

Mode: non-preemptive

Criteria:- Response Ratio.

$$= \frac{W+S}{S}$$

Where

W = Waiting time at time t.

S = Burst time of process P.

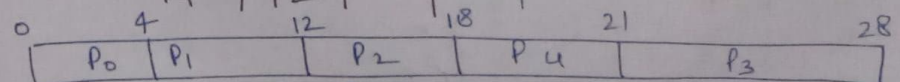
→ HRRN not only favours shorter jobs but also limits the waiting time of longer jobs.

Ex

Process	AT	BT	ST	CT	TAT	WT	RT
P_0	0	4	0	4	4	0	0
P_1	3	8	4	12	9	1	1
P_2	6	6	12	18	12	6	6
P_3	9	7	21	28	19	12	12
P_4	12	3	18	21	9	6	6

CPU% = 100%

Throughput = 5/28.



$RR_2 = \frac{6+6}{6}$ $RR_3 = \frac{9+13}{10}$ $RR_4 = \frac{3+0}{3}$ $RR_3 = \frac{7+9}{7} = 2.29$

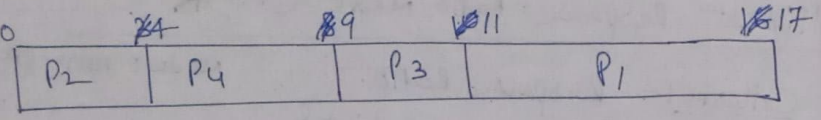
Priority Scheduling:-

Criteria: Priority mode: non-preemptive

Ex:①

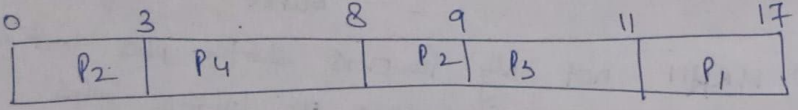
Process	BT	AT	Priority
P_1	6	0	1 (L)
P_2	1	0	3
P_3	2	2	2
P_4	5	3	4 (H)

Soln



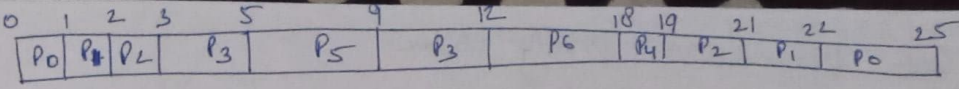
Ex:① mode: Preemptive

Soln



Ex:-

Process	Priority	AT	BT
P_0	2 (L)	0	4 3 0
P_1	4	1	2 1 0
P_2	6	2	3 2 0
P_3	10	3	5 3 0
P_4	8	4	1 0
P_5	12 (H)	5	4 0
P_6	9	6	6 0



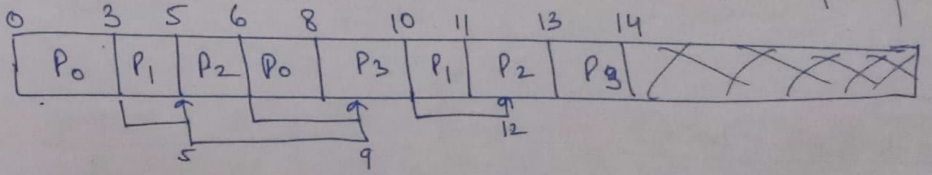
Process	ST	CT	TAT	WT	RT
P_0	0	25	25	21	0
P_1	1	22	21	19	0
P_2	2	21	19	16	0
P_3	3	12	9	4	0
P_4	18	19	15	14	14
P_5	5	9	4	0	0
P_6	12	18	12	6	6

· CPU % = 100% Throughput = 7/25 ·

Que using SRTF · mode – non-preemptive.

Process	AT	BT	P/O	ST	CT	TAT	WT	RT
P_0	0	3,2	2	0	8	8	3	0
P_1	0	2,1	4	3	11	11	8	3
P_2	2	1,2	3	5	13	11	8	3
P_3	5	2,1	2	8	14	9	6	1

Soln



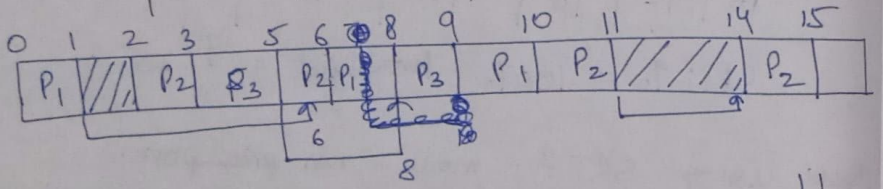
Throughput = 4/14

CPU % = 100%

Que Pre-emptive priority sche.

Process	AT	Priority	BT	I/O
P_1	0	2	~~1~~, ~~3~~ 1	5
P_2	2	3 (L)	~~1~~ ~~2~~ ~~3~~, 1	3
P_3	3	1 (H)	2, 1	3

Soln



P_{id}	ST	CT	TAT	WT	RT
P_1	0	10	10	~~9~~ 1	0
P_2	2	15	~~13~~	~~6~~ 6	0
P_3	3	9	6	~~0~~ 0	0

CPU % = $\frac{11}{15} \times 100$

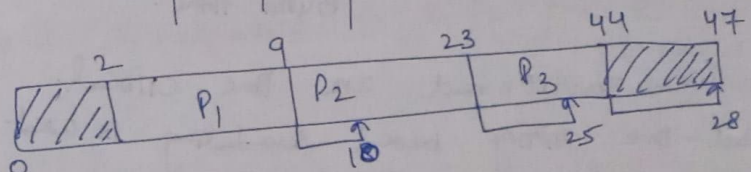
Throughput = 3/15

Que (Gate-2006)

a) 0% b) ~~10.6%~~ c) 30.07% d) 89.49%.

Soln

Process	AT	BT	I/O ~~ST~~ ~~CT~~
P_1	0	7	2, 1
P_2	0	14	4, 2
P_3	0	21	6, 3

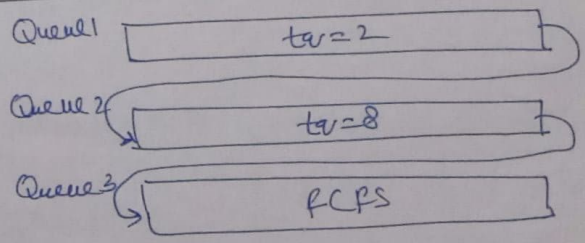


CPU % = $\frac{42}{44} \times 100$ ⇒ CPU idle % = $1 - \frac{42}{44} \times 100$

$\frac{5}{47} \times 100 = 10.6\%$

$= \frac{2}{44} \times 100 = 4.55\%$

Multi-level feedback Queue:-

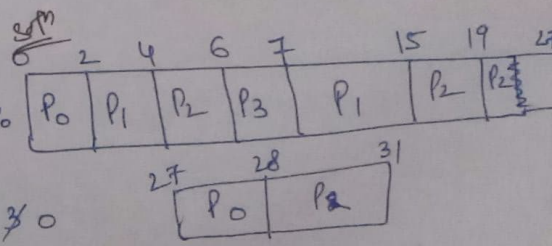


* first it serves 1st Queue after it become empty then go to 2nd Queue & so on.

Eg:-

Process	AT	BT
P_0	0	~~19~~ ~~6~~ 0
P_1	1	~~8~~ 4 0
P_2	2	~~13~~ ~~1~~ ~~3~~ 0
P_3	3	~~X~~ 0

Soln



General Reviews!-

① Response ratio of a process

$$= \frac{\text{process time (Burst time)}}{\text{TAT of that process}}$$

② Penalty ratio = 1/Response ratio.

or Penalty ratio = $\frac{\text{TAT of that process}}{\text{Burst time}}$

③ Real-time environment are time critical.
In real-time priority based scheduling is used.

④ RR is used in interactive jobs.

⑤ Normalised TAT = $\frac{\text{Process TAT}}{\text{Burst time}}$

Deadlocks

Each process has the characteristics:

- Request a resource
- use
- Release the resource

Resources ⟶ I/O devices, memory, tables, database

eg:- ① 2 processes P1 & P2 are concurrently executing (it means interleaving not simultaneously)

P1

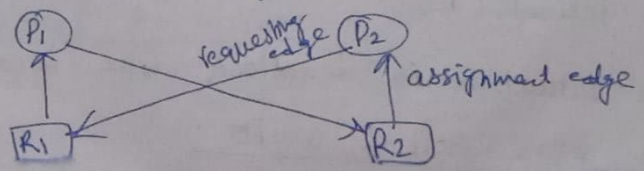
```
request(R1);
⟶ use(R1);
request(R2);
|
release(R1);
|
release(R2);
```

P2

```
request(R2);
!
⟶
request(R1);
!
use(R1);
use(R2);
!
release(R2);
release(R1);
```

* This Situation will not always Result in deadlock because if P1 executes completly in one go then NO Deadlock

Deadlock!- A situation in which no further movement is possible even in infinite time.



4 essential conditions for deadlocks :-

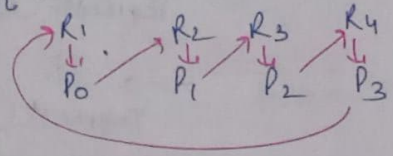
① **Mutual exclusion :-** Few Resources are mutually exclusive. That means they are not sharable.

② **Hold & wait :-**
Each process in deadlock cycle holds a resource & waits for a resource.

③ **No preemption :-**
Resources cannot be snatched from a process.

④ **Circular wait :-**
There exists a cycle in which processes wait.

eg :-



eg :-

P_1	P_2
Request (R_1);	request(R_1);
Request (R_2);	Request (R_2);
!	!
release (R_1);	release (R_1);
release (R_2);	release (R_2);

Above situation is deadlock free because it is free from **hold & wait**.

eg :-

P_1	P_2
Request (R_1);	request (R_2);
⋮	⋮
release (R_1);	release (R_2);
⋮	!
request (R_2);	request (R_1);
⋮	⋮
release (R_2);	release (R_1);

* Deadlock free because **hold & wait** is violated.

4 ways in which Deadlock can be removed

① **Deadlock ignorance :-** One of the 4 essential conditions is violated.

② **Deadlock prevention :-** One of the 4 essential conditions is violated.

③ **Deadlock Avoidance :-** Considering max needs of each process, sys is kept is safe state.

④ **Deadlock Detection & recovery :-**
Run an algo to detect a deadlock. When detected run a recovery module.

Deadlock prevention :-

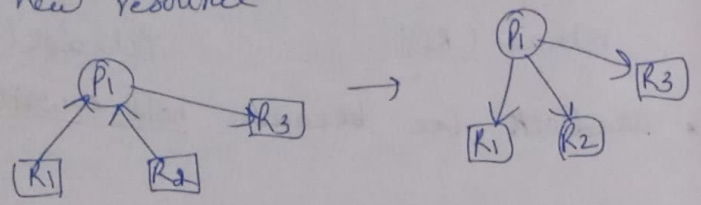
① **No Mutual Exclusion :-** If possible make resources Sharable. But many resources are inherently non-Sharable.

② No Hold & wait!-

i) All requirements of process is given during starting.

ii) Release all resources before you request for a new resource.

eg!-



③ Preemption!-

i) Higher priority process preempts the resources held by low priority process.

ii) Lower priority process releases resources when it can't acquire resources held by high priority process.

There are following problems with ⓑ & ⓒ method

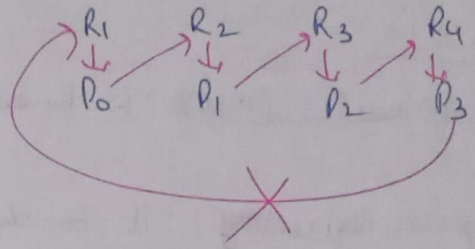
① Starvation.

② low resource utilization
↳ since most of the resources are idle most of the time.

③ low process throughput.

④ No circular wait!- Simply order the resources. Processes can request resources only in ascend

eg!-

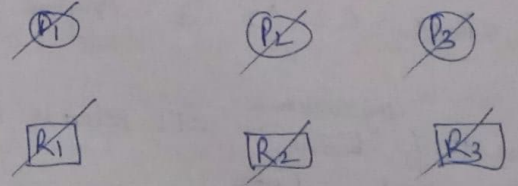


assuming that processes can request resources only in ascending order.

Situation!- Analyze it & tell whether deadlock is there or not.

P1	P2	P3
request(R1);	request(R2);	request(R1);
request(R2);	request(R3);	request(R3);
⋮	⋮	⋮
release(R2);	request(R1);	release(R1);
request(R3);	release(R1);	request(R2);
⋮	⋮	⋮
release(R3);	Release(R2);	Release(R2);
release(R1);	Release(R3);	Release(R3);

Sol: Deadlock B/w <P1, P2>



At arrows P1 is waiting for R2 & P2 is waiting for R1 whereas R1 is held by P1 & R2 is

Deadlock avoidance:-

→ Max needs of each process is known when process starts.

→ Before granting a request, it is checked if the resulting system will be safe.
if yes, request is granted.
otherwise, " " denied & process has to wait.

eg:-

Customers	Max Need	Allocated	Future needs
C_1	12	6	6
C_2	7	3	4
C_3	6	2	4
C_4	9	4	5

Total money in Bank = 19 Lakhs

sol^n Available " " " = Total − Allocated
= 19 − 15 = 4 lakhs

With available amount we can fulfil the request of either C_2 or C_3. Anyone can be choosen.

After granting request of any customer, it will release its allocated resources also.

C_2 Available 4
released 3
total available after C_2 completes
= 4 + 3 = 7.

By using 7 lakhs any customer's need can be fullfilled in the system.

C_4 available 7
released 6
total available = 7 + 6 = 13

C_3 available 13
released 2
total available = 13 + 2 = 15

C_4 available 15
released 4
total available = 15 + 4 = 19

After completing all customers need bank must have total available balance as total money in bank. (useful for checking the correctness)

Order of execution = $\langle C_2, C_1, C_3, C_4 \rangle$

Now let say there is a urgent request of Rs 1 lakh by C_1, should bank grant it?
If 1 lakh is provided to C_1 then its future needs will be 5 lakhs. & allocated will 7.
total available will bank is 19 − 16 = 3 lakhs.

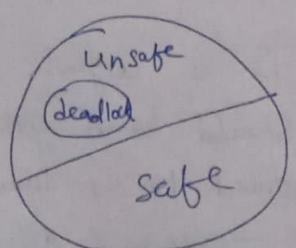
So the request will be denied & in this way we have avoided the deadlock.

Safe sequence:-

Hypothetical order of execution such that process P_i's need are satisfied & on completion, it will be assumed to release its allocated resources which can be used by another processes.

Safe State:- If in a system, there is atleast one safe sequence then the state is called safe state & there will be no deadlock.

Unsafe state does not necessarily imply deadlock.
As max need does not reflect always future requests.



Note:- If you increase the no's of resources a lot,

Ex:-

(a)

Process	Max need	Allocated	Future need
P_1	7	2	5
P_2	6	4	2
P_3	3	0	3
P_4	4	0	4
		6	

Total resources = 9
Total available = 9-6 = 3.

Soln With 3 resources either P_2 or P_3 needs can be fullfiled. pick any.

P_2: After its completion
available = 3+4 = 7

P_1: available = 7+2 = 9

P_3: " " = 9+0 = 9

P_4: " " = 9+0 = 9

Safe sequence = $\langle P_2, P_1, P_3, P_4 \rangle$ ⇒ System is in Safe State.

(b) Request of 1 resource to P_1 & 1 resource to P_3. Now, is the system in safe state?

Soln If we give 1-1 resource of P_1 & P_3 then system will look below.

	System max need	Allocated	future need
P_1	7	3	4
P_2	6	4	2
P_3	3	1	2

available resources = 9-8 = 1

⇒ with 1 available resource we cannot full fil the

Ex

Process	Max need R_1 R_2 R_3	Allocated R_1 R_2 R_3	future need R_1 R_2 R_3
P_1	7 5 2	4 3 1	3 2 1
P_2	6 4 3	2 2 2	4 2 1
P_3	2 8 1	2 2 0	0 6 1
P_4	5 6 4	2 1 3	3 5 1
		10 8 6	

Sol^n ⑨ Total resources = (13, 11, 8)

Available resources = (3, 3, 2)

With available resources only P_1's request can be fulfilled.

P_1 → Completes available = 3 3 2
4 3 1 → returned
7 6 3

P_2 → Completes available = 2 2 2 → Returned
9 8 5

P_3 → Completes available = 2 2 0 → Returned
11, 10, 5

P_4 → Completes available = 2 1 3
13, 11, 8

Safe sequence = $\langle P_1, P_2, P_3, P_4 \rangle$

⇒ system is in safe state.

⑤ will OS grant a request of (0,1,1) from P_2?

Sol^n System picture if P_2's request is granted

Process	Max need R_1 R_2 R_3	Allocated R_1 R_2 R_3	future need R_1 R_2 R_3
P_1	7 5 2	4 3 1	3 2 1
P_2	6 4 3	2 3 3	4 1 0
P_3	2 8 1	2 2 0	0 6 1
P_4	5 6 4	2 1 3	3 5 1
		10, 9, 7	

Available resources = 13, 11, 8 − 10, 9, 7 = 3, 2, 1

With above available resources P_1's request can be filled.

P_1 → Completes Available 3 2 1
4 3 1 → Returned
7 5 2

P_2 → Can be completed with new available resources.
→ Completes. available → 7 5 2
2 3 3 → returned
9 8 5

P_3 → Completes. available 9 8 5
2 2 0 → Returned
11, 10, 5

P_4 → Completes available = 11, 10, 5
2 1 3

⇒ P_2's request is granted. Since

© Will OS granted a request of (1,0,0) from P_4?

Soln System picture if P_4's request is granted.

Process	MAX Needs R_1 R_2 R_3	Allocated R_1 R_2 R_3	Future need R_1 R_2 R_3
P_1	7 5 2	4 3 1	3 2 1
P_2	6 4 3	2 2 2	4 2 1
P_3	2 8 1	2 2 0	0 6 1
P_4	5 6 4	3 1 3	2 5 1
		11, 8, 6	

Available resources = 13, 11, 8
11, 8, 6
2, 3, 2

Now, with available resources none of the processes requests can be fullfilled completely.
So, OS will reject the P_4's request to keep system in safe state.
From the above process OS has avoided deadlock.

Min Resources Requirements

* If we have unlimited resources or we increase the no's of resources drastically then the system can always be in safe state.
* Increasing resources can be costly.

deadlock free

eg:- Find min resources required to keep sys.

Process	max needs	Allocated	Future Need
P_1	5	4	1
P_2	4	3	1
P_3	3	2	1

Soln * If all the processes acquire Max-1 resources then also system will be in deadlock.
* By including 1 more resource, system will be deadlock free. Because whichever process takes this 1 resource will return atleast this resource & can be utilized by other processes & deadlock will be breaked.

→ min resources = $\sum$ max needs − no's of processes + 1
= $n \times m - n + 1$ where
or
n = no's of proces
m = max ne

min resources $\geq \sum_{max need} - no's\ of\ processes + 1$

$\geq n \times m - n + 1$

min resources $= 12 - 3 + 1$

$= 10$ Ans

Ex Find min no's of resources so that sys. always remain free from deadlock.

Process	Max needs R_1	R_2	R_3
P_1	7	5	2
P_2	6	4	3
P_3	2	8	1
P_4	5	6	4

Soln $\sum_{Max\ needs} = 20, 23, 10$

min resources $= (20, 23, 10) - (4, 4, 4) + (1, 1, 1)$

$= (16, 19, 6) + (1, 1, 1)$

$= (17, 20, 7)$ Ans.

Deadlock Detection & Recovery :-

→ OS will periodically check for if any deadlock is present in the system.

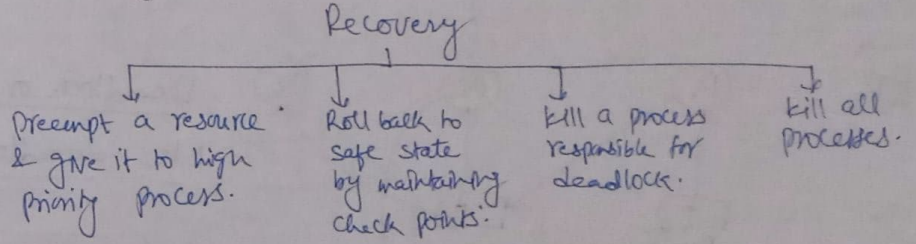
→ If deadlock is detected then, OS will run recovery module to ~~det~~ remove deadlock.

Detection :-

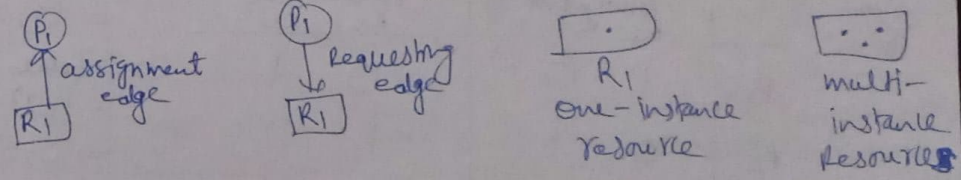
→ In single instance resource type, if a cycle is present then deadlock is detected.

→ In multi-instance resource type, we will run safety algo. (Modification of Banker's Algo).

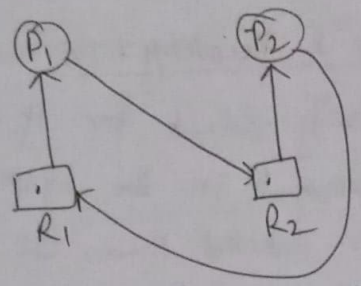
Recovery :-



Identification of deadlock using Resource allocation Graph



Eg



Deadlock or NOT?

Yes

Soln

	Allocated R1	Allocated R2	Requested R1	Requested R2
P1	1	0	0	1
P2	0	1	1	0
	1	1		

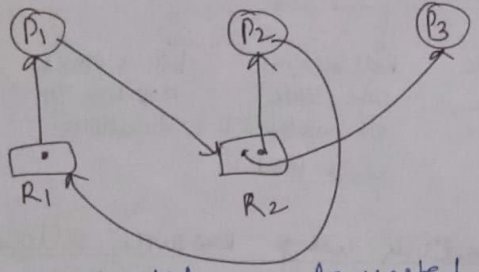
Total resources = (1,1)

Available resources (total) = (1,1) − (1,1) (allocated)

= (0,0)

By the available resources no process needs can be satisfied. Hence Deadlock is present.

Eg:- *



Deadlock or NOT?

No deadlock.

Soln

	Allocated R1	Allocated R2	Requested R1	Requested R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0
	1	2		

Total resources = (1,2)

available resources = (1,2) − (1,2)

= (0,0)

But as P3 don't need any resources it may release

P3 releases (0,1)

total available now (0,1)

P1 needs can be satisfied & will release (1,0)

total available = (0,1) + (1,0)

= (1,1)

P2 needs can be satisfied & will release (0,1)

total available = (1,1) + (0,1)

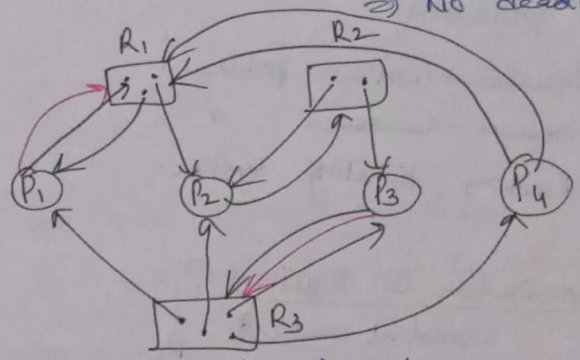
= (1,2)

order of execution

= (P3, P1, P2) ⇒ safe state.

⇒ No deadlock.

Eg:-



Deadlock or NOT?

No, deadlock

Soln

	Allocated R1	Allocated R2	Allocated R3	Requesting R1	Requesting R2	Requesting R3
P1	1	0	1	2	0	0
P2	1	1	1	0	0	0
P3	0	1	0	0	0	2
P4	0	0	1	2	0	0
	2	2	3			

Total resources = (3,2,4)

Available resources (3,2,4) − (2,2,3)

= (1,0,1)

P1 needs → can be satisfied

total ava = 101 + 101

= 2,0,2

P3 → needs can be satisfied

total available = (2,0,2) + (0,1,0)

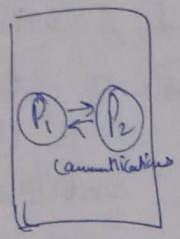
= (2,1,2)

P2 → needs can be satisfied.

total available = (2,1,2) + (1,1,1)

P4 needs can be sa...

Process Sychronisations



may share variables & various resources.

Processes which running in interleaving fashion because of their preemptive nature are called concurrently running processes.

Eg: Reader-writer problem.
Producer-consumer "
Railway ticketing system.

Why Sychronisation is required?

Shared int counter=3

P1
|
Counter++
|
}

P2
|
Counter--
|
}

→ counter value is stored in m/m.
→ ALU ~~does~~ interacts with Registers, not m/m.
→ So, counter++ or counter-- is not an atomic opr.

Shared int counter = 3

P1
|
1: MOV counter, R0
2: INC R0
3: MOV R0, counter
|
}

P2
|
4: MOV counter, R1
5: DEC R1
6: MOV R1, counter
|
}

Situation-1
let say order of execution is
P1: 1, 2, 3 & then P2: 4, 5, 6

R0	R1	counter
~~3~~ 4	~~4~~ 3	~~3~~ 4 ~~4~~ 3

Ans = (3) → correct.

Situation-2
let say order of execution is
P1: 1, 2, then P2: 1, 2, 3 then P1: 3

R0	R1	counter
~~3~~ 4	~~3~~ 2	~~3~~ ~~4~~ 4

Ans = (4) → wrong.

Situation-3
let say order of execution is
P2: 1, 2 then P1: 1, 2, 3 then P2: 3

R0	R1	counter
~~3~~ 4	~~3~~ 2	~~3~~ ~~4~~ 2

Ans = (2) → wrong.

As per the above situation, if order of execution

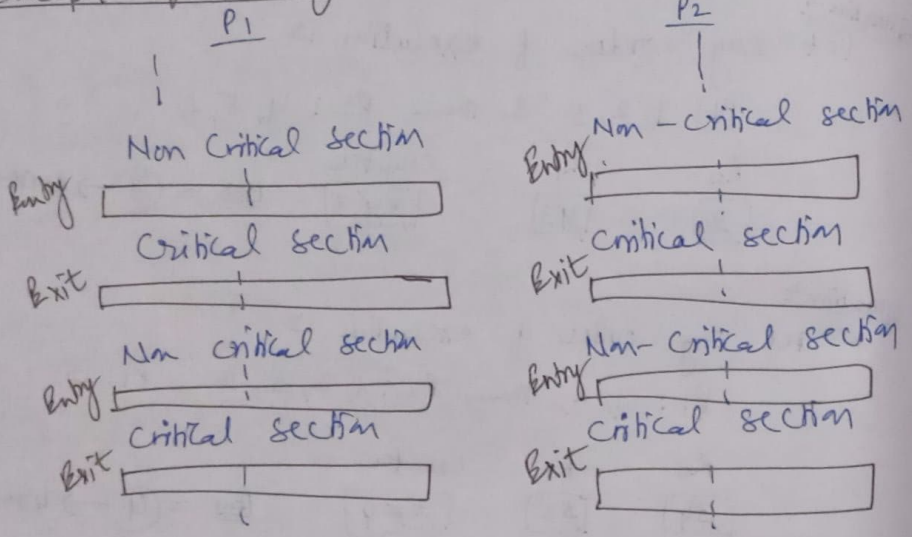
Race Condition:-

When final result depends on interleaving of execution of process.

Critical Section:-

A part of code, which consist of one or more shared variable & improper ordering of execution may result in race condition.

Concept for making process Synchronise:-



Requirements of Synchronisation:-

Primary

1. **Mutual Exclusion:-** Only one process should access the shared variable at a time inside critical section.
2. **Progress Requirement:-** No deadlock & No process should permanently [illegible] critical section for others.

3. **Bounded waiting:-** No Starvation.
   Bound on waiting time.
4. **Portability or Architecture Neutral:-**
   No specific architecture or OS is concerned.
   It is generic.

Busy Bounded waiting:- When a process continuously checking for accessing C.S. then CPU cycles gets wasted & process is in Busy waiting.

Solutions for synchronisation of processes.

- Software based
- O.S. provided structure
  - Semaphore
  - Monitor
- H/w provided instr.
  - Test & set
  - Swap

Software based solution:-

Algo 1:- shared int lock=0

P1

Entry section:
1. while(lock!=0)
2. lock=1

3. C.S.

Exit section:
4. lock=0

P2

5. while(lock!=0)
6. lock=1

7. C.S.

8. lock=0

Situation-1
P1: 1, 2, P2: 5, P1: 3, 4, P2: 5, 6, 7, 8.

problem

Situation -2:-

P1: 1, P2: 5, 6 ↓ inside C.S. P1: 2, 3 ↓ inside C.S.

⇒ NO M.E. is ensured.

Failed...!

Later | ignore here

* If we make the instr. 1 & 2, instr 5 & 6 as atomic then ~~it will become~~ we need some support from OS & hardware & it will become TSL (test set lock).

Now. Since entry is atomic so no 2 processes can enter inside C.S. at the same time.

So. ME is ensured.

& after execution of C.S. while executing exit section entry to no process is ensured

So progress is also ensured.

bounded waiting:- can't say.

because incase of priority processes one process has to wait longer so starvation is possible.

spin lock problem:-

While P1 is in C.S. & a higher priority process P2 came then P1 will be terminated. & P2 will be waiting ~~in critical section while remaining in~~ (P1). So a kind

Algo-2

shared int turn = 0;

P1

```
while(1){
1:  while(turn!=0);
2:      C.S.
3:  turn = 1;
    :
}
```

P2

```
while(1){
4:  while(turn!=1);
5:      C.S.
6:  turn = 0;
    :
}
```

Situation 1:-

P1: 1 P2: 4 ↓ wait P1: 2 P2: 4 ↓ still waiting P1: 3 ↓ exited from C.S. P2: 4, 5, 6... ↓ enter C.S. ↓ exit

→ ME. is ensured.

→ No deadlock.

→ progress is violated. because before P1, P2 can't execute. specially alternating processes are running.

Algo-3:-

shared int flag[2] = [0,0];

P1

```
1: flag[0] = 1;
2: while(flag[1]);
3:     C.S.
4: flag[0] = 0;
```

→ when P1: 1, P2: 5 executed then

P2

```
5: flag[1] = 1;
6: while(flag[0]);
7:     C.S.
8: flag[1] = 0;
```

Algo 4:- (Dekker's Algo) or (Petersons Solution)

Shared int turn=0, flag[2] = {0,0};

P1

```
flag[0] = 1;
turn = 1;
while(flag[1] && turn==1);
        C.S.
flag[0] = 0;
```

P2

```
flag[1] = 1;
turn = 0;
while(flag[0] && turn==0);
        C.S.
flag[1] = 0;
```

Try context switching at various lines & we will get.

→ ME. is ensured.
→ Bounded waiting (P0, P1; P1)
→ Progress req is also ensured.
→ it is strictly working for 2 processes only.

H/W

①

P0

```
turn = 1;
flag[0] = 1;
while(flag[1] && turn==1);
        C.S.
flag[0] = 0;
```

P1

```
turn = 0;
flag[1] = 1;
while(flag[0] && turn==0);
        C.S.
flag[1] = 0;
```

* ME. is NOT ensured.

②

P0

```
flag[0] = 1;
turn = 0;
while(flag[1] && turn==1);
        C.S.
flag[0] = 0;
```

P1

```
flag[1] = 1;
turn = 1;
while(flag[0] && turn==0);
        C.S.
flag[1] = 0;
```

* ME. is NOT ensured.

③ Shared int x = 3;

P1

```
x++;
x++;
```

P2

```
x--;
x--;
```

Possible value of x are 1, 2, 3, 4, 5

④ Shared int x = 0;

P1

```
for(i=0; i<100; i++)
    x++
```

P2

```
for(j=0; j<100; j++)
    x--;
```

Possible value of x are -100 to +100.

⑤ Consider Dekker's algo. Why it can't work for 3 or more processes?

Sol^n

OS provided structure :-

Semaphore :-

→ OS provided semaphore & 2 atomic operations.

I) wait()	II) signal()
or	or
down()	up()
or	or
P()	V()

→ OS generally switches off all interrupts when wait() & signal() are executed & they run at kernel level. so that context switch cannot take place.

eg:-

semaphore mutex = 1;

P1	P2
wait(mutex)	wait(mutex);
C.S.	C.S.
signal(mutex);	signal(mutex);

call to OS code

2 types of semaphore :-

1) Counting semaphore :-

→ Semaphore ~~value~~ variable value indicates no's of processes can run C.S. at a time.

→ semaphore var. value can go in negative.

functional description counting semaphore

```
wait (semaphore mutex)
{
    mutex--;
    if(mutex<0){
        add calling
        process p
        to mutex.queue();
        block(p);
    }
}
```

```
signal (semaphore mutex){
    mutex++;
    if (mutex<=0){
        remove from (P1)
        of mutex.queue();
        wakeup(P1);
    }
}
```

* This code is inside O.S., not at programmer's level.

eg:

semaphore mutex = 1;

P1	P2	P3
wait(mutex);	wait(mutex);	wait(mutex);
C.S.	C.S.	C.S.
signal(mutex);	signal(mutex);	signal(mutex);

Q Try the above code for mutex = 2.
It will allow 2 processes in C.S. at a time.

Q semaphore m = 0;

P1	P2
print("0");	Print("2");
wait(m);	signal(m);
Print("1");	Print("3");

Q

```
Semaphore mutex[10] = 0;
Signal (mutex[0]);

Pi
while(1){
    wait (mutex[i]);
        C.S.
    Signal ( mutex[(i+1) % 10]);
}
```

Order of execution of processes:- $P_0, P_1, P_2 ---- P_9$

Q

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

P1                          P2
while (true){               while(true){
    P(a);                       P(b);
    print("1");                 print("0");
    V(b);                       V(a);
}                           }
```

Sol^n 101010 ----

2) Binary Semaphore:-

→ Semaphore var value can be either 0 or 1.

→ Only one process can access C.S. at a

functional description of binary semaphore

```
wait(BinarySemaphore M){
    if(M==1)
        M=0;
    else{
        put the process P
        in M.list();
        block(P);
    }
}
```

```
Signal (BinarySemaphore M){
    if( M.list().isEmpty)
        M=1;
    else{
        select a process
        from M.list();
        wakeup(P);
    }
}
```

Eg:-

```
Semaphore mutex = 1;

Pi = 1,---- 9                   P10
while(1){                       while(1){
    wait(mutex);                    signal(mutex);
        C.S.                            C.S.
    signal(mutex);                  signal(mutex);
}                               }
```

How many processes can be present in C.S. at max? All

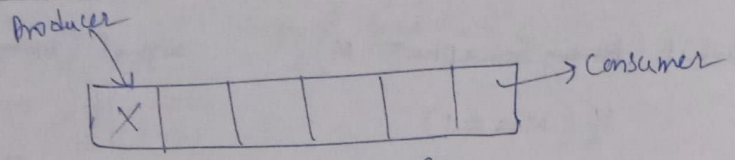
Sol^n let say all processes came together.

P_1: mutex = 0 goes inside C.S.

$P_2, P_3 ---- P_9$ gets blocked.

P_{10}:- signal the mutex & wakeup the P_2 & both P_2 & P_1 are present in C.S.

Producer – Consumer problem:-



Size = 6.

semaphore empty_size = size;
filled_space = 0;

Producer

```
while(1){
    wait(Empty_space);
    Put an item in
    Buffer.
    Signal(filled_space);
}
```

Consumer

```
while(1){
    wait(filled_space);
    Consume an
    item.
    signal(empty_space);
}
```

Empty_space = 5, Empty_space = 6

C_1 P_1 C_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 C_2 P_8

Block (↑ P_8) wakeup (↑ C_2)

C_1 ↓ Block

C_1 ↓ wakeup

P_7 ↓ empty_space = 0, filled_space = 6

Q semaphore mutex = 1;

producer

```
wait(mutex)
wait(empty_space);
    C.S.
signal(mutex)
signal(filled_space);
```

Consumer

```
wait(mutex)
wait(filled_space);
    C.S.
signal(mutex);
signal(empty_space);
```

let say all items are filled & buffer is full.
⇒ Producer cannot put anymore items in buffer.
Now let say producer comes so it will wait at wait(empty_space). while mutex=0.
If consume comes then it will wait at wait(mutex) because producer have already kept it. So it is in deadlock.

Reader writer problem:-

→ Two or more readers can read simultaneously.
→ Writer can't write when a reader is reading or another writer is writing.

→
Read – Read	✓
Read → write	✗
Write → write	✗

eg:-

writer

```
wait(mutex);
write;
signal(mutex);
```

Reader

```
wait(mutex);
read;
signal(mutex);
```

Problem:- 2 reader can't read.

eg:-

Semaphore mutex = 1;
int reader_count = 0;

Writer	Reader
wait (mutex)	reader_count ++;
	if (reader_count == 1) wait(mutex);
write;	Read
signal (mutex);	read_count --;
	if (reader_count == 0) signal (mutex);

Problem:- $R_1 \rightarrow$ is restricted

$R_2 \rightarrow$ entered for reading while W_1 is writing.

Q What's wrong with the following code.

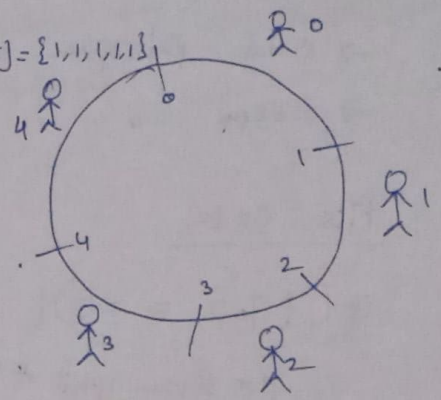
Semaphore S1=1, S2=100;

Writer	Reader
wait(S1);	wait(S1);
	wait(S2);
write;	Read;
signal(S1);	Signal(S2);
	signal (S1);

If R_1 & R_2 comes then because S1=1 R_1 is allowed but R_2 & further reader

Dinning Philosopher problem :-

Attempt ① Semaphore chopstick[5] = {1,1,1,1,1};



Philospher i's code

```
Acquire right chopstick;
    wait( chopstick[i]);
Acquire left chopstick;
    wait( chopstick[i+1] %5);

    Eat

    Signal(chopstick[i+1] %5);
Release left chopstick;
    Signal(chopstick[i]);
Please right chopstick;
```

* The above code will result in deadlock. (when all person take one chopstick then all person will wait for another chopstick & can lead to deadlock).

Attempt ②

Semaphore mutex = 1;

Pi

```
wait (mutex);
wait(chopstick[i]);
wait (chopstick[(i+1)%5]);
Signal(mutex);
```

```
signal (chopstick[i]);
signal (chopstick[(i+1)%5]);
```

* poor resource utilisation.
* let Po acquired 0 & 1.

Attempt ③

→ Odd philospher will pick left chopstick first.
→ even " " " right " ".

Pi's code

```
if(i%2==0){
    /* Even phil */
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
}
else{
    // odd philospher.
    wait(chopstick[(i+1)%5]);
    wait(chopstick[i]);
}
```

let i=0 came then he will acquire 0 (first) & 1 chopstick. if i=1 came then he will acquire 2 (first) & 1 chopstick.

Advantage :- No deadlock.
Disadvantage :- Starvation. (If only even philospher's are coming then odd will get chance to eat. &

H/w

Q Analyze the above code for 3 philosphers with the following sequence.
a) P_1, P_2, P_3.
b) P_1, P_3, P_2
c) P_2, P_3, P_1

Use of Dinning philospher problem :-

a) Five processes sharing 5 resources.
b) Shared m/m variables updated by 2 processes.

Hardware based solution :-

a) Test And Set :-

Atomically returns the original value & set the var to 1.

Functional description

```
Int TestAndSet(int &x){
    int temp = *x;
    *x = 1;
    return temp;
}
```

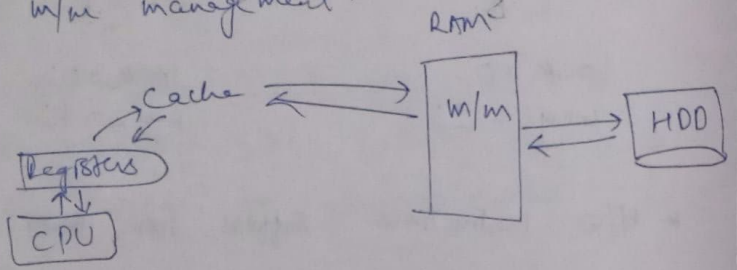
Memory management

Why memory management is required?

→ In a system we have; Registers, cache, m/m & HDD. In order to keep all levels work in sync. m/m mng. is required.

→ To increase the CPU utilisation proper use of main m/m is req. hence. m/m mng. is necessary.

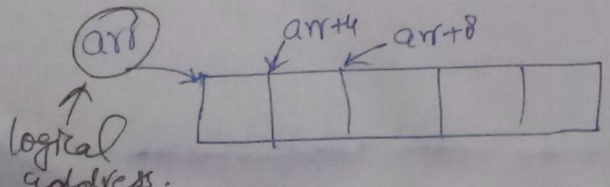
→ How to allocate m/m to a process is called m/m management.



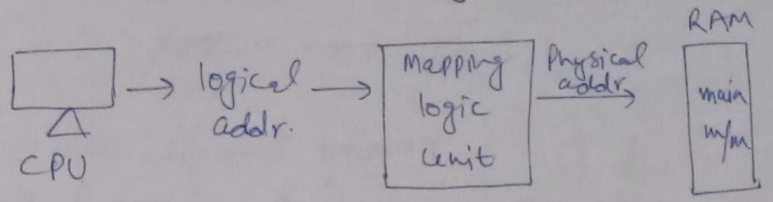
→ m/m can be accessed by addresses only.

Addresses → logical address
→ Physical "

eg:-
```
int x=20;
int *ptr=&x;
cout << ptr
```
→ virtual or logical addl.



Q How does address binding take place?



Mapping logic unit :-

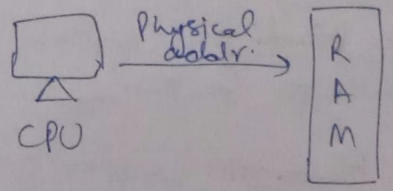
→ It consist of compiler, Assembler, Linker, loader.

→ Modern compilation process also involve assembler & linkers.

→ Object code is o/p of whole process which consist of relocatable addr. / logical addr.

→ loader will load the program into main m/m.

What if CPU directly gives physical addresses?



→ If CPU directly gives physical addr. then each process is given physical addr. during compile time or load time.

(I) let $P_1 \longrightarrow$ 200k – 550k

$P_2 \longrightarrow$ 400k – 700k

If P_1 is running then P_2 can't run.
It will degrade the degree of multiprogramming

(II) It is difficult to increase m/m at run time.
⇒ Difficulty in dynamic m/m allocation

(III) Difficult to relocate or difficult in swap in & swap-out.
Because each process will have to get same m/m partition ~~each~~ every time.

Address Binding :-

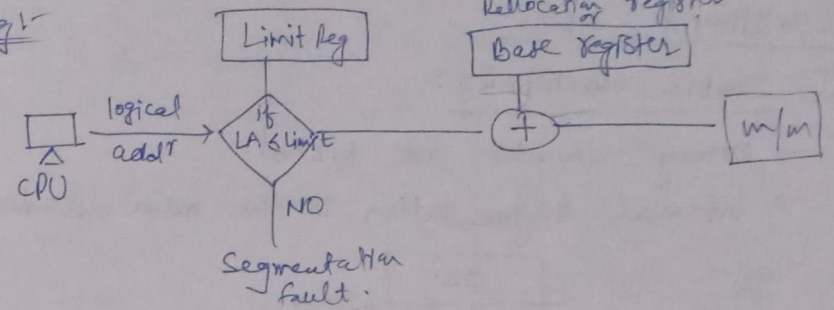
process of converting logical address into physical address.

(1) **Compile time Binding :-** Compiler assigns physical address to m/m location.

(2) **load time Binding :-** Compiler assigns logical addr but loader assigns physical addr. during loading a process

(3) **Execution time Binding :-**
Compiler & loader assigns logical addr. During run time, logical addr is converted into physical

eg :-



→ OS sets the value of Limit register & relocation reg. before executing a process.

Memory Allocation :-

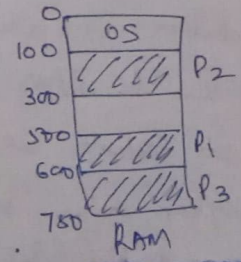
Memory can be allocated in RAM in 2 ways.

(1) Contiguous memory allocation.
(2) Non-contiguous " "

Contiguous memory allocation :-

When a whole process in kept in contiguous fashion in RAM ie. whole process in inside a chunk of memory called partition.

eg :- $P_1 \longrightarrow$ 100KB $P_2 \longrightarrow$ 200KB $P_3 \longrightarrow$ 150KB



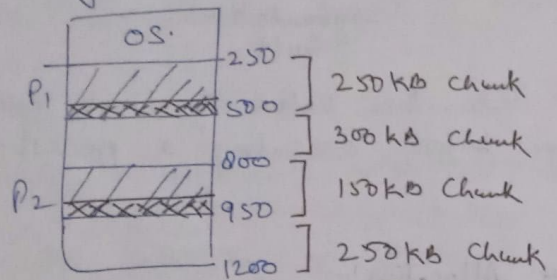
Partitioning types!-

① Static Partitioning!-

→ Memory chunks are fixed.

→ Internal fragmentation is the main disadvantage

eg!-



P_1 required 220KB
But chunk is of 250KB
So, 30KB is wasted.
Called Internal fragmentation.
Same is the case with P_2 also.

② Dynamic Partitioning!-

→ Memory chunks are not fixed.

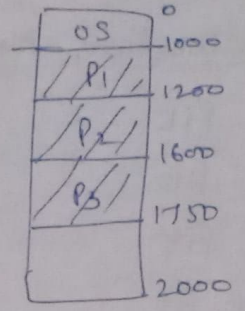
→ As per process req. chunk can be created & allocated to a process.

→ When process leaves the m/m, it creates the hole in between. which leads to External fragmentation.

→ External fragmentation is when the enough m/m is available in RAM but since it is not contiguous so can't be allocated to a process with higher m/m needs.

~~External fragmentation~~ in this method.

eg!- $P_1 \rightarrow 200KB$
$P_2 \rightarrow 400KB$
$P_3 \rightarrow 150KB$
$P_4 \rightarrow 600KB$

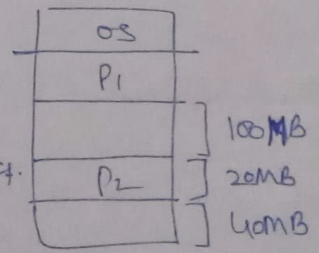


Now P_2 leaves.
& make a hole of 400 KB.
In total we have 650 KB available but it is not contiguous.
P_4 arrives with a req. of 600KB, which can't be allocated to RAM.
This is external fragmentation.

→ Solution to external fragmentation is compaction.
Shifting the process upwards to join 2 or more free holes in RAM is called compaction.

→ It is expensive in ~~time~~ terms of time.

eg!- We want to shift P_2 to make 1 chunk of 140MB.
let 1B → 4ns. to shift.



⇒ 20MB shifting will take

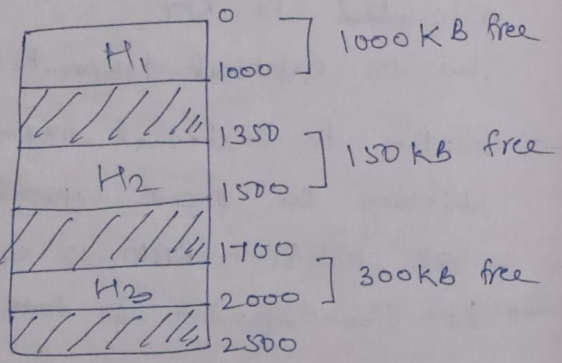
$= 20 \times 10^6 \times 4 \times 10^{-9}$ sec

$= 80 \times 10^{-3}$ sec

Memory allocation Algo in Contiguous Memory allocation

1. First fit
2. Next fit
3. Best fit
4. Worst fit
5. Quick fit.

eg:- Consider the following snapshot of memory at particular moment.



In this moment P_1 with a need of 100KB comes.

In
First fit → P_1 is allocated to H_1 (whichever first block of req size or more is available)

Next fit → P_1 is allocated to next pointer to available hole.

Next fit means whenever the pointer is at last time it will search for next possible accomodation hole.

Best fit:- P_1 is allocated to H_2.
(Best suitable for process req). Smallest possible which can accomodate a process.

Worst fit:- P_1 is allocated to H_1.
(largest chunk available).

Quick fit:- In Quick fit we have a LL pointing to the frequently used m/m size chunks. It will take the smallest m/m chunk pointer from that LL & allocate it to P_1.

General notes:-

→ First fit is most widely used technique.
→ Best fit is not always give best result.
→ Quik fit is having overhead to manage extra Linked List to freq. used m/m blocks.
→ Best fit & worst fit, searching is time consuming.

Overlays:- If we can't accomodate whole process in memory, we can divide the mutually exclusive code in different parts & then run it.

eg: P_1 is a process.

Pass 1: 100kB

Pass 2: 200kB

Common routines → 50kB

Symbol tables → 40kB

Pass driver → 10kB

so total m/m req = 100 + 200 + 50 + 40 + 10
= 400kB

But since Pass1 & pass2 can run independly

so, m/m req is

pass 1 = 100 + 50 + 40 + 10
= 200kB

pass 2 = 200 + 50 + 40 + 10
= 300kB

So, in 300kB m/m also P_1 can be completed.

→ using overlays, we can reduce the m/m req upto some extent.

Non-contiguous memory allocation:-

let say

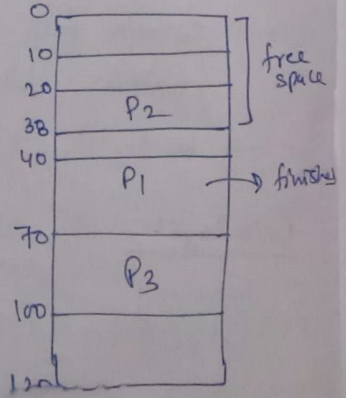
P_1 → 16KB

P_2 → 16KB

P_3 → 32KB

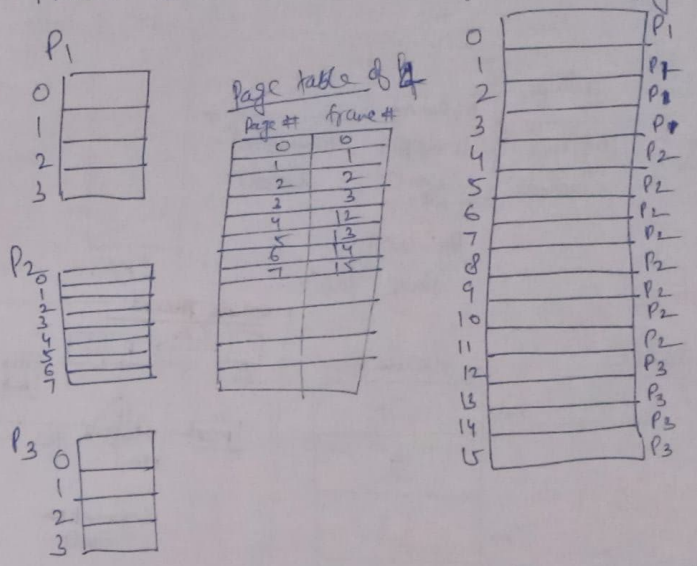
P_4 → 64KB

Cannot allocate contiguous.



→ process is divided into pages.

→ frame size must be equal to page size.



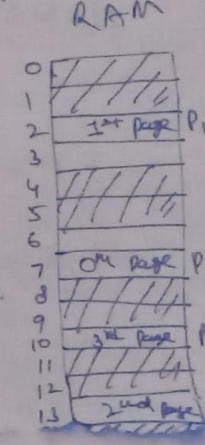
After some time P_1 & P_3 completes & P_4 comes with a req of 8 pages.

we can allocate it in non-contiguous fashion.

But it require maping of page to frame table called page table.

Because it require converting a logical addr to physical addr.

→ whole process picture here (shown in next page)



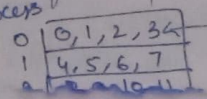
eg: Process P_1 = 16KB.

Page size = frame size = 4B

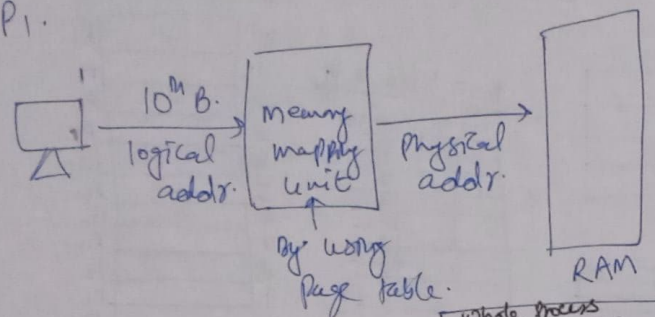
⇒ memory size = 64B

⇒ no's of frames = 64/4 = 16 frames.

⇒ no's of page req = 16/4 = 4 pages.

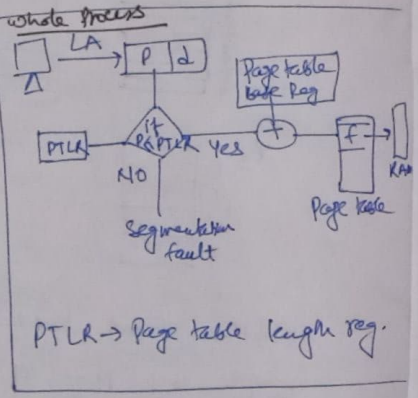


what is logical physical addr of 10th Byte of P1.



Page table

	Frame #
0	7
1	2
2	13
3	10



logical addr of 10th Byte = 10

Page no's of 10th Byte = $\lfloor 10/\text{page size} \rfloor = \lfloor 10/4 \rfloor = 2$

offset of 10th Byte in the page
= 10 % 4 = 2

Frame no's = 13 (By looking at page table)

Physical addr = Frame no's × frame size + offset
= 13 × 4 + 2
= 54.

logical addr = | Page no's | offset |

Physical addr = | Frame no's | offset |

max pages = 4 ⇒ 2 bits are req.
max page size = frame size = 4B = 2 bits are req.
max frame = 16 = 4 bits are req.

⇒ logical addr

Page #		offset	
1	0	1	0

⇒ Physical addr.

Frame #			offset		
1	1	0	1	1	0

H/W

(1) Page size = 1KB = 2^{10} B ⇒ 10 bits for offset.
logical address = 20 bits.

a) Draw virtual addr. format.

Page #	offset
10	10

b) How many max pages a process can have?
$2^p = 2^{10}$

c) If there are total 2^{16} frames in m/m. Draw physical addr. format?

16	10
frame #	offset

(2) Memory size = 2MB = 2^{21} = 21 bits for Physical addr
Page size = 512 Bytes = 2^9 ⇒ 9 bits for offset

a) How many pages are allocated to a process P1 of 5000 Bytes? $\lceil 5000/512 \rceil$ = 10 pages.

(b) How much m/m is allocated to P_1?

10 pages × 512 B (size of each page) = 5120 B

(c) Max size of process = 8 MB = 2^{23} B ⇒ total bits for VA = 23 bits.

Draw Virtual addr & Physical addr format.

(d) How many frames are there in m/m?

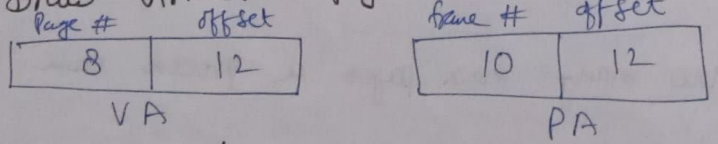
$2^f = 2^{12}$ frames

(3) Page size = 12 Bytes ⇒ offset is of 12 bits.

Total frames in m/m = 1024 = 2^{10} ⇒ 10 bits for frame #

Total pages req = 2^8.

a) Draw Virtual & physical addr format.



b) Size of m/m.

$2^{f+d} = 2^{10+12} = 2^{22} = 4$ MB.

c) Max size of process.

$2^{p+d} = 2^{8+12} = 2^{20} = 1$ MB.

d) Page table size = $2^8 \times 8$ bits.

Summary:-

Max nos of pages = 2^p

Page size = 2^d

Max size of process = 2^{p+d}

max nos of frames = 2^f

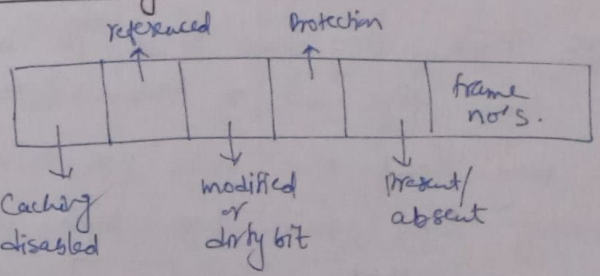
frame size = 2^d

max size of memory = 2^{f+d}

→ sometimes Logical addr space is reffered as process size & physical addr space is reff. as m/m size.

size of page table = (no's of pages × bits req for each entry) bits.

Page table entry details:-



Gatte Que 2004.

2-level paging! =

let say LA = 22 bits

⇒ process size = 2^{22} Bytes.

Page size = 1KB

⇒ offset = 10 bits.

Total nos pages = 2^{22-10}

$= 2^{12}$

Now let say each page table entry is of 4 Bytes.

so Total size of Page table 2

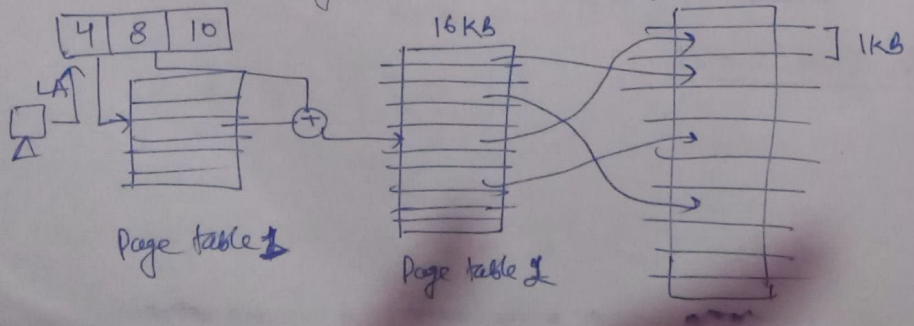
$= 2^{12} \times 4$ B

$= 2^{12} \times 2^2 = 2^{14}$ B

$= 16$ KB.

This page table also has to be in main memory but page size & frame size is 1KB.

So we may need one more page table for referencing this main page table.



Now let say each entry of page table 1 is also of 4 B.

Total size of Page table 1 =

nos of entries req for storing Page table 2 pages
16KB / 1KB = 16

nos of entries × 4 B.

→ 16 × 4 = 64 Bytes.

This can be reffrenced into 1 frame entry.

Que 1

	VA	Page size	Page table entry	PT1 size	PT2 size	PT3 size	address format
I)	48Bit	16KB	4B	2^{36}B	2^{24}B	2^{12}B.	10 | 12 | 12 | 14
II)	64bit	1MB	"	2^{46}B	2^{28}B	2^{10}B	8 | 18 | 18 | 20
III)	72bit	1GB	"	2^{44}B	2^{16}B	—	14 | 28 | 30
IV)	72bit	256MB	"	2^{46}B	2^{20}B	—	18 | 26 | 28
V)	72bit	16MB	"	2^{50}B	2^{20}B	2^{6}B	4 | 22 | 22 | 24

① VA = 48bits.

Page size = 16 KB

⇒ 2^{14} B ⇒ 14 bits is for offset.

max nos of pages = $2^{48-14} = 2^{34}$

1 entry = 4B.

⇒ total entries in each Page = $\frac{2^{14}}{2^2} = 2^{12}$ entries.

Page table 1 size = total entries × 4 = $2^{34} \times 4 = 2^{36}$ B.

Page table 2 size = $\frac{2^{36}}{2^{14}} \times 4$ ⇒ $\frac{\text{total page table 1 size}}{\text{Page size}}$

$= 2^{22} \times 4 = 2^{24}$ B.

Total size of page table 3

$= \frac{2^{24}}{2^{14}} \times 4$

$= 2^{10} \times 2^2 = 2^{12}$ B.

nos of entries in PT 3 $= \frac{2^{12}}{2^2} = 2^{10}$

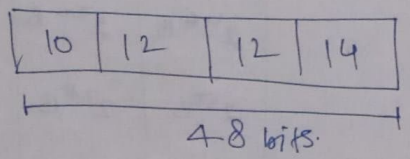
= 10 bits req.

nos of entries in one page of PT2 $= \frac{2^{14}}{2^2}$
(Same is the case for PT1 also)

$= 2^{12}$

⇒ 12 bits req.

⇒ VA =



(ii) VA = 64 bits

Page size = 1MB $= 2^{20}$ B.

⇒ 20 bits is offset.

⇒ max nos of pages $= 2^{64-20}$

$= 2^{44}$.

Page table entry = 4Bytes.

Page table 1 size

$= 2^{44} \times 4 = 2^{46}$ B.

Page table 2 size

$= \frac{2^{46}}{2^{20}} \times 4 = 2^{28}$ B.

Page table 3 size

$= \frac{2^{28}}{2^{20}} \times 4 = 2^{10}$ B. < Page size

so no further page tables req.

nos of entries in PT3 $= \frac{2^{10}}{2^2} = 2^8$ entries

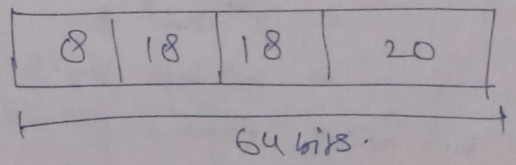
1 page of " " PT2 ⇒ 8 bits req.

$= \frac{2^{20}}{2^2} = 2^{18}$ entries

⇒ 18 bits req.

(Same case for PT1 also)

⇒ VA =



(iii) VA = 72 bits.

Page size = 1GB

⇒ 2^{30} B ⇒ offset = 30 bits.

⇒ max nos of pages $= 2^{72-30}$

$= 2^{42}$ pages.

nos of entries in each page

$= \frac{2^{30}}{2^2} = 2^{28}$ entries.

size of PT1 $= 2^{42} \times 4 = 2^{44}$ B.

" " PT2 $= \frac{2^{44}}{2^{30}} \times 4 = 2^{14} \times 4 = 2^{16}$ B.

< page size

so no more page table req.

nos of entries req. in PT2

$= 2^{14} \Rightarrow$ 14 bits req.

VA.

14	28	30

(v) VA = 72 bits.

Page size = 256 MB.

$= 2^{20} \times 2^8$ B

= 28 bits req. for offset.

max nos of Pages = $2^{72-28} = 2^{44}$ Pages.

nos of entries each page = $\frac{2^{28}}{2^2} = 2^{26}$ ← size of each entry

PT1 size = $2^{44} \times 2^2 = 2^{46}$ B.

PT2 size = $\frac{2^{46}}{2^{28}} \times 2^2 = 2^{20}$ B. < page size

so no more page table

nos of entries in page table 2 = $\frac{2^{20}\text{ B}}{2^2} = 2^{18}$

$\Rightarrow$ 18 bits req.

VA

18	26	28

72 bits.

(i) VA = 72 bits.

Page size = 16 MB

$\Rightarrow 2^{24}$ B

$\Rightarrow$ 24 bits for offset

max nos of pages = $2^{72-24} = 2^{48}$ pages.

nos of entries per page = $\frac{2^{24}}{2^2} = 2^{22}$.

PT1 size = $\frac{2^{72}}{2^{24}} \times 2^2 = 2^{50}$ B.

PT2 size = $\frac{2^{50}}{2^{24}} \times 2^2 = 2^{28}$ B.

PT3 size = $\frac{2^{28}}{2^{24}} \times 2^2 = 2^6$ B. < page size.

nos of entries req ~~for~~ in PT3 = $\frac{2^6}{2^2} = 2^4$

$\Rightarrow$ 4 bits req.

VA.

4	22	22	24

72 bits.

Q*

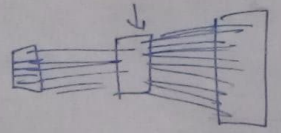
	Page size	PTE	Outer page table size	levels of Paging	virtual addr space (max ~~outer~~ process size)
(I)	2KB	4B	2KB	1	1MB.
(II)	2KB	4B	2KB	2	512MB
(III)	2KB	4B	2KB	3	256GB
(IV)	2KB	4B	128B	1	64 KB
(V)	2KB	4B	128B	2	32MB
(VI)	2KB	4B	128B	3	16 GB

(I) Page size = 2KB
$= 2^{11}$ B
$\Rightarrow d = 11$ bits for offset.

no's of entries in page table $= \frac{2^{11}}{2^2} = 2^9$
$\Rightarrow p = 9$.

max process size $= 2^{p+d} = 2^{11} + 2^9 = 2^{20}$
$= 1$ MB

(II)



Page size = 2KB
$= 2^{11}$ bytes.
no's of entries in one page $= \frac{2^{11}}{2^2} = 2^9$.

$2^9 \times 2^9 \times 2^{11}$ → offset
↓ ↓ ↓
sized each page
$= 2^{29}$

(III) $2^9 \times 2^9 \times 2^9 \times 2^{11}$
$= 2^{38}$ B $=$ 256 GB

(IV) Page size = 2KB
$\Rightarrow$ 11 bits offset.
entries in each page $= \frac{2^{11}}{2^2} = 2^9$ entries.
Outer page table size = 128B
$= 2^7$ B.
$\Rightarrow$ it is not fully filled.
$\Rightarrow$ max process size $= \frac{2^7}{2^2} \times 2^{11} = 2^{16}$
$=$ 64KB.

(V) $2^5 \times 2^9 \times 2^{11} = 2^{25} = 32$MB.
2^5: outer table which is not fully filled
2^9: fully filled.

(VI) $2^5 \times 2^9 \times 2^9 \times 2^{11}$
$= 2^{34}$ B $=$ 16 GB.

Virtual m/m:-

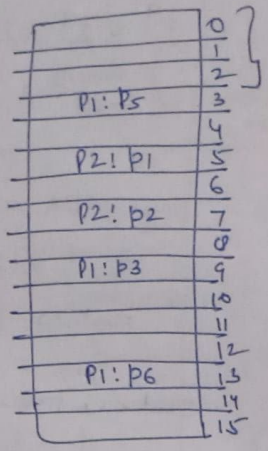
A part of secondary m/m treated as main mem to increase the size of physical m/m is called virtual m/m.

eg:- P_1 req → 8 Pages
P_2 " → 4 "

	frame#	V/I	R	D
0		0	0	0
1		0	0	0
2		0	0	0
3	9	1	1	1
4		0	0	0
5	3	1	0	0
6	13	1	1	0
7		0	0	0

(D: modified bit)

P_1 Page table

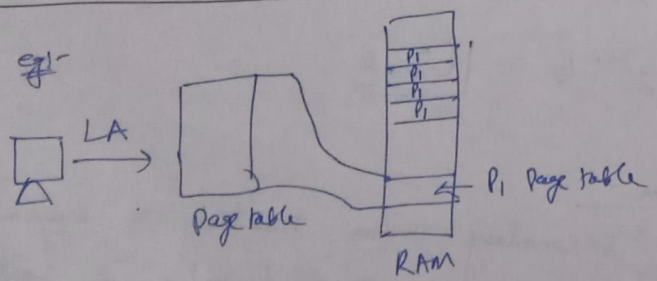


RAM

	frame#	V/I	R	D
0		0	0	0
1	5	1	1	0
2	7	1	0	1
3		0	0	0

P_2 page table

Translation lookaside Buffer (TLB):-



Effective access time = time for searching a frame# in page + time to access the frame

If m = 100ns

→ EAT = 2×100ns
= 200ns.

for 2 level of paging EAT = 2×m + m
(2×m: time for 2 page tables; m: time for frame access.)
= 3m.

Now, if we want to decrease the EAT, we can use TLB. (cache).

so, instead of storing the page tables inside the main m/m. we can store page table or part of it inside the cache called TLB for faster access.

EAT = time to access page table + time to access m/m.
for 1-level paging in TLB

$= (h)(t_{TLB} + t_m) + (1-h)(t_{TLB} + t_m + t_m)$

h: hit ratio; $(t_{TLB} + t_m)$: if page is found in TLB
(1-h): miss ratio; $(t_{TLB} + t_m + t_m)$: if page is not found in TLB

eg:- t_{TLB} = 20ns. hit ratio = 80%
t_m = 100ns.
1-level paging.

EAT = (.8)(20+100ns) + (.2)(20+100+100)
= 96ns + 44 = 140ns.

EAT for K-level Paging = $(h)(t_{TLB}+t_m) + (1-h)(K \times t_m + t_m)$

eg:- For 2-level paging.
$t_{TLB} = 20ns$.
$t_n = 100ns$.
$H = 80\%$

Without TLB.

$EAT = 3 \times t_m$
$= 3 \times 100$
$= 300ns$.

With TLB

$EAT = (.8)(20+100) + (.2)(2 \times 100 + 100)$
$= 96 + 60 = 156ns$.

Que: Gate(2008)

A paging scheme uses TLB. A TLB access takes 10ns and main m/m access takes 50ns. What is the effective access time, if the TLB hit ratio is 90% & there is no page fault?

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

Sol^n $t_{TLB} = 10ns$, $t_m = 50ns$, $H = .90$

$EAT = (0.9)(10+50) + (0.1)(10+50+50)$
$= 54 + 11 = 65ns$.

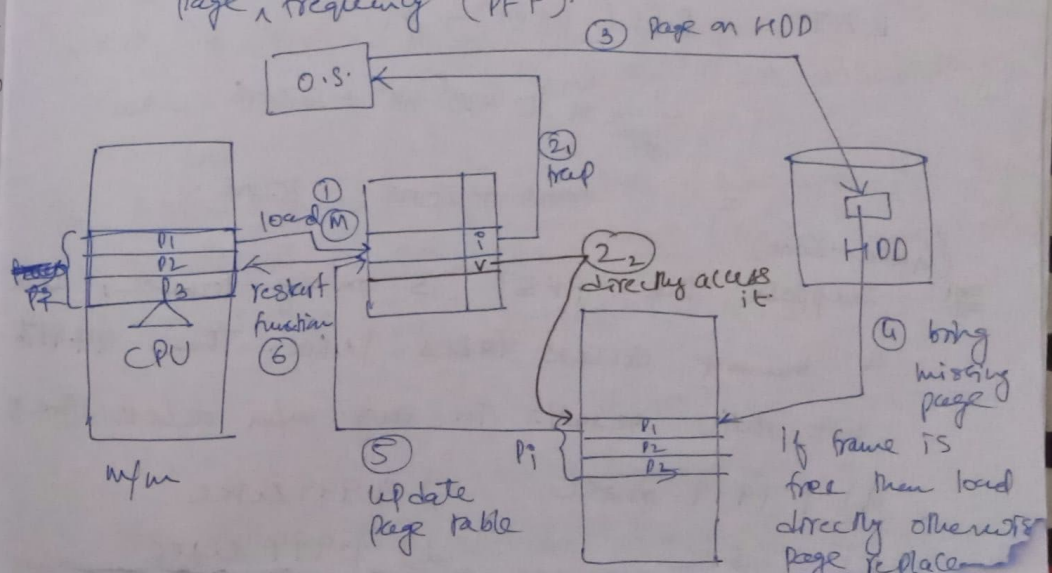
Demand Paging

Processes generally resides on HDD. Instead of loading all pages of a process in m/m. We can load whichever page is req. then & there itself on demand.

→ Generally a process have fixed no's of frames allocated to it. When a page is available in m/m, process can access it. If not then it has to bring into m/m using page fault.

→ Time taken to do page fault is called Page fault servile time (PFST)

→ No's of page fault per x pages is called Page fault frequency (PFF).



Effective access time (EAT)

$$= \frac{Pff}{p} * \left(\frac{PFST}{PS} + \frac{tm}{m}\right) + \left(1 - \frac{PFF}{p}\right)\left(\frac{tm}{m}\right)$$

$$= p(PS) + p*m + m - p*m = p(PS) + m$$

(Gate-2011)

eg:- let the page fault service time be 10ms in a system with avg m/m access time being 20ns. If one page fault is generated for every 10^6 m/m accesses. what is the EAT for m/m?

a) 21ns b) 30ns c) 23ns d) 35ns.

Sol^n

PFST = 10ms

tm = 20ns

Pff = 1 in 10^6

EAT = Pff(PFST) + M

$$= \frac{1}{10^6} * 10 \times 10^6 \text{ ns} + 20\text{ns}$$

$$= 10\text{ns} + 20\text{ns} = 30\text{ns}.$$

(Gate-2000)

eg:- Suppose the PFST is on avg 10msec, while a memory access takes 1μsec. Then 99.99% hit ratio results in avg. m/m access time of

a) 1.9999 msec c) 9.999 μsec

b) 1msec d) 1.999 μsec

Sol^n

PFST = 10msec

tm = 1μsec

H = 99.999%

⇒ Pff = 0.0001 or 0.01%

EAT = Pff * PFST + tm

$$= 0.0001 \times 10 \times 10^{+3} \mu\text{sec} + 1\mu\text{sec}$$

$$= 10^{-4} \times 10^4 \mu\text{sec} + 1\mu\text{sec}$$

$$= 2\mu\text{sec}.$$ (2) Ans.

Q: (Gate-2003)

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

10	10	12

VA

PTE = 4B.

TLB has hit ratio of 96%. cache has hit rate of 90%. Main m/m access time is 10ns. Cache access time is 1ns & TLB access time is 1ns.

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

a) 1.5ns 2) 2ns 3) 3ns 4) 4ns.

Sol^n EAT = VA to PA + access word.

$$= h(t_{TLB}) + (1-h)(t_{TLB} + t_{em}) + h_{cache}(t_{cache}) + (1-h)(t_{cache} + t_m)$$

(b) Suppose a process has only the following pages in virtual addr. space: 2 contiguous code pages starting at virtual addr 0X00000000, two contiguous data pages starting at virtual addr 0X00400000 & a stack page at virtual address 0XFFFFF000. The amount of m/m required for storing page tables of this process.

a) 8kB b) 12kB c) 16kB d) 20kB

Soln

d = 12 bits.

⇒ page size = 2^{12} B = 4kB.

PTE = 4B.

⇒ no's of entries in 1 page table = $\frac{2^{12}}{2^2}$ = 2^{10} entries.

Code pages:

0X 00000|000 (page no's | offset) to 0X 00000|FFF] one page.

0X00001|000 to 0X00001|FFF] second page

Here Outer page table is having same entry for 2 different entries but a page can contain 2^{10} entries. So 1 access these pages. We need 1 page at outer page table & 1 page at inner page table.

Data pages

0X00400|000 to 0X00400|FFF] 1st page

0X00401|000 to 0X00401|FFF] 2nd page.

Here also outer page table is having same bits. i.e. 0000000001 (outer is just next entry to 0000 of last entry).

Inner page table entry is differing to same other page. because both are diff entries. so it req. diff. page for inner page table.

Stack page:-

0X fffff|000 to 0X fffff|fff] 1st page.

Outer page table is same but inner page table is diff. again.

→ 1 page for Outer page table + 3 pages for inner page

Q GATE -2004

Consider a system with a 2-level paging scheme in which a regular m/m access takes 150nsec & servicing a page fault takes 8msec. An avg instr. takes 100nsec of CPU time & 2-m/m accesses. The TLB hit rate is 90% & page fault rate is one in every, 10000 instr. what is the effective avg. instr execution time?

a) 645 nsec b) 1050sec c) 1215 nsec d) 1250 nsec

Soln

Effective Instr execution time =
100ns + 2(EMAT)

EMAT = $h(t_{TLB}$ ~~+tm~~) + $(1-h)(t_{TLB} + k*t_m + t_m + PFF * PFST$

= $0.9(0) + (1-0.9)(0 + 2*150) + 150 + \frac{1}{10^4} \times 8 \times 10^6$ nsec

= $0 + 0.1 \times 300 + 150 + 800$ nsec

= 980 nsec.

EIET = 100 + 2(980)

= 2060 nsec

(12) Gate-2004

Consider a paging h/w with a TLB. Assume that the entire page table & all the pages are in the physical m/m. It takes 10 msec to search the TLB & 80 msec to access the physical m/m. If the TLB hit ratio is 0.6, the EMAT is _____.

Soln EMAT = $h*(t_{TLB} + t_m) + (1-h)(t_{TLB} + t_m + t_m)$

= $0.6(10+80) + (0.4)(10+80+80)$

= $90 \times 0.6 + 170 \times 0.4$

= 122 msec. Ans.

Page replacement Algorithms

① FIFO Page replacement algo.

(Gate-2014)

Q Total no's frames = 3

Page series = 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

No. of Page faults ?

Sol^n

Using FIFO

Page fault = 6.

4	1
7	2
6	7

Using LRU

Page fault = 6

4	1
7	2
6	7

Using Optimal

4	2
7	
6	

Page fault = 5.

(Gate-1995)

Q Reference string:-

0100, 0200, 0430, 0499, 0570, 0530, 0569

0120, 0220, 0240, 0260, 0320, 0370.

(PF, PF, PF, PF, PF, PF, PF)

100 records per page with 1 free main m/m frame.

Page faults = ?

Page mis.

Sol^n Since direct addr of pages are given.

So we have to find the page nos first.

only 1 free frame → so no choice.

⇒ Page faults = 7.

$$\text{miss rate} = \frac{\text{Page faults}}{\text{total ref.}} = 7/13$$

$$\text{hit rate} = 1 - \text{miss rate}$$

$$= 6/13.$$

(Gate-2012)

Q Ref string → 1, 2, 3, 2, 4, 1, 3, 2, 4, 1. frame = 3.

Page fault = ?

Sol^n

FIFO

1	4
2	1
3	2

PF = 6

miss rate = 6/10

LRU

1	4 2
2	3 1
3	4

PF = 9 miss rate

Optimal

1	
2	4
3	2

PF = 5

miss rate = 5/10

H/W

(Gate-2007)

Q① Ref string = 1, 2, 1, 3, 7, 4, 5, 6, 3, 1

frames = 3

Gate-2014

Q② Ref string = 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6

frames = 3.

Sol" ①

Sol ②

Belady's Anamoly

Eg Ref string = 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

a) Frame = 3 b) Frame = 4 using FIFO.

a)

0	4
1	2
2	3

Pf = 9

b)

0	3
1	4
2	1
3	2

Pf = 10

If on increasing frames, page fault increases it is called Belady's anamoly. It only occurs in FIFO.

General notes:-

① Sometimes FIFO suffers from Belady's Anomaly. Not always.

② LRU & Optimal follows stack algo but FIFO does not follow it always because of Belady's Anomaly occurs.

③ Optimal algo always give best results. Because it can ~~adj~~ tranform itself into LRU, MRU based on situation.

Gate-1994

Que:- A memory page containing a heavily used variable that was initialised very early & is in constant use is removed when FIFO is used.

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

Gate-2010

Que:- A system uses FIFO policy for page replacement. It has 4 page frames with no pages loaded to begin with. The system ~~first~~ 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^n Let say first it accessed 1, 2, 5, --- 100 since there is no page available m/m initially. So all will give page fault. Now when they were accessed in reverse order we have 100, 99, 98, 97 pages already in m/m so no page fault for 4 pages (since only 4 frames have 4 pages). & rest other 96 will have page faults.

So total page faults = 100 + 96
= 196 Ans.

Working Set Algorithm

→ It assumes that the nearest future is the close approximation of the recent past.

→ It is used to predict the no's of frames for a process dynamically (specially incase of blocked process)

→ Working set is the no's of pages it can look at a time in recent past.

→ Window is the max size of the working set.

eg:- string ref = 1, 2, 5, 1, 2, 4, 1, 4, 2, 1, 5, 1, 2, 4, 3, 1

window(A) = 4 , working set = ws.

$w = \{1\} = 1$ $w = \{1, 2, 5, 4\} = 4$ $w = \{1, 2, 4, 5\} = 4$

$w = \{1, 2\} = 2$ $w = \{1, 2, 4\} = 3$ $w = \{1, 2, 5\} = 3$

$w = \{1, 2, 3\} = 3$ $w = \{1, 2, 4\} = 3$ $w = \{1, 2, 5\} = 3$

$w = \{1, 2, 3\} = 3$ $w = \{1, 2, 4\} = 3$ $w = \{1, 2, 4, 5\} = 4$

$w = \{1, 2, 3\} = 3$ $w = \{1, 2, 4\} = 3$ $w = \{1, 2, 4\} = 3$

avg frame req = total of working set / total ref.

= (1+2+3+3+3+4+3+3+3+3+4+3+3+4+3+3)/16

(Gate-1992)

Que:-

Sol^n Ref string = c c d b c e c e a d

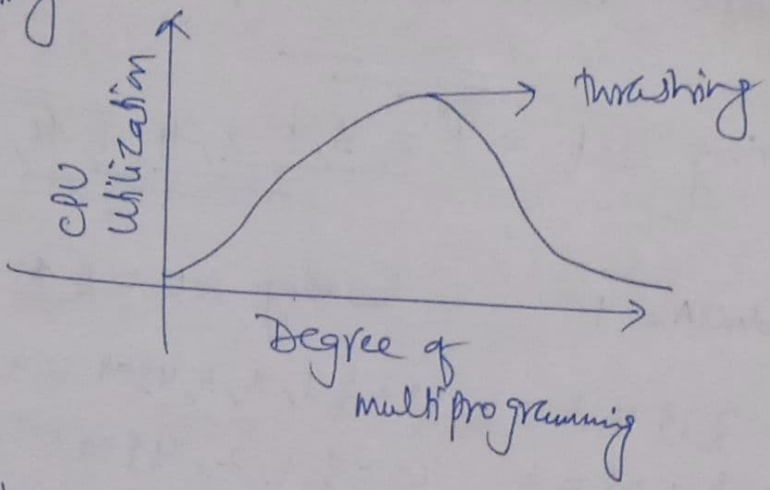
$w = \{a, d, e\}$, $\Delta = 4$

Thrashing:-

As degree of multi-programming increases, each process gets lesser m/m frames. Thrashing is a situation where most processes do not have their locality in a m/m & therefore they frequently do page faults.

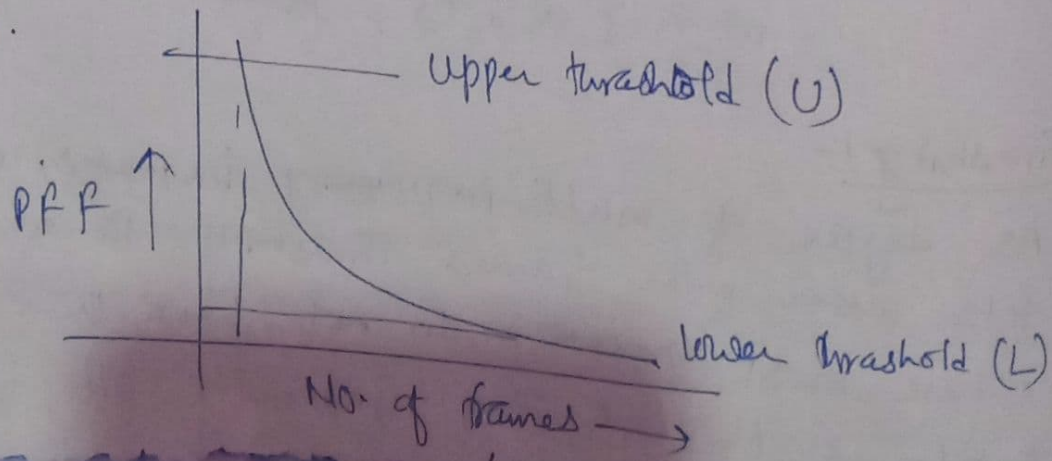
DOS advantage

→ Very CPU utilization.
→ HDD usage is very high.
→ very large no's of processes are running but none is completing.



Measure taken by OS

a) Pff base monitoring.
→ OS will monitor Pff of each process.
→ If $Pff_i > U$, OS will try to give frames from other processes.
→ If there are no frame available, OS will swap out process Pi.
→ If $Pff_i < L$, OS can take frames from proce Pi.



File management

File:- File is an abstract D.S. which holds the info about a collection of data.
A file typically consist of:

- → Name
- → Type
- → Location
- → Size
- → protection
- → time & date

following are the operation we can perform on a file.

- → creating a file
- → writing into a file
- → Reading a file
- → Move a file
- → Delete a file
- → Copy a file.

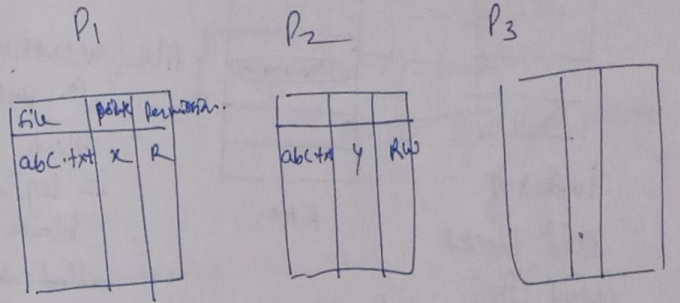
Directory:- It is also an abstract D.S. which holds the information about collection of files within in a partition. (refer pic on next page).

HDD:- Permanent storage device.

- → It is divided into ~~blocks~~ blocks. (Block size = page size)
- → ~~Blocks have tracks are further divided into sectors.~~
- → ~~Each sector is consisting of some info.~~

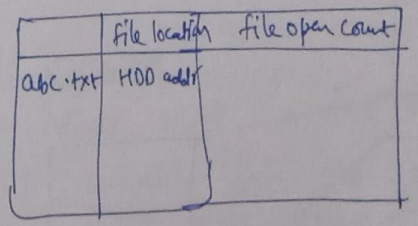
Information required for accessing a file

(I) file pointer
(II) file open count
(III) Disk location of a file
(IV) Access rights.



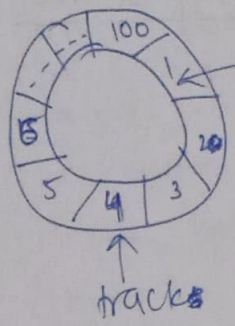
per process table. (local)

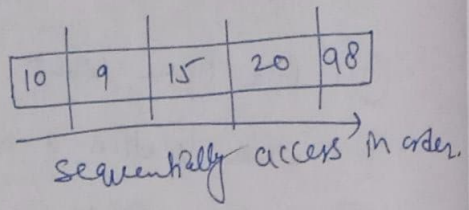
Global table for all the open files related to all processes.

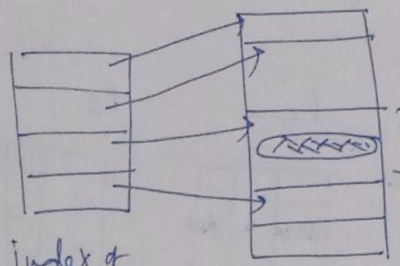


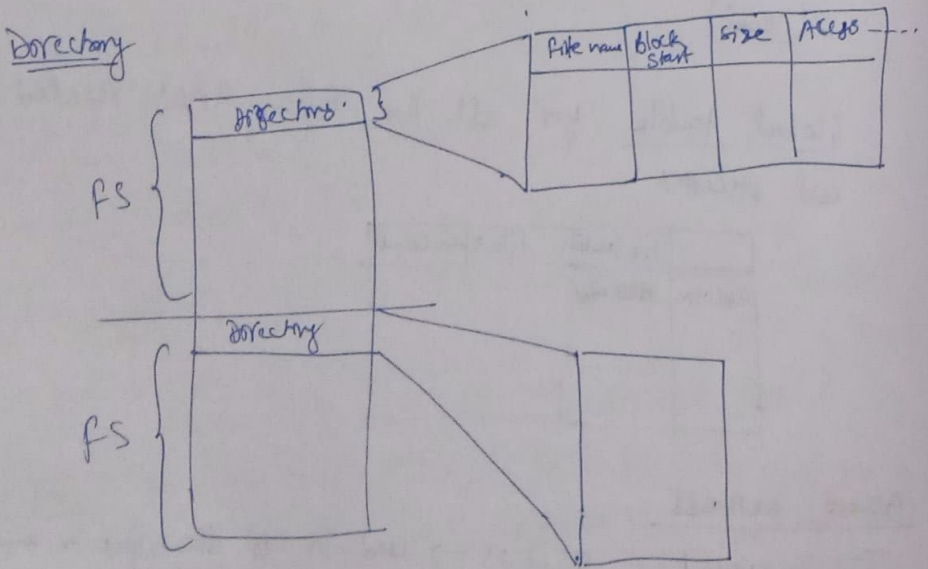
Access methods

(1) Sequential access → used for app like video or music file
(2) Direct access → used for DB like apps.
(3) Indexed → used for faster searching sce





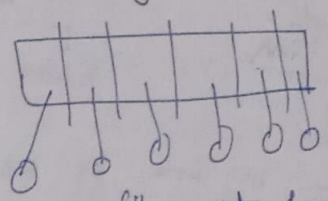




Operation supported by directory

- → Search
- → Create
- → delete
- → Rename
- → List
- → Traverse

Single level Directory :-



files at 1-level only.

Advantages :-

- → searching is easy.
- → Best for small size files.
- → Implementation is very simple.

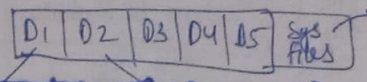
Disadvantage :-

- → Not suitable for large files.
- → Naming is difficult (can't have same name for 2 files).

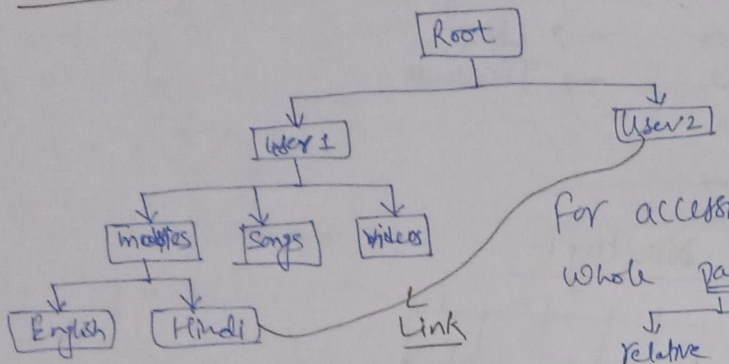
Two level Directory

- → At first level, master file directory (MFD)
- → MFD contains the info of various directory at level-2 & their access rights.
- → At level-2, various other directories various having data of one user in each or one type in each.

DFS:
→ Can't group same type of files under



Tree structure Directory



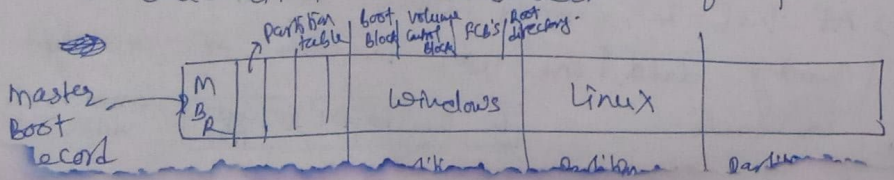
For accessing a file whole path is required.
- relative
- absolute

* Sharing of files can be done by using links.
* ~~Tree~~ Links type sharing is used in Linux.
* Because of these links a cycle is formed called acyclic graph.
* Deletion is difficult. (counter of links is used to keep track of it).

File System:-

A file system is a method or a data structure that OS uses to keep track of files on disk. It also allows the access to program & data.

File system helps
→ to keep track of file structures, location of data.
→ allocation & deallocation of space on disk.



Boot control block:-

→ It is a section available in each volume.
→ Used to load OS present in that volume.
→ In Unix it is called boot block.
→ In NTFS " " " partition boot sector.

Volume control block:-

→ It have info about no's of blocks, size etc
→ In Unix it is called super block.
→ In NTFS " " " master-file table.

Directory structure:-

→ Used to organize the files.
→ In Unix, this includes file names & associated Inode numbers.

FCB (file control block):-

→ It store details about the file.
like file permissions, date & time, size etc.

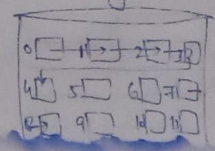
Allocation Methods:-

allocating space to files on disk efficiently so that files can be accessed faster.

1) Contiguous allocation:-

Allocating contiguous blocks.

Directory file	starting block #	length
abc.txt	0	5



Advantage

→ Reading is faster.
→ Easy to implement

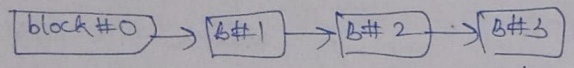
Disadvantage

→ External fragmentation is available.
→ Internal frag. is also available.
→ Can't increase the file size dynamically.

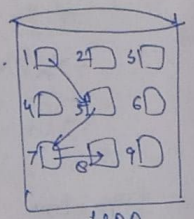
② Non-Contiguous allocation:-

Linked list allocation:-

abc.txt



Physical block # : 7 → 3 → 6 → 4



HDD

Advantage:-

① External fragmentation is removed.

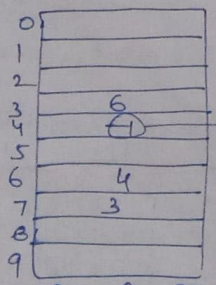
Disadvantage:-

① Random access is slower.
② holding the pointer to next block is an overhead.

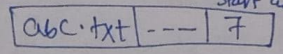
File allocation table (FAT):-

FAT is used to overcome the disadvantage of Linked list allocation to access randomly.

Eg:-



→ ends.



Disadvantage:-

FAT will also have some size & it can be very large.

Eg:- Block size = 1kB
HDD size = 1GB

no's of entries in FAT = $\frac{1GB}{1kB} = \frac{10^9}{10^3} = 10^6$ entries

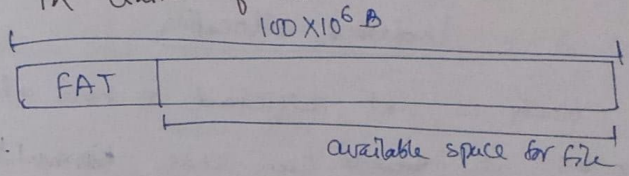
If each entry = 4B.

Size of FAT = $10^6 \times 4$
= 4MB.

GATE-2014

Que:- A FAT based file system is being used & the total overhead of each entry in the FAT is 4B in size. Given a 100×10^6 B disk on which the file system is stored & data block size is 10^3 B, the max size of a file that can be stored on this disk in units of 10^6 B is 99.6

Sol:



FAT size = no's of entries × 4B
= $(100 \times 10^6) / 10^3 \times 4B$
= 0.4×10^6 B

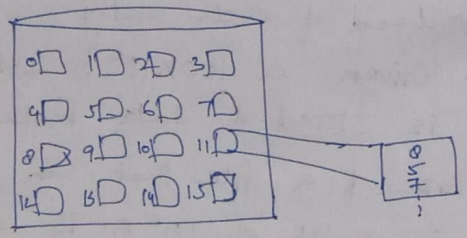
Max size of a file = $100 \times 10^6 - 0.4 \times 10^6$
= 99.6×10^6 B.
⇒ 99.6 Ans

③ Indexed allocation:-

→ In linked allocation direct access was the problem coz it was solved by FAT. But FAT size could be very large.

→ Indexed allocation solves the problem by bringing all the pointers together into one location: the Index block. ~~(Inode)~~

filename	index node
abc	11

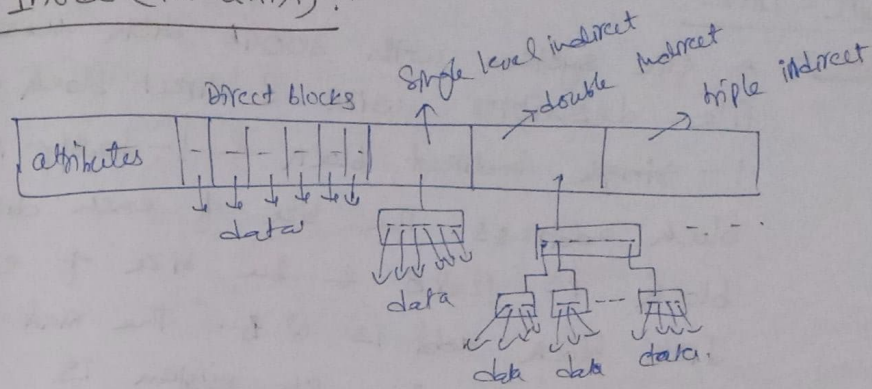


Indexed allocation

→ If 1-block is not sufficient to hold all indexes of blocks. we ~~can~~ can use ~~(in)~~ multiple index node.

→ Multiple index nodes can be linked together using linked list or can ~~be~~ their indexes can be stored again in ~~some~~ more index block called outer index block. (Just like multi-level paging).

Inode (In Unix):-



Gate-2004

Que:- A Unix style I-node has 10 direct pointers & 1-single, 1-double & 1-triple indirect pointers. Disk block size is 1kB, disk block addr is 32-bit & 48 bit integers are used. What is the max possible file size? a) 2^{24} B b) 2^{32} B c) 2^{34} B d) 2^{48} B

Soln

1-block size = 1kB

⇒ no's of entries of pointers in a block

$= \frac{1024\,B}{32\text{ bits}}$ ⇒ size of one pointer

$= 2^{10}/2^{2} = 2^{8} = 256.$

10 direct add = $10 \times 1kB$

1 single pointer = $256 \times 1kB$

1 double " = $256 \times 256 \times 1kB$.

1 triple " = $256 \times 256 \times 256 \times 1kB$

Gate - 2012

Que:- A file system with 300GB disk uses a file descriptor with 8 direct block addr, 1 - single indirect block & 1 - doubly indirect block address. The size of each disk block is 128B & the size of each disk block addl is 8 B. The max possible file size in this file system is

a) 3kB b) 35kB c) 280kB d) dependent on the size of the Disk.

Sol

Size of disk block = 128B

Size of each block addr = 8B.

No's of entries in a block

$= \frac{128}{8} = 16$ entries.

max size of a file

$= (8 \times 2^7 + 16 \times 2^7 + 16 \times 16 \times 2^7)B$

$= 2^7(8 + 16 + 256)B$

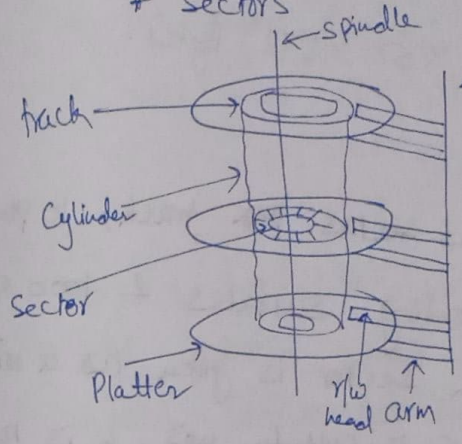
$= 2^7 \times 2^3(1 + 2 + 32)B$

$= 35kB$ Ans

Disk layout :-

→ Disk consist of
* platters.
* Tracks
* sectors

R/W in disk is done by the help of head.



* if we write in surface 0 Cylinder 1 then seek latency will be counted but writing in other surface on cy. 1 no seek latency because arm movement is relative not independent.

→ To read or write from a disk head has to come to appropriate position of correct sector in correct track of correct platter.

→ It includes some delays.

Seek latency :- The time taken by head to move to the right track.

Rotational latency :- The time taken to come right sector under head. avg rotational latency = 1/2 x rotational latency

Data transfer latency :- Time taken to transfer the data in or out the disk. It majorly depends on rotational latency.

Density :- No's of bits stored in some area:
eg: 30 bits / cm.

Q There are 10 platters in a disk, each platter have 20 tracks, each track have 50 sectors. Each sector can store 512 Bytes. Find the size of the disk.

Sol^n $(10 \times 20 \times 50 \times 512)$ Bytes.

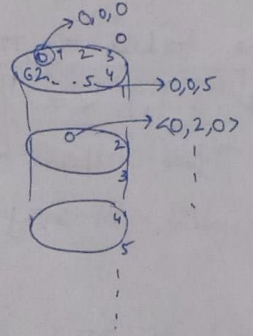
Gate-2009

Que A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces & 1000 cylinders. The address of each sector is given as a triple $\langle c, h, s \rangle$, where c is cylinder no's, h is the surface no's & s is the sector no's. Thus the 0th sector is addressed as $\langle 0,0,0 \rangle$, the 1st sector as $\langle 0,0,1 \rangle$ & so on.

The addr. $\langle 400, 16, 29 \rangle$ corresponds to which sector no's.

a) 505035 b) 505036 c) 505037 d) 505038.

Sol^n



$\langle 400, 16, 29 \rangle$

↳ that means 0, 1, 2, ... 399

⇒ 400 cylinders are already consumed.

⇒ Capacity of each cylinder

$= 10 \times 2 \times 63 = 1260$ sectors

⇒ Already consumed cylinders 400

$= 400 \times 1260$

on 400th cylinder we are 16th surface that mean 0, 1, 2, ---15 (16) surfaces are already over.

Capacity each surface = 63 sectors

400 16 29 $= 63 \times 16 = 1008$ sectors.

on 16th surface we are at 29th sector.

⇒ total sector no's $= 504000 + 1008 + 29$

$= 505037$ Ans.

Gate-1993

Que No's of tracks/recording surface = 200

Disk rotation speed = 2400 rpm.

Track storage capacity = 62,500 bits

The avg rotational latency of this device P msec.

& data transfer rate is Q bits/sec.

Write the value of P & Q.

Sol^n

Rotation speed = 2400 rpm

1 rotation time $= \frac{1}{2400} \times 60 \times 10^3$ msec

$= \frac{25}{\cancel{60000}\ \cancel{2400}} = 25$ msec.

Avg rotational latency $= \frac{1}{2} \times 25$ msec $= 12.5$ msec.

Data transfer rate is sec = no's of rotation in a sec * bits read in 1 rotation.

$= \frac{40}{\cancel{2400}\ \cancel{60}} \times 62,500$

$= 62,500 \times 40$ bits/sec.

Que

A Hard disk has the following parameters:

Num of tracks = 500

Num of sectors/track = 100

Nos of bytes/sector = 500

Time taken by the head to move from one track to adjacent track = 1ms.

Rotation speed = 600 rpm.

What is the avg time taken for transferring 250 Bytes from the disk?

a) 300.5ms b) 255.5ms c) 255ms d) 300ms.

Sol^n Total Time = avg seek time + avg Rotational latency + Data transfer time

$= \frac{500 \times 1ms}{2} + \frac{60 \times 1000}{600 \times 2} + 0.5 msec$

$= 300.5 msec$

avg Rotational latency $= \frac{1}{2}\left(\frac{1}{600} \times 60 \times 1000 \text{ msec}\right)$

Data transfer in 1 rotation = 500 B × 100 sectors.

= 50000 Bytes in 100 msec.

250 B in $\frac{250 \times 100}{50000}$ msec

$= \frac{0.5}{2}$ msec

$= 0.5$ msec

Disk scheduling:-

Accessing the sectors from the different-2 tracks needs proper scheduling.

① FCFS

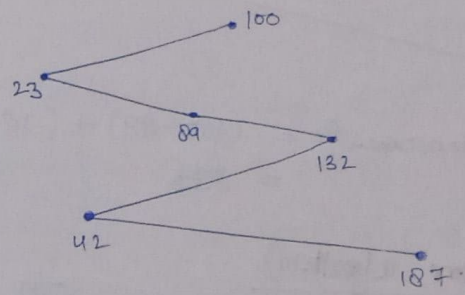
→ Simplest of all, easy to implement.

→ Perform operations in order of arrival.

→ No starvation: every request is served.

Ex A disk Queue with requests for I/O to blocks on cylinders. 23, 89, 132, 42, 187. with disk head initially at 100.

Sol^n



total movement = (100-23) + (132-23) + (132-42) + (187-42)

= 77 + 109 + 90 + 145

= 421

SSTF scheduling

→ Select the disk I/O request that requires the least movement of the disk arm from its current position, regardless of direction.

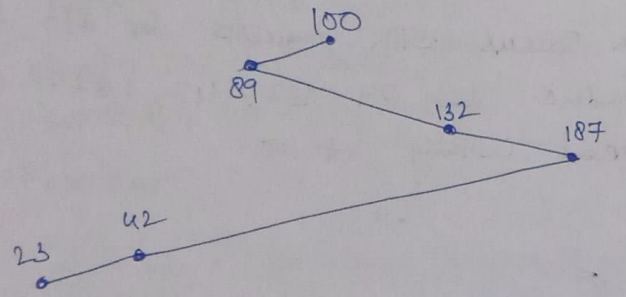
→ Reduce total seek time compared to FCFS.

DisAdvantage:-

→ Starvation is possible.
→ Switching directions slow down the process.
→ Not the most optimal.

eg:- 23, 89, 132, 42, 187.
currently at 100.

Soln



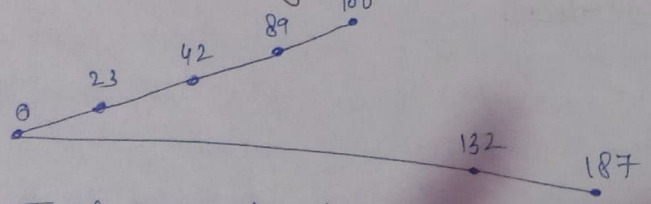
Total movement = (100-89) + (187-89) + (187-23)
= 273.

SCAN (Elevator algorithm)

→ move from ~~one~~ ~~side~~ current position to the ~~end of that side~~ 0th cylinder & back to the ~~other~~ ~~end~~ last ~~request~~ to read. cylinder

eg:- 23, 89, 132, 42, 187.
currently at 100 & moving towards 0.

Soln



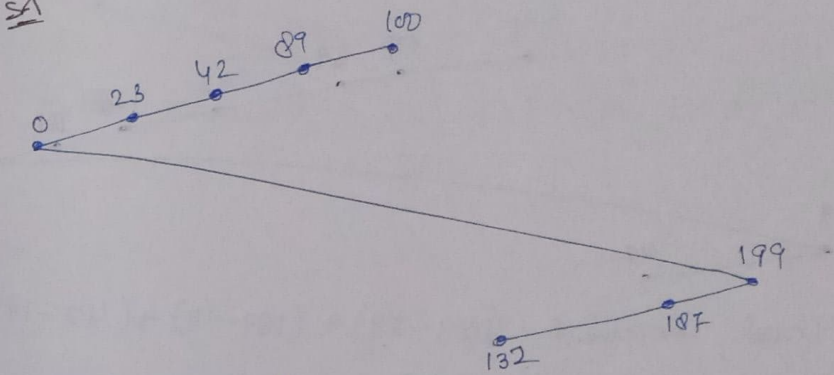
Total movement = 100 + 187 = 287

C-SCAN :-

→ move from current position to the 0th cylinder & jumps to the outermost cylinder without servicing any request.

eg:- 23, 89, 132, 42, 187
current position = 100
& there are 200 cylinders.

Soln

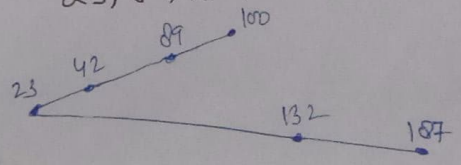


Total movement = (100-0) + (199-0) + (199-132)
= 100 + 199 + 67
= 366.

Look:-

→ Just like SCAN but we do not go till end instead revert the direction at the last cylinder request of that side.

eg:- 23, 89, 132, 42, 187.



Total movement
= (100-23) + (187-23)
= 241

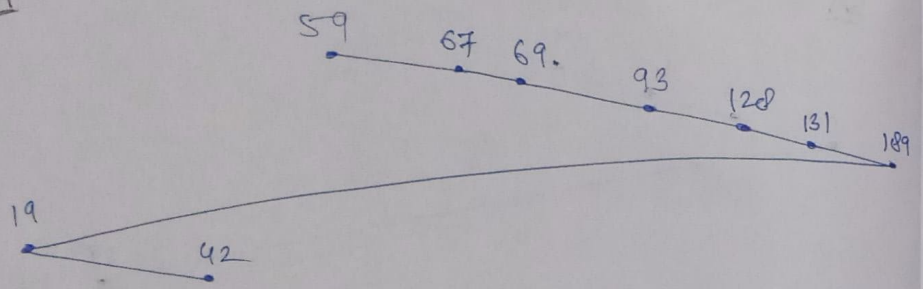
C-look:-

→ Just like C-SCAN, But we do not go till end but the last requested cylinder.

eg:- 93, 189, 42, 131, 19, 120, 67, 69.

head is at 59.

Sol^n



Total movement = (189-59) + (189-19) + (42-19)

= 323.

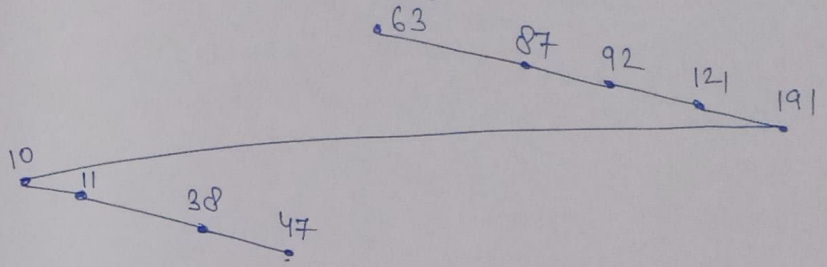
Gate-2016

Que

Sol^n

Req = 47, 38, 121, 191, 87, 11, 92, 10.

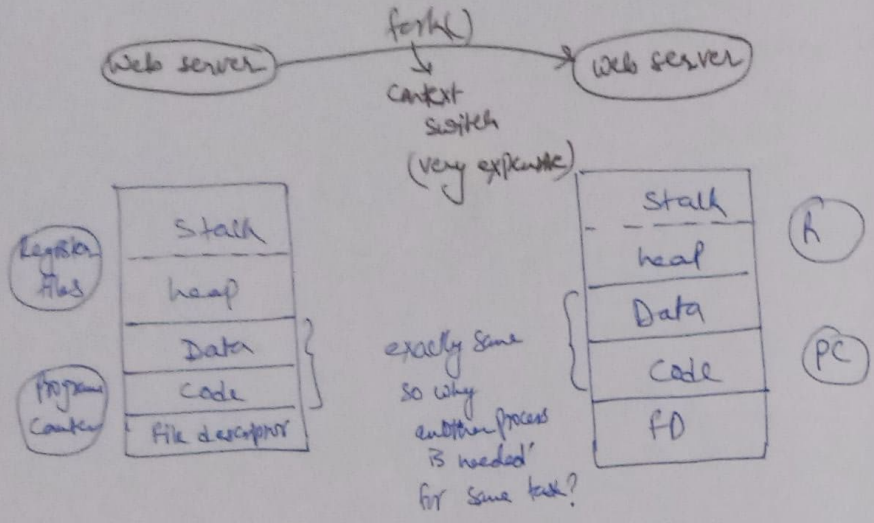
initial position = 63, moving towards 199.



total movement = (191-63) + (191-10) + (47-10)

= 346

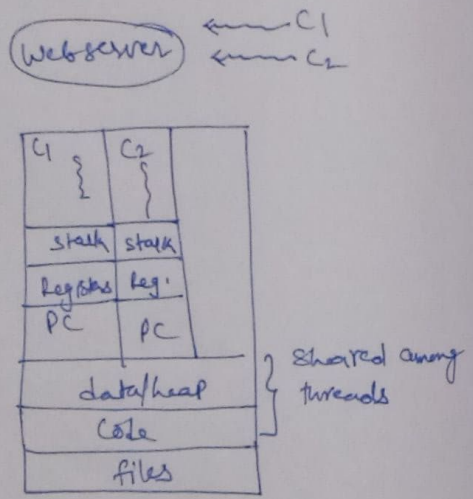
Process vs thread



Threads

User level thread

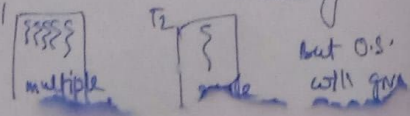
* User create the new thread. (O.S. is not involved).
* Switching b/w threads also O.S. is not involved.



Process (fork)	Threads (user level)
→ system calls are involved	→ NO system call
→ context switching is req.	→ NO context switching simply reg. set switch.
→ Diff copies of code & data	→ same copies of code & data.
→ C.S. & locks are req. for sync.	

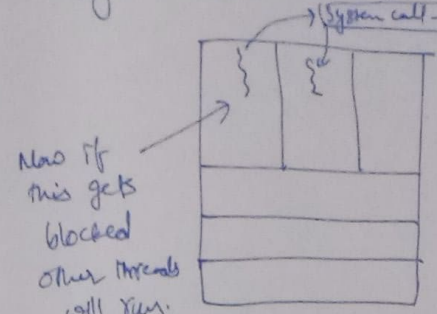
Disadvantages

* Even if 1 thread is blocked by O.S. then whole process (all threads) are blocked.
* Unfair Scheduling.



Kernel level threads :-

→ Now to create a new thread we have to make a system call to make kernel aware of diff. threads.



→ Appropriate scheduling will be done now.

Disadvantage

→ Expensive as compare to user level thread for creation.

→ Switching a thread req. a systemcall (context switch).

Hybrid thread :-