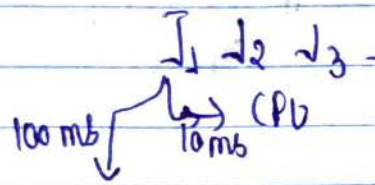
Operating System

→ Interface b/w user and h/w.
→ also called resource allocator.
↳ CPU, Ram, I/O devices (Keyboard, mouse, HDD)
→ act as a manager → m/m, process, files, security etc.

Types:-

1) Batch OS:- whoever gives the job first should finish it, then only other jobs will be taken for processing.



I/O total 110ms after this only processing will be given to J2.

→ J2 cannot process until J1 is finished (CPU + I/O).
→ Efficiency is very low due to starvation (waiting time). Urlolong.
→ Non-interactive OS.

2) Multiprogramming OS:- → Extended Batch OS.
→ while J1 is in I/O, CPU is idle and now we can allow J2 to execute on CPU.
→ Efficiency is ↑ses. → CPU is busy most of the time.

3) Multitasking or Time-sharing OS:-

→ Preemtive Job: Once job is entered in CPU we do not give complete time which job req. however after some time we give processing power again to same job to get complete. (Snatching resource (CPU) before complete execution.).

eg → J_1 ↓ 10ms, J_2 ↓ 15ms

J_1 completed in 4

J_1	J_2	J_1	J_1
0	5	10	15	25

→ J_2 completed.

→ Non-pre-emptive job → first job will be processed completly then only second comes. No division of time.

eg →

J_1	J_2
0	10	25

→ Multi-program. OS with pre-emption facility is multi-tasking OS.

→ Interactive OS as one job can undergo C.PU → I/O → CPU → other process etc

→ CPU choose all jobs for some time without completing one or the r. In this way interactiveness can be achieved.

4) Multiprocessing OS → → Instead of one CPU we have multiple.
→ Jobs can be executed on all C.PU simultaneously. J_1 J_2 J_3 J_4
(C1) (C2) (C3) (C4)

→ Efficiency is increased largely.
→ Throughput is improved → no of processes executed unit time.

5) Real time OS: → Jobs will be given with deadline.
$J_1 \rightarrow 7$

→ Jobs must be finished within deadline.

★ CPU bound Process → when process need more CPU time.
★ I/O " " → " " " I/O time.

★ Program → is a code. eg. main.C

★ Process → when OS takes this program & load into m/m (RAM) for execution. in some data structure which is called process.

PCB → Process control Block. or context.

Prg id
m/m
Reg.
files

O.S → process

Stack ↓
Heap ↑
Static & Global var.
Executable code

1000 ← | 12F ←

★ Attributes of Process :- Components.

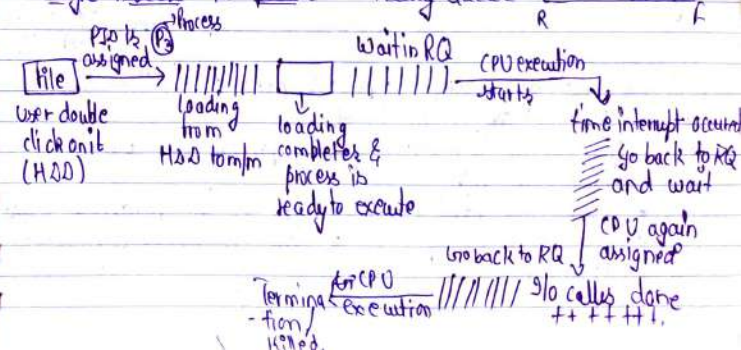
① → Process ID :- unique identification no's.
② → Prog. Counter → contains the address of next instr.
③ → Process state ④ → Priority ⑤ → General purpose Register.
⑥ → List of open file → means file it will be needing while executing a prog.
⑦ → List of devices. ⑧. Protection.

→ PCB's are linked from one other using Linked list

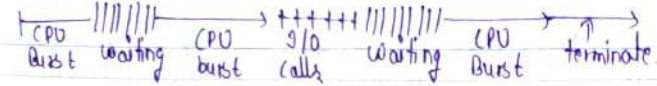
***

★ Single Process Perspective :- Ready Queue

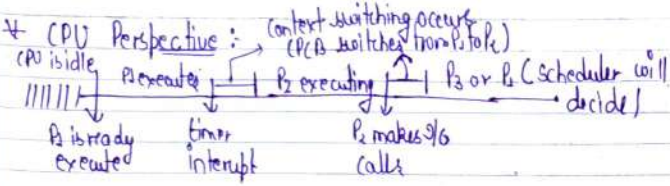
P_3	P_2	P_1
R		F



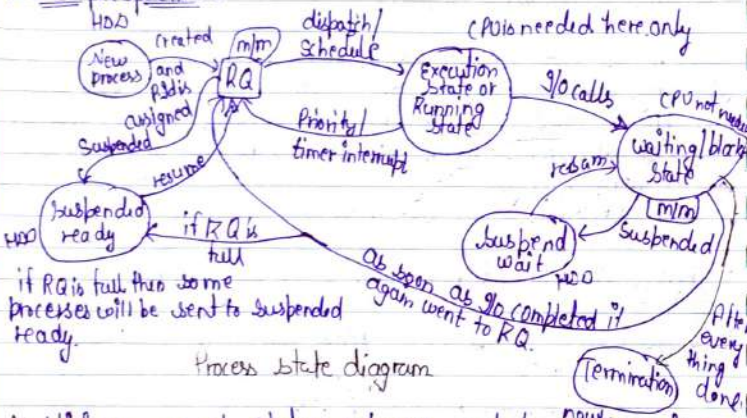
∴ Process Life cycle: is



* CPU Perspective :



* OS perspective :—



Process state diagram

* Minimum no. of states a process needs to travel for its completion is 4.

→ New :- Process is created, given process Id but is not loaded in m/m

→ Ready :
- Process is loaded into m/m & is ready to execute
- one or more process is can be in ready state at a tim.

→ Execution: Process is executing on CPU. &

Waiting/Block : The Process is added to Queue of specific I/O devices.

Termination :- The Process has finished its execution.

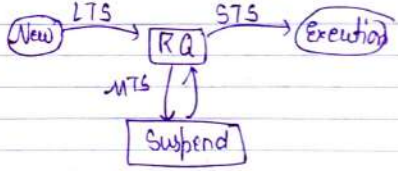
Suspended State :- The process is suspended out to HDD for some time, to release m/m (in RQ.)

* Types of Scheduler : Scheduler are decision maker for selecting a process.

1) Long Term Scheduler (LTS) :
- decision is imp → b/c it affects the system performance.
- from New to Ready state
- Works on which process to load amongst newly created process

2) Short Term Sch. →
- selection must be based on less context switch.
- works on which process to pick for execution from RQ

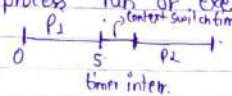
3) Medium Term Sch :
- Swapping must be less.
- process from ready to suspend & vice versa.



- Dispatcher :- It will execute the context switch decision made by the schedulers.

Context Switch :- PCB
↳ Reg., Open files, m/m used etc.

- Every process run or executes in its context.



- Before switching to a new process, context of old process is stored by OS.
- OS saves all req. info. of currently executing process & restores info of new process.
- Time taken to do context switch is called CST.
- CST is an overhead → b/c no useful work is in this time as it only storing & retriving process in this time.

Ques → CST = 1ms, Process Burst Time = 100ms, processes keep coming to keep CPU busy., effective CPU utilization = ?

Ans
Burst time + CST = 100 + 1 = 101 ms for 1 process total time

C.P.U Utiliz = $\frac{\text{total useful time}}{\text{total time}} \times 100 = \frac{100\text{ms}}{101} \times 100 = 99.09\%$

Ques → CST = 1/2 C.P.U Burst time. what is CPU utilization.
Ans Let CBT = 100ms
∴ CST = 50ms
∴ % = $\frac{100}{150} \times 100 = 66.66\%$

* System Calls :-
- It is like a func^n with parameter & a return value.
- System calls results in a sw interrupt.
- OS will check the privileges of the user before executing system calls requested by him/her
- The system calls results in 2 context switches.
- No context switch in func^n calls
- System calls are slower than func^n calls.

- System calls are the services provided by an OS in form of func^n.

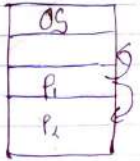
* Categories of System Calls :-
1. Process Related Sys calls eg → fork(), exit(), Kill().
2. I/O " " " eg → read(), write(), open(), close()
3. Disk " " " eg → format(), partition().
4. Information " " " eg → getpid(), setpid().

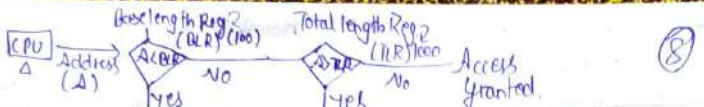
* System programs :- • Set of utilities (executable files) available which can be executed at command prompt.
- System prog. are written by developers and assist easy use of OS.
- System prog. internally uses sys. calls to access the OS code

* System level Library func^n :- Code written in HLL, it may or may not result in sys. calls.
eg. <stdio.h>, <Xyz.h> ↳ created by own.

* Hardware protection :-
1) CPU Protection :- • No process should hog over CPU infinitely.
• Timer is a hardware device in a process -or, and its value is set by OS to a specific value before a process is given to CPU.
- At interrupt, OS code decides which process to execute next.

2) M/m Protection :-
→ No illegal access to other process m/m should be allowed.
- If m/m protec. is not given then P_2 can access the imp/critical data of above process or P_2.



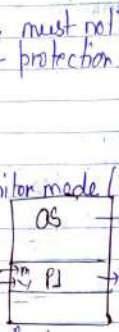


so if A = 90 → Seg. fault (A<100 (BLR))
if A = 1100 → No → (A>1000 (TLR)) → Seg. fault
if A = 800 → No → No → Access Granted.

3) I/O Protection : • Illegal access to I/O devices must not be allowed
• System calls are used for protection.

* DUAL MODE :

1) Supervisor mode or ~~Kernel~~ kernel mode or Monitor mode (MBR=0) (for OS).
2) User mode (MBR=1). act b/w processes or in process.



→ MBR (mode bit Reg) is available on processor.
→ Instructions are of two types.
a) Privileged inst. b) Non-Privileged inst.
→ Privileged inst can only run in kernel mode.
→ Any sys. call is a Privileged inst.

* Criteria of Process Scheduling :→ Process scheduling is picking up one process from Ready state/Queue & give it to CPU for its execution.

→ Parameters for evaluation :
a) CPU Utilization :- % of time CPU is busy doing useful work.

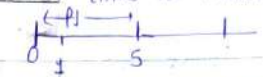
CPU % = $\frac{\text{useful time} \times 100}{\text{total time}}$

eg-

% = $\frac{14}{15} \times 100$.

b) Response time : time for which a process has to wait in RQ

eg-



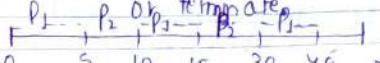
P_2 = 4ms Burst time
Arrival = 1

P_2 come but can't execute bc P_1 already running so

response time is = process start − arrival time = 5 − 1 = 4.

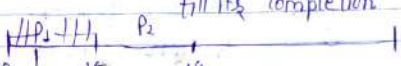
c) Completion time : the time at which process completes or terminate

eg



time for P_2 is → so completion = 40

d) Turn around time : total time of process from its arrival till its completion



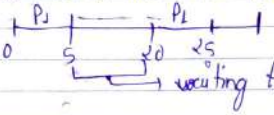
P_2 = 4ms Burst
Arrival = 1ms

= 19 − 1 = 18ms. ✓

Turn around time = Completion − arrival.

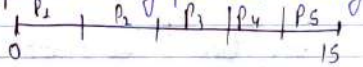
e) Waiting time : It is the total time a process waits in RQ.

eg



Waiting time = Turn around time − Burst time

f) Through put : No. of processes running per unit time.



Through put = $\frac{\text{No. of Process}}{\text{total time}}$ = $\frac{5}{15} = \frac{1}{3}$ = 0.33 process in 1 unit time

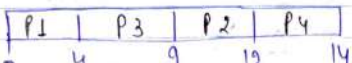
g) Premtive & Non Premtive mode : already discussed

* FCFS (First come first Serve) Algo :- Non-Primitive Sched-algo.
→ Criteria is arrival time.

eg:-

Process	Arrival	Burst	Start	Complete
P_1	0	4	0	4
P_2	2	3	9	12
P_3	1	5	4	9
P_4	4	2	12	14

Gantt's Chart



	Turn around time	waiting time	Response time	Avg turn around time	Avg wait time	Avg Response
P_1	4	0	0	4+10+8+4 / 4 = 8	0+7+3+8 / 4 = 4.5	4.5
P_2	10	7	7			
P_3	8	3	3			
P_4	10	8	8			

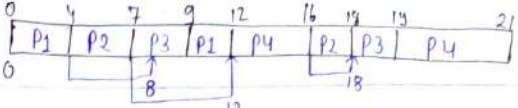
→ CPU % = $\frac{14}{14} \times 100 = 100\%$. No time CPU is empty.

→ Through put = $\frac{4}{14}$.

Ques → FCFS Example, where we have multiple I/O devices

Process	Burst time	Arrival time	I/O Burst time	Total Burst Time
P_1	4, 3	0	4	11
P_2	3, 2	1	5	10
P_3	2, 1	4	3	6
P_4	4, 2	10	2	8

Gantt's Chart



	Start time	Completion time	Comple - Arrival Turn around time	TAT - Burst Waiting time	start - Arrival Response time
P1	0	12	12	1	0
P2	4	18	17	7	3
P3	7	19	15	9	3
P4	12	21	11	3	2

Avg TAT = 12+17+15+11 / 4 = 55/4

Avg W.T = 1+7+9+3 / 4 = 20/5 = 5

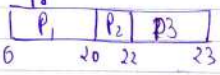
Avg R.T = 0+3+3+2 / 4 = 8/4 = 2

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

Through put = $\frac{4}{21}$

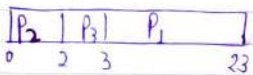
** Convoy effect is the disadvantage of FCFS algo.
↳ when waiting time of other process increases due to more burst time of earlier process.

Process	Arrival	Burst
P_1	0	20
P_2	1	2
P_3	1	1



AWT = 0+19+21 / 3 = 40/3 (more)

Process	Arrival	Burst
P_1	1	20
P_2	0	2
P_3	0	1



AW.T = 0+2+2 / 3 = 4/3. (less)

* SJFS (Shortest Job first Scheduling):-

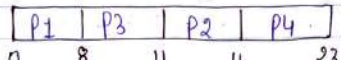
Criteria: Shortest Burst time first.

Mode: Non pre-emtive & pre-emtive (Shortest Remaining job first or SRTF).

Note: Tie of multiple shortest job is broken with FCFS criteria

eg → 1) Mode is Non-pre-emtive.

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



calculate rest thing and make table

Avg TAT = 52/4
Avg WT = 29/4
Avg RT = 29/4
CPU % = 100%
Throughput = 4/23

→ Imp. Notes:
- SJFS gives best throughput but suffers from starvation.
- FCFS doesn't have starvation problem. (Assuming no process is running for infinite time)
- It is very difficult to predict the CPU burst time before actually executing a process.

* 8

* Burst Time Prediction:

Prediction Techniques

Static

1) Process size
If P_1 → 100kb → 10ms
then predicting that
P_2 → 102kb → 10ms

2) Process type:-
a) OS type:- (4-5) unit time
b) User type
→ Interactive: (5-8) Unit
→ Background: (15-20) Unit

Dynamic

1) Simple averaging
If time for P_1 P_2 P_3 P_4 is → 100ms so 100/4 will be time given to P_5 → x_5
$T_{n+1} = \sum_{i=1}^{n} t_i/n$

2) Exponential average method

$$T_{n+1} = \alpha t_n + (1-\alpha) T_n$$

T_n → Estimate of CPU burst for n^{th} Process
t_n → Actual CPU burst for n^{th} Process
α → Smoothing Parameter (0 to 1)

eg → $\alpha = 0.6$, $T_1 = 10ms$, actual Burst time of P_1 P_2 P_3 are 5, 9, 5. (t_1, t_2, t_3) respectively. $T_4 = ?$

Ans

$$T_4 = \alpha t_3 + (1-\alpha) T_3$$

$$\therefore T_2 = \alpha t_1 + (1-\alpha) T_1 = 0.6 \times 5 + (0.4) \times 10 = 7.0$$
$$T_3 = \alpha t_2 + (1-\alpha) T_2 = 0.6 \times 9 + (0.4) \times 7 = 6.4$$

$$\therefore T_4 = 0.6 \times 5 + (0.4) \times 6.4 = 5.56$$ Ans

→ It is exponential b/c:- $\because T_{n+1} = \alpha t_n + (1-\alpha) T_n$

$n=0 \Rightarrow T_1 = \alpha t_0 + (1-\alpha) T_0$ — (i)
$n=1 \Rightarrow T_2 = \alpha t_1 + (1-\alpha) T_1$ — (ii)
Put eqn (i) in eqn (ii)

$T_2 = \alpha t_1 + (1-\alpha)[\alpha t_0 + (1-\alpha) T_0]$

$= \alpha t_1 + \alpha(1-\alpha) t_0 + (1-\alpha)^2 T_0$

$T_3 = \alpha t_2 + (1-\alpha) T_2$

$T_3 = \alpha t_2 + (1-\alpha)[\alpha t_1 + \alpha(1-\alpha) t_0 + (1-\alpha)^2 T_0]$

$= \alpha t_2 + \alpha(1-\alpha) t_1 + \alpha(1-\alpha)^2 t_0 + (1-\alpha)^3 T_0$ ----

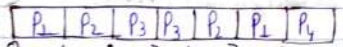
∴ it is increasing exponential thatis why exponential averaging

Shortest Remaining time first.

* Pre-emptive Shortest Job first Schedule (SJFS) or SRTF.

eg:-

Process	Burst	Arrival
P_1	6	0
P_2	3	1
P_3	4	2
P_4	6	3



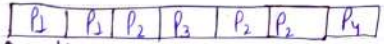
	from left	from right	CT − AT	TAT − BT	ST − AT
	Start	Completion	TAT	WT	RT
P_1	0	13	13	6	0
P_2	1	7	6	2	0
P_3	2	4	2	0	0
P_4	13	19	16	10	10
Average			37/4	18/4	10/4

CPU% = $\frac{19}{19} \times 100 = 100\%$

Throughput = 4/19.

eg2:- Total waiting time of P2 using SRTF (GATE-2007).

Process	Arrival	Burst
P_1	0	20 5
P_2	15	25 10
P_3	30	10
P_4	45	15

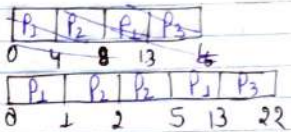


	Completion	TAT	WT
P_1	20	20	0
P_2	55	40	15
P_3	40	10	0
P_4	76	25	10

∴ P_2 waiting time is 15unit.

eg:- Average w.T using SRTF or Pre-emptive SJFS?

Process	Arrival	Burst
P_1	0	9 5
P_2	1	4 0
P_3	2	9



$w.T = \frac{(0-0)+(5-1)}{P_1} + \frac{0}{P_2} + \frac{(13-2)}{P_3} = \frac{4+0+11}{3} = 5ms$ (same)

or

	Completion	TAT	WT
P_1	13	13	4
P_2	5	4	0
P_3	22	20	11

$\frac{4+0+11}{3} = \frac{15}{3} = 5$ unit.

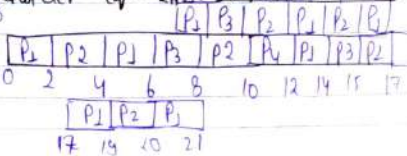
* ROUND ROBIN (RR) ☆☆

- Each process is given a time quanta t_q.
- If a process is completed before that t_q then we schedule next process immediately
- If a process need more time then we will put it at the end of RQ after t_q time is over.

eg:-

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

Consider t_q = 2ms. RQ: P1 P2 P3 P1 P4



	Start	Completion	TAT	WT	RT	
P_1	0	21	21	12	0	CPU% = 100%
P_2	2	20	19	12	1	
P_3	6	15	12	9	3	Throughput = 4/21
P_4	10	12	7	5	5	

Eg → time quanta t_q = 3ms.

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

RQ → [P_6|P_4|P_5|P_3|P_2|P_1|P_5|P_3|P_2|P_1|P_5|P_2]

ideal

[P_6|P_4|P_5|P_3|P_2|P_1|P_5|P_3|P_2]

0 1 4 6 9 12 15 18 21 23 26

[P_1|P_5|P_1]

26 29 32 33

CPU% = $\frac{33}{33} \times 100$. < 100%. throughput = $\frac{6}{33}$ ✓

Eg → Consider 4 Jobs P_1, P_2, P_3 & P_4 arriving in RQ at same time = 0. If B.T req of these are 4, 1, 8, 1 respectively. What is the completion time of P_3, assume RR with t_q = 1ms

Ans

Process	Arrival	Burst
P_1	0	4
P_2	0	1
P_3	0	8
P_4	0	1

RQ → [P_1|P_2|P_3|P_4|P_1|P_3|P_1|P_3|P_1|P_3]

Gantt → [P_1|P_2|P_3|P_4|P_1|P_3|P_1|P_3|P_1|P_3|P_3|P_3]

0 1 2 3 4 5 6 7 8 9 10 11

[P_3|P_3|P_3]

11 12 13 14

Ans = 14

Ques: Consider n processes sharing CPU in a RR fashion. Assuming that each process switch takes s sec. 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 sec.

Ans

Let n = 4. P_1 | P_2 | P_3 | P_4 | P_1

t_q t_q t_q

t sec period

for 1 process ↑ context switch = s sec.

" n " " = n×s

∴ we cannot add more than t_q time in time quanta.

∴ for n process it will be (n-1)t_q.

So total: $n \times s + (n-1)t_q \leq t \Rightarrow t_q \leq \frac{t - ns}{n-1}$

* Important points about RR: Most fair algo.

1) RR is pre-emptive algo & doesn't suffer from starvation.
2) gives a bound on response time for a process.
3) makes it useful for interactive system. (Unix, Linux, Windows)
4) If t_q increases then context switch time decreases & Response time ↑
5) " " decreases " " " " ↑ses & " " ↓ses
6) If t_q = very large it becomes FCFS.

(Let)

[loooday | P_2 | P_3]

P_1

7) If t_q = very small (0.001ms Let) → CPU% is very very less.

0 ↓ ↓

many context switches

8) If t_{cs} = t_{cs} (Context switch time)

then CPU% = $\frac{t_q}{t_q + t_{cs}}$ = $\frac{t_q}{t_q + t_q}$ = 0.5 x 100 = 50% utilization

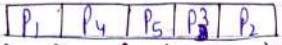
9) So, Generally t_q is kept in such a way that 80% of processes finishes in 1 time quanta only. → (LRTF)

* Longest Job first scheduling (LJFS):- both premtive & non pre-emtive.

Eg:-

Process	Arrival	Burst
P_1	0	3
P_2	1	9
P_3	2	4
P_4	3	6
P_5	4	5

for non-preemtive ↓



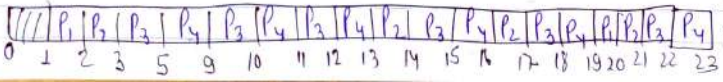
	Start	Completion	TAT	WT	Response
P_1	0	3	3	0	0
P_2	18	20	19	17	17
P_3	14	18	16	12	12
P_4	3	9	6	0	0
P_5	9	14	10	5	5

CPU% = $\frac{20}{20} \times 100 = 100\%$

through put = $\frac{5}{20} = \frac{1}{4}$

* Longest Remaining time first or Premtive (LTFS)

Process	Arrival	Burst
P_1	1	2-1
P_2	2	4 3
P_3	3	7 6
P_4	5	9 8



	Start	Completion	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% = $\frac{22}{23} \times 100$

through put = $\frac{4}{23}$

Gate (2006):- Consider three processes (process id 0, 1, 2 respectively) with compute time burst 2, 4 and 8 time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with the lowest process id. The average turn around time is:

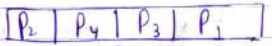
Process	Arrival	Burst
P_0	0	2
P_1	0	4
P_2	0	8

Completion	TAT
12	12
13	13
14	14

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

* Priority Scheduling: mode → Non-Preemtive

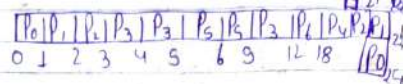
Process	Arrival	Burst	Priority
P_1	3	6	3 (lowest)
P_2	0	4	3
P_3	2	2	2
P_4	3	5	4 (Highest)



Mode :- Pre-emptive :-

(20)

Process	Arrival	Burst	Priority
P_0	0	4 3	2 (low)
P_1	1	2 1	4
P_2	2	3 2	6
P_3	3	5	10
P_4	4	1	8
P_5	5	4	12 (High)
P_6	6	6	9



Process	Start	Complete	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%

Thro put :- 7/25

* Highest Response Ratio Next (HRRN) :-

$$\text{Response Ratio} = \frac{w + s}{s}$$

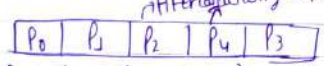
where w = waiting of a process at time t.

s = Burst time of that process

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

Process	Arrival	Burst
P_0	0	4
P_1	3	8
P_2	6	6
P_3	9	7
P_4	12	3

Assume - Non-Pre-emptive (After calculating Response ratio)



(21)

$RR_2 = \frac{(12-6)+6}{6} = \frac{6+6}{6} = 2.0$ ✓ $\quad RR_4 = \frac{0+3}{3} = 1$

$RR_3 = \frac{3+7}{7} = 1.43$

$RR_3 = \frac{9+7}{7} = 2.29$ $\quad RR_4 = \frac{6+3}{03} = \frac{9}{3} = 3.0$ ✓

	Start	Complete	TAT	WT	RT
P_0	0	4	4	0	0
P_1	4	12	9	1	1
P_2	12	18	12	6	6
P_3	21	28	19	12	12
P_4	18	21	9	6	6

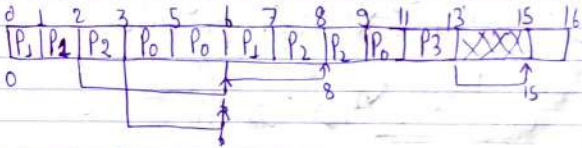
CPU % = $\frac{28}{28} \times 100 = 100\%$

Through put = 5/28

Qun (Infosys - 2019) Using SRTF, Calculate Average TAT.

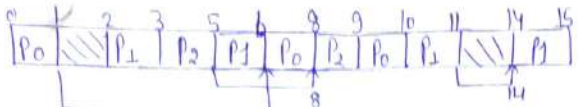
Process	Arrival	Burst	% Burst.	Completion	TAT
P_0	0	3,2	2	11	11
P_1	0	2,1	4	7	7
P_2	2	1,2	3	9	7
P_3	5	2,1	2	16	11

Avg TAT = $\frac{36}{4} = 9.0$ units



Qun - Using Priority scheduling, Mode : Pre-emptive (parameters) calculate others

Process	Arrival	Burst	% Burst	Priority	Start	Completion	TAT	WT	RT
P_0	0	1,3	5	2	0	10	10	1	0
P_1	2	3,1	3	3 (low)	2	15	13	6	0
P_2	3	2,1	3	1 (High	3	9	6	0	0



CPU % = $\frac{(15-4)}{15} \times 11 = \frac{11}{15} \times 100$.

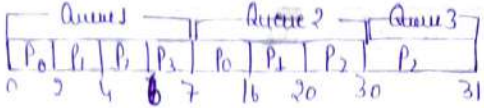
* Multilevel Feedback Queue : Used Now a- days.

Q1 → t_{aj} = 2 ms
Q2 → t_{aj} = 10 ms
Q3 → FCFS

Q1 > Q2 > Q3

Out-thing

Process	Arrival	Burst	Given ↓
P_0	0	11	$Q_1 \to 2$ ms → t_{aj}
P_1	1	6	$Q_2 \to t_{aj} \to$ 10 ms
P_2	2	13	$Q_3 \to$ FCFS.
P_3	3	1	



UNIT → 2 Completed

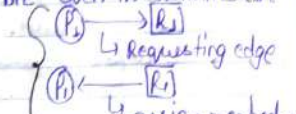
Till → Mid-Sem

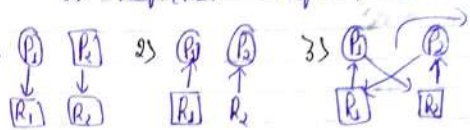
UNIT → 3 till END-SEM

* Introduction to Deadlocks : No further movement is possible even in infinite time

i.e deadlock situation ⇒

	P_1	P_2
1>	request(R_1)	request(R_2)
2>	Use R_1	Use(R_2)
3>	Req.(R_2)	Req(R_1)

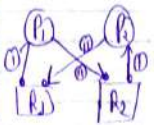




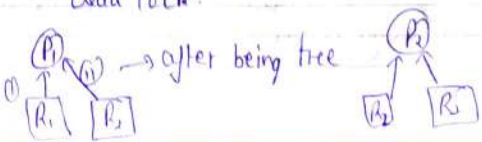
∵ R_1 is assigned to P_1 and R_2 is assign to P_2 so now in '3' P_1 req. for R_2 which is not available - same with P_2 so there an infinite loop occure thus situation is deadlock.

Starvation: Movement is not possible for long time & for few processes movement is still going on.

- Now suppose if R_1 & R_2 have two instances then it can be shared and no deadlock even in the cycle.



- Now if no context switch occurs means no movement from P_1 to P_2 i.e first every thing of P_1 gets completed then also no deadlock.



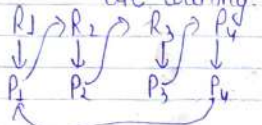
Essential conditions for deadlock:

1) Mutual Exclusion: Resources are not sharable.
2) Hold & wait :- One process is holding a resource and waiting for another.

→ Each process in deadlock cycle holds a resource and waits for another resource.

3) No Pre-emption: Resources can't be snatched from another processes.
4) Circular wait :- There exist a cycle in which processes are waiting.

eg →



→ ★★★ So these 4 conditions are like 4 legs of chair if any of one is broken then there will be no deadlock.

* Removal of Deadlock:

1) Deadlock Prevention
2) " Avoidance
3) " Detection & Recovery
4) " ignorance.

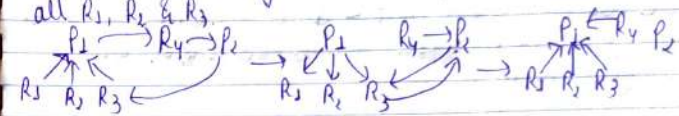
1) Deadlock Prevention:- One of the 4 essential condition is violated.

a) No mutual Exclusion:- • Make resources sharable
problem - few resources are inherently non-sharable. eg- Printer.

b) No hold & wait :- I → All requirement of process is given during start. eg → P_1 wants $R_1, R_2 \& R_3$ (29)

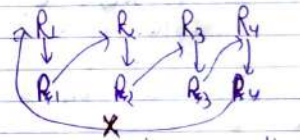
II → if available assign all 3 if not then ~~assign~~ do not run P_1 otherwise it req. for resource which are not available and create deadlock.

II → Release all ~~process~~ resources before a process request for a new resource. eg → P_1 executing and in future request for R_4 (not available) and P_2 req. for R_3 (not available → cycle). so to request R_4, P_1 has to release all R_1, R_2 & R_3



c) ~~pre~~-emption: • Higher priority process can pre-empt (snatch) resources from lower priority process
• lower priority process release resources when it can't acquire resources held by high priority process.

d) No-circular wait :-



processes can req. resources only in ascending order.

---

* Deadlock Avoidance :→ Max need of each process is known when process starts.
→ Before granting a request, it is checked if the resulting system will be safe.
→ if Yes, then req. is granted otherwise denied.
→ Bankers Algo is used.

Customer	(lakh) Max Need	(lakhs) Allocated	future need
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.

available amount = total amt − allocated amt. = 19 − 15 = 4.
we cannot allocate these 4 lakh to C_4 & C_4 b/c they are in higher need.

C_2 → available amt = 4 lakh
released " = 3
total available = 7 lakh.

Now → C_1 → Avai amt = 7
released " 6
total avai = 13.

C_3 → Avai = 13
release = 2
total avai = 15

C_4 → Avai = 15
released = 4
Avai = 19.

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

Qw → Now there is an urgent req. of 1 lakh by C_1 will request be granted?

Ans

Custom.	max	Allocated	future need
C_1	12	6	5
C_2	7	3	4
C_3	6	2	4
C_4	9	4	5
		15	

total amt in Bank = 19 lakh
available amt. = 19 − 16 = 3
Not granted b/c none of the above requirement meet as we only have 3 lakhs.

Qw → Now if req. from C_2 comes.

Customer	Max	Allocated	future
C_1	12	6	6
C_2	7	3 4	3
C_3	6	2	4
C_4	9	4	5

total amt = 19 lakh
avai = 19 − 16 = 3 lakh

C_2 → total available = 3 + 4 = 7 → everyone's need can be fulfilled.
$\langle C_2, C_1, C_3, C_4 \rangle$ → safe sequence.
Granted request to C_2.

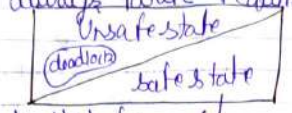
* Safe Sequence: Hypothetical order of execution of processes, such that process P_i's need are satisfied & on completion, it will be assumed that P_i will release its allocated resources which can be used by another processes.

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

Note:- Unsafe state does not necessarily means deadlock.

eg:- necessary water is " for human life. | sufficient water is sufficient + food + shelter ✗

→ As max need doesn't reflect always future requirement.
→ Safe state is sufficient for a system to be deadlock free.



Ques - Process	Max	Allocated	future(need)
P_1	7	2	5
P_2	6	4	2
P_3	3	0	3
P_4	4	0	4

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

P_2 → avai = 3, release = 4, total avai = 7
P_1 → available → 7, release → 2, total avai → 9
P_3 → avai = 9, release = 0, (avai) total = 9
P_4 → available = 9, release = 0, (avai) total = 9

safe seq. $\langle P_2, P_1, P_3, P_4 \rangle$.

b) Req of 1 resource to P_2 & 1 to P_3. is the system in safe state?

Process	Max	Allocated	future
P_1	7	2 3	4
P_2	6	4	2
P_3	3	0 1	2
P_4	4	0	4

total resources = 9
available = 9 - 5 = 1 → none of the req. can be fulfilled, so this request is denied.

eg→ Process	Maxneed			Allocated			future needs		
	R_1	R_2	R_3	R_1	R_2	R_3	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

total resources → (13, 11, 8)
" available → (13, 11, 8) - (10, 8, 6) = (3, 3, 2)
only P_1's req. can be fulfilled in first case of ".

P_1 → Avai → 3, 3, 2; release → 4, 3, 1; total avai → 7, 6, 3
P_2 → Avai → 7, 6, 3; release → 2, 2, 2; total avai → 9, 8, 5
P_3 → Avai → 9, 8, 5; release → 2, 2, 0; total avai → 11, 10, 5
P_4 → Avai → 11, 10, 5; release → 2, 1, 3; total avai → 13, 11, 8

safe seq. → $\langle P_2, P_1, P_3, P_4 \rangle$. → safe state → deadlock free.

b) will BS grant req. of (0, 1, 1) from P_2?

Process	MaxNeed			Allocated			future need.		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	2	4	3	1	3	2	1
P_1	6	4	3	2	1	1	4	1	0
P_2	2	8	1	2	2	0	0	6	1
P_3	5	6	4	2	1	3	3	5	1

total resource = (13, 11, 8)

available = (13, 11, 8) − (10, 9, 7) = (3, 2, 1)

only P_1 can finish at first.

P_1 → avai → 3, 2, 1 — P_2 → 7, 5, 2 → avail
release → 4, 3, 1 — 2, 3, 3 → release
total → 7, 5, 2 — 9, 8, 5 → total

P_3 → Avai → 9, 8, 5 — P_4 → Avai → 11, 10, 5
release → 2, 2, 0 — release → 2, 1, 3
total → 11, 10, 5 — total → 13, 11, 6

seq. (safe) → $\langle P_1, P_2, P_3, P_4 \rangle$ → safe state

* Minimum Resource Requirement:

- If we have unlimited Resources or we increase the no. of resources drastically then the system will always be in safe state.
- Increasing resources can be costly
- If all the processes acquire / Allocated max − 1 resource then also system will be in deadlock. By increasing 1 more resource, system will be deadlock free.

Min resources req = $\sum$ maxneed − no. of processes + 1

	Max	Allocate	Req.
P_1	5	4	1
P_2	4	3	1
P_3	3	2	1
		9 +1	

5+4+3−3 +1
9+1

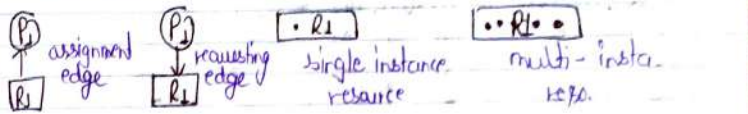
↳ Now fulfilled.

Ques: find minimum no. of resources so that system always remain free from deadlock.

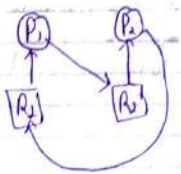
Process	Max Need		
	R_1	R_2	R_3
P_0	7	5	2
P_1	6	4	3
P_2	9	8	1
P_3	5	6	4

minimum resources = $\sum$ maxneed − no's of processes + 1
= (20, 23, 10) − (4, 4, 4) + (1, 1, 1)
= (16, 19, 6) + (1, 1, 1)
= (17, 20, 7)

* Resource allocation graph:



eg:



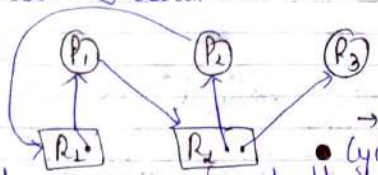
→ cycle is enough to show a dead lock in single-instance resource allocation graph.

→ but what if we are not able to show a/c to this. so we have an algorithm.

	Allocated Res.		Requesting Res.	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
	1,	1		

total res. = (1,1)
avai " = (1,1) − (1,1) = (0,0)
∴ P_1 req. an resource R_2 but we have (0,0) resources ⇒ Deadlock

eg2:



→ multi instance Resource
• Cycle is not sufficient but necessary for deadlock in multi-instance resource allo. graph.

	Allocated		Requesting	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0
	1	2	1	1

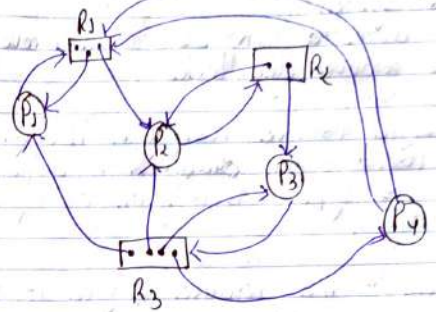
total res = (1, 2)
avai " = (1, 2) − (1, 2)
= (0,0)

R_1 → 1 instance
R_2 → 2 "
total = (1, 2)
avai = total − alloca.

same (0,0) but no dead lock why?
b/c P_3 → will complete & it will release (0,1)
P_1 → avai = 0,1
release = (1,0)
total ava = (1,1)
P_2 → avai = (1,1)
release = (0,1)
total = (1,2).

order of execution < P_3, P_1, P_2 > → safe state
⇒ Deadlock free.

eg3 → Deadlock or not



	Allocated			Requesting		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	1	0	1	1	0	0
P_2	1	1	1	0	1	0
P_3	0	1	1	0	0	1
P_4	0	0	1	2	0	0
	2,	2,	4	3,	1,	0

total res = (3, 2, 4)
avai = (3, 2, 4) − (2, 2, 4)
= (1, 0, 0).
→ P_1 req. can be fulfilled

P_3 → avai total = (1,0,0)
release = (1,0,1)
total = (2,0,1)

P_3 → avai total = (2, 0, 1)
release = (0, 1, 1)
total = (2, 1, 2)

P_2 → Avai = (2,1,2)
release = (1,1,1)
total = (3,2,3)

P_4 → Avai = (3,2,3)
release = (0,0,1)
total = (3,2,4) ✓
matched → no

mistake, order < P_1, P_3, P_2, P_4 > → safe
→ No deadlock

* 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 ~~receive~~ remove deadlock.

→ Detection :- • In single instance res. type, if a cycle is present then deadlock is detected.
- In multi-instance res. type, we will run safety algo (modified bankers algo).

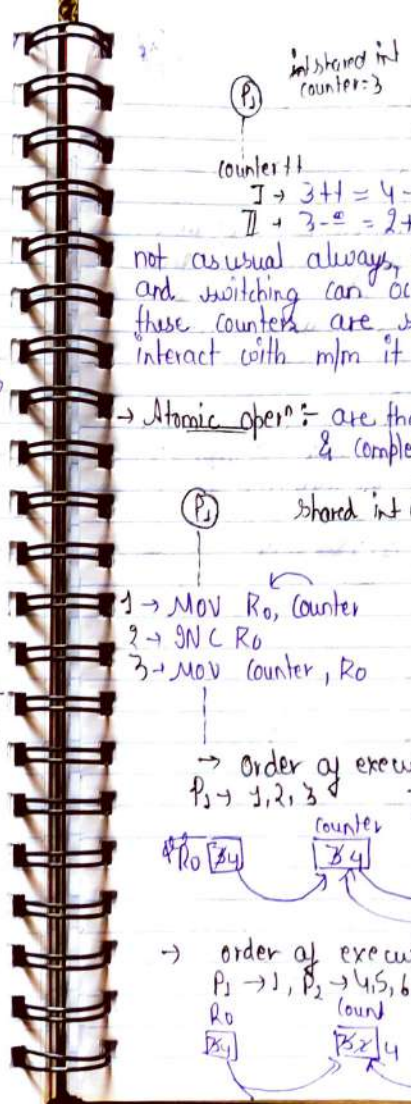
→ Recovery :- • Pre-empt the resource & give it to high priority process.
- Roll Back to safe state → $A_1 | A_2 | A_3 | A_4 | \ldots | A_n$
- Kill a process responsible for deadlock.
- Kill all processes

→ Questions on lecture - 29 ✓

* Inter Process Communication & Process Synchronization :-

→ Process which are running in interleaving fashion b/c of their pre-emptive nature are called concurrently running processes. eg→ Reader - writer Problem.
Producer Consumer "
Railway Reservation System.

→ Why synchro. is required?



→ order of exe → $P_2 \to 4$, $P_1 \to 1,2,3$, $P_2 \to 5,6$.

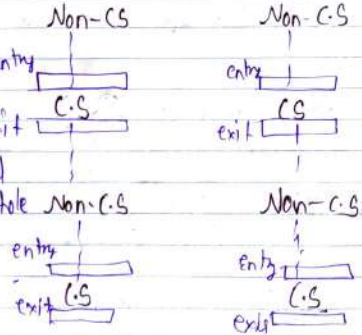
R_0	Count	R_1	
3	3 4 2	3	= 2.

- As per the above situations, if order of execution changes then final answer is getting changed and is called Race condition.
  → when final result depends on interleaving of execution of code.
- 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 synchronisation:

C.S → Critical section. P1 P2

So, we create entry and exit pair for the critical section as, they contain shared variable, and if it enters in critical section then first this whole section will execute then only it exits.



* Synchronization Requirement:

- Primary
  - Mutual Exclusion
  - Progress
- Secondary
  - Bounded waiting
  - Portability

(i) Mutual Exclusion ⇒ Only one process should access the shared variable at a time inside critical section.

(ii) Progress ⇒ No deadlock should be there & no process should permanently lock the entry to critical section for other processes.

(iii) Bounded waiting :- No starvation or Bound on waiting time.

(iv) Portability or Architecture Neutral :- No specific architecture or O.S is ~~connected~~ concerned. It should be generic.

* Solutions for synch. of processes :-

- Software based Sol^n
- O.S. Provided structure
  - Semaphore
  - monitor
- Hardware provided Instruction
  - test & set
  - Swap

* S/w based sol^n for Process synch.:-

Algo 1:- shared int lock = 0

* lock 1 is outside 'while' (no body of while).

	P1		P2
1.	while(lock != 0);	5.	while (lock != 0);
2.	lock = 1 (entry)	6.	lock = 1 (entry)
3.	C.S	7.	C.S.
4.	lock = 0 (exit)	8.	lock = 0. (exit)

→ P1: ① ② P2: ⑤ P1: ③ P2: ⑤ P1: ④ P2: ⑤, ⑥ ⑦
(P2: ⑤ → busy waiting; P2: ⑤ → busy)

P1: ① → busy waiting

→ P1: ① P2: ⑤ ⑥ ⑦ P1: ② ③
(⑦ → C.S; ③ → C.S) → Now two critical section at a time → voilation of mutual exclusion. sol^n i.e. Algo 1 failed.

Algo 2: shared int turn = 0;

	while(1){ P1		while(1){ P2
1.	while (turn != 0);	4.	while (turn != 1);
2.	C.S	5.	C.S
3.	turn = 1;	6.	turn = 0;
}		}	

P1: ① P2: ④ P1: ② ③ P2: ④ ⑤ P1: ①
(P2: ④ → Busy; ② → C.S access; ③ → " " complete; ⑤ → C.S access; P1: ① → Busy)

so, here mutual exclusion is ensured ✓. and now check progress.

→ No deadlock found as turn can only be 0 & 1. so both "while" will not infinitely wait.
→ Progress of process is voilating here as process P1 is blocking the entry in critical sec. of P2. as if P1 runs completely then P1 again can't run it will be in busy waiting same goes with P2.
P1 P1, P2 P2 → X P1 P2, P2 P1 ✓

Algo 3: shared int flag[2] = [0,0];

	while(1){ P1		while(1){ P2
1.	flag[0] = 1;	5.	flag[1] = 1;
2.	while (flag[1]);	6.	while(flag[0]);
3.	C.S	7.	C.S
4.	flag[0] = 0;	8.	flag[1] = 0;
}		}	

→ P1: ① P2: ⑤ ⑥ P1: ②
(⑥ → Busy; ② → busy) → both busy 'deadlock'

→ P1: ① ② ③ P2: ⑤ ⑥
(③ → In (C.S); ⑥ → busy) → Not able to come in C.S of P2

→ P1: ① ② ③ ④ P2: ⑤ ⑥ ⑦ P1: ① ②
(③ → C.S; ④ → out C.S; ⑦ → C.S; ② → busy) Not able to come in C.S of P1.

$P_1 \to C.S$ ✓ $P_2 \to C.S$ ✓ ⇒ M. Exclusion is ensured. but deadlock is there.
$P_2 \to C.S$ ✗ $P_1 \to C.S$ ✗

so failed Algo 3.

* Dekker's Algo & Peterson's Solution :-
↳ it is valid only for

Algo 4: shared int turn=0, 2 processes.
flag[2]={0,0};

P_1	P_2
1. flag[0]=1;	6. flag[1]=1;
2. turn=1;	7. turn=0;
3. while(flag[1] && turn==1);	8. while(flag[0] && turn==0);
4. C.S	9. C.S
5. flag[0]=0;	10. flag[1]=0;

P_1: ①,②,③,④ C.S P_2: ⑥,⑦,⑧ Busy

P_2: ⑥,⑦,⑧,⑨ C.S P_1: ①,②,③ Busy → so mutual exclus ensured.

progress check: $P_1 \to P_1$ ✓ working
$P_2 \to P_2$ ✓ "
$P_1 \to P_2$ ✓ "
$P_2 \to P_1$ ✓ "

No deadlock occuring.

passed → Algo 4.

* Semaphore :- • OS provided structures & it have 2 atomic oper^n.
1) wait() or down() or P 2) signal() or up() or V.
• OS generally switches off all interrupts when wait() & signal() are executed & they run at kernal level, so that context switch cannot take place

• eg:- Semaphore mutex = 1;

P_1		P_2
1. wait(mutex);	call to OS	4. wait(mutex);
2. CS		5. C.S
3. signal(mutex)	code	6. signal(mutex);

• Two types of semaphore: counting & binary.

1) Counting semaphore :→ Semaphore variable value indicates no. of processes can run critical section at a time (Generally). −∞ to ∞
→ Semaphore variable value can go in negative.

→ functional_description of 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 front(P1) of mutex.queue();
        wakeup(P1);
    }
}
```

Let semaphore mutex = 1.
mutex --- 0, -1, -2, -3
P1: C.S ← P2: P3 : P4, P5 → block
↓ ↓ ↓
block block block] after this C.S over and signal() comes.

[P2 | P3 | P4 | P5]
Queue

-2 : remove P2 from Queue and P2 entered (wakeup) in Critical sec. ↳ signal again
-2 : P3 in C.S ↳ signal
-1 : P4 in C.S → sig
0 : P5 in C.S

1 is not ≤ 0 come out.

→ wait() → decrementing the value if value < 0 → block
→ signal() → incrementing " " " " ≤ 0, wake-up (Queue(front)).

→ This code is inside OS and not at programmer level → no context switch

eg1 → semaphore mutex = 1;

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

P1 mutex ~~1~~ 0

P1 → C.S ↳ finished →
P2 → Block
P3 → Block
P2 → wakeup ↳ C.S → finish
P3 → wake up ↳ C.S → finish.

[P2 | P3 | | |]

mutex value = 1.

★★ Some questions on lecture 36 (last) & 37.

2. Binary Semaphore :- • Semaphore variable value can be either 0 or 1.
• Only one process can access Critical sec. at a time.

→ functional descrip. of Binary semaphore :

```
wait(Binary Semaphore m)
{
    if(m==1)
        m=0;
    else
    {
        Put the process P
        in m.list();
        block(P);
    }
}
```

```
signal(Binary Semaphore m)
{
    if(m.list() is Empty)
        m = 1;
    else
    {
        select a Process
        from m.list();
        wakeup(P);
    }
}
```

Que eg → How many processes can be present in C.S at max. at a time.
semaphore mutex = 1;

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

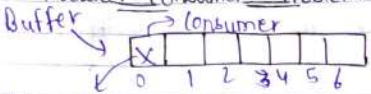
```
P10
while(1){
    signal(mutex)
    C.S
    signal (mutex)
}
```

Queue
[P2 | P3 | P4 | ... | P9]

mutex - ~~1~~, ~~0~~, -1, -2, -3 ... -7
1 ... -4, -5, -6

All processes can access C.S at a time.

* Producer Consumer Problem :-



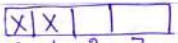
		Produce	Consum
Producer	empty-space = ~~4~~ 5	6	
	filled-" = 0 ~~1~~	0	

semaphore empty-space = size
filled-" = 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);
}
```

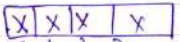
→



e = 4 3
f = 0

mutual exclusion violated.

→ Block



e = 4 3 4
f = 0 ~~1~~ 0

C_1 P_1 C_1 P_2 P_3 P_4 P_5 P_6 C_2

C_1 wake up
e.s = 4 ~~3~~ ~~2~~ ~~1~~ 0
f.s = 0
~~1~~ ~~2~~ ~~3~~ 4

P_6 block
C_2 wake up.

semaphore mutex = 1;

```
Producer
wait(mutex);
wait(empty-space);
C.S <- same as earlier
signal(mutex)
signal(filled-space)
```

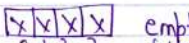
```
Consumer
wait(mutex);
wait(filled-space);
C.S <- same
signal(mutex);
signal(empty-space);
```

→ : we know if wait got -ve value it will block that process.

Let if P_1 P_2 came. mutex [1] 0 -1 → P_2 block
P_1
empty-space = [4] 3

→ so now two producer & two consume cannot came together

but let all buffer filled by producer and new producer come



P_{new} → block
empty-space = 0 -1
filled-" = 4
mutex = 1 0 -1 → when it will go to consumer's wait it will get blocked (as wait block when -ve value) here ~~then~~ both Producer and consumer has block so deadlock occured.

* Reader Writer Problem :-

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

R-R ✓, R-W → X, W-R → X, W-W → X

Semphore mutex = 1;

Writer
wait(mutex);
write;
signal(mutex);

Reader
wait(mutex);
Read;
signal(mutex);

mutex	R	W-1
1 → 0 → -1	↓	↓
	C.S	Block ✓

mutex	W	R-1
1 → 0 → -1	↓	↓
	C.S	Block ✓

mutex	W_1	W_2 →	W_3
1 → 0 → -1 → -2	↓	↓	↓
	C.S	Block	block

but R_1 R_2 should allow without any blockage.

mutex	R_1	R_2
1 → 0 → -1	↓	↓
	C.S	block

failed.

Semaphore mutex = 1;
Int reader_count = 0;

Writer
wait(mutex);
write; (C.S)
signal(mutex);

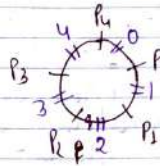
Reader
reader_count ++;
if(reader_count == 1)
wait(mutex);
Read; (C.S)
read_count --;
if(~~count~~ reader_count == 0)
signal(mutex);

mutex = 1 0 -1 -2
reader = 0 1 2 3
1 0 -1

W_1	R_1	R_2	R_3 →
↓	↓	↓	↓
C.S	Block	Allow	Allow
		→ problem.	

mutex	R_1	R_2
reader	↓	↓
1 0 -1	C.S	Block
0 1 2		

* Dining Philosopher Problem :- One person uses two chopsti-ck (neighbour).



Philosopher = 5
Chopstick = 5

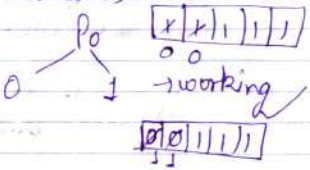
→ so race round condition can occur b/c 1 chopstic is shared b/w two philosopher.

Attempt 1:

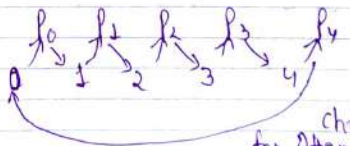
Semaphore Chopsticks[5] = {1,1,1,1,1}

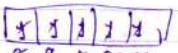
Phil. $P_{(i)}$
Acquire right chopstick.
wait(chopstick[i]); R_1
Acquire left chopstick.
wait(chopstick[i+1]%5); R_2
Eat; → C.S
Release left chop.
signal(chop[i+1]%5)
Release right chop.
signal(chop[i]);

Let P_0 comes



But Let P_0 P_2 P_4 P_3 P_4 comes together. and context switch occurs.



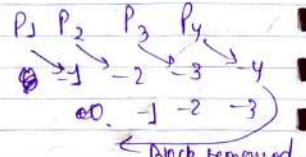


but they can't eat with 1 chopstick so they have to wait for other 1. and hire cycle of

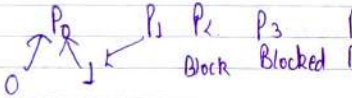
wait comes and deadlock occurs.

Attempt 2: semaphore mutex = 1;
chop[5] = {1, 1, 1, 1, 1}

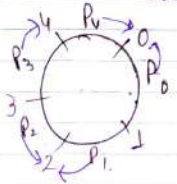
Philosopher P_i:
wait (mutex);
wait (chop[i]); → right
wait (chop[(i+1)%5]); → left
signal (mutex);
Eat; → C.S
signal (chop[(i+1)%5]);
signal (chop[i]);



☆ Here deadlock prob is removed, but poor resource utilization occured. b/c when P_0 acquire 0 & 1 and P_1 go inside & waiting for 1 chop. blocking the Gate (entry section).
Now chop 2, 3, 4 are idle which could be used by P_2 or P_3 but can't used.

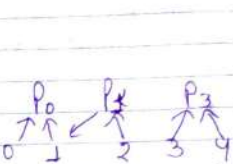


Attempt 3: → odd phil. will pick left chop. first.
→ even " " " right " ".



odd → 1, 3
even → 0, 2, 4

P_i (code):
If (i%2 == 0) {
// even phil.
wait (chop[i]); → right
wait (chop[(i+1)%5]); → left
}
else { // odd
wait (chop[(i+1)%5]); → left
wait (chop[i]); → right
}
Eat; C.S
release

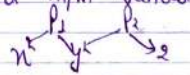


resources are getting utilized properly.

☆ No deadlock, No poor utilization.
☆ But starvation is occuring.
☆ one more ques from lecture 47.

→ Use of Dining Philosopher Problem :-

1> five processes sharing 5 resources.
2> shared m/m variables updated by 2 processes



* Hardware provided Solution :-

→ Test and Set :- Atomically returns the original value & set the variable to 1.

functional description :-
int Test & Set(int &n)
{
int temp = n;
n = 1;
return temp;
}

int lock = 0 → 1
Set lock.
test & set (lock)
↳ will return 0, and

P_1 P_2

eg → int lock = 0;

```
while (Test and set (lock));
  C.S
lock = 0;
```

```
while (test & set (lock));
  C.S;
lock = 0;
```

lock = 0 → P_1 → C.S; P_2 → Busy waiting

mutual exclusion ensured ✓

→ test & set is actually provided as an Hardware instr^n and mutual exclusion is ensured, but not bounded waiting.

→ Some other higher level algo can ensure Bounded wai-ting like checking the processes in sequential order for waiting states.

→ Spin lock problem can also occur.

```
Pi
while (test & set (lock));
  C.S
lock = 0;
```

lock = 1

P_0 → C.S; P_1 → higher prio. proc → Busy waiting

when P_0 is in C.S & higher priority process P_1 comes then P_0 will be pre-empted & P_1 will be waiting at entry section.
while remaining in CPU so a kind of deadlock occ-ured called Spin lock.

→ Swap:
- Atomically swaps the values of two variable
- Each process has a local variable called key

Shared int = 0';

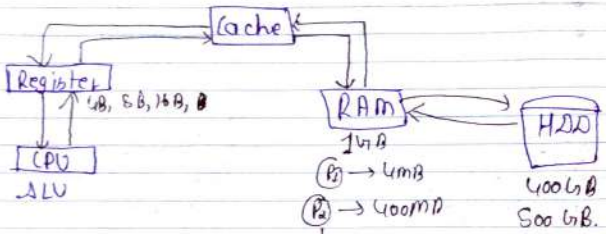
P1
```
int key1 = 1';
while(key1 == 1) {
    swap(lock, key1);
}
C.S
lock = 0';
key = 1';
```

P2
```
int key2 = 1';
while(key2 == 1) {
    swap(lock, key2);
}
C.S;
lock = 0';
key2 = 1';
```

Key1	Key2	lock	
~~1~~	~~1~~	~~0~~	
0		1	Key1
	1	1	Key2 → busi waiting
	1	0	outside loop.

→ Hardware instr^n suffers from busy waiting & starvati - on (bounded waiting is not ensured).

* Introduction to Memory Management :



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

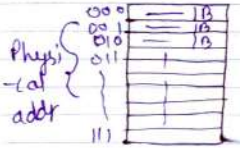
non - s/o s/o

→ In a system we have [Reg, cache, m/m] & (HDD).

→ In order to keep all levels work in sync. m/m management is req.

→ to increase the CPU% proper use of main m/m is req.

→ M/m can be accessed by using addresses.



1 word = 1 Byte.

8 Bytes = 2^3

⇒ 3 bits are req. to represent 8 addresses.

eg: 1MB m/m is built, what is the min req. of address size? provided that m/m is Byte addressable.

⇒ 1MB = 2^{20} Bytes.

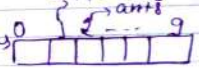
⇒ 20 bits are required for addrs.

→ Addresses → logical
→ physical.

logical addr ← 1000

```
int n = 20;
int *ptr = &n;
cout << ptr;
```
↳ logical addr

```
int an[10];
```

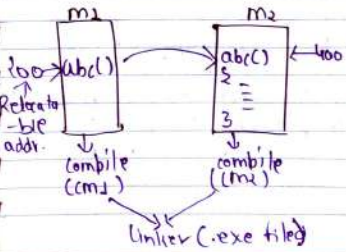


* Address Binding : Converting logical to physical addrs.

Compiler, Assembler, linker, loader

① error checking.
② convert code into assembly lang.

Assembler → Assembly to m/c code.



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

→ Modern compilers also include assemblers & linkers.

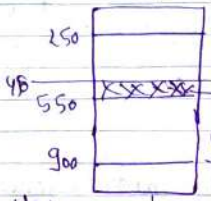
→ obj code is output of whole process, which consist of relocatable addrs.
↳ logical.

→ Loader will load the program into main m/m from HDD.

→ why logical addrs req. why not only physical?

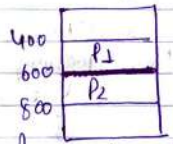
(i) If CPU directly gives physical addr, then each process is given physical addr during compile time or load time.

Let: $P_1 \longrightarrow 250$ to 550
$P_2 \longrightarrow 450$ to 900



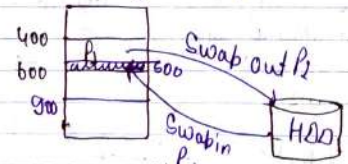
if P_1 is inside CPU we can't run P_2 and vice versa, it will degrade the degree of multiprogramming.

(ii) → $P_1 \longrightarrow 400 - 600$
$P_2 \longrightarrow 600 - 800$



∴ it is tightly packed so, if in case at run time if P_1 need extra 100 space so it is not available. → Difficulty in dynamic m/m allocation.

(iii) $P_1 \longrightarrow 400 - 600$



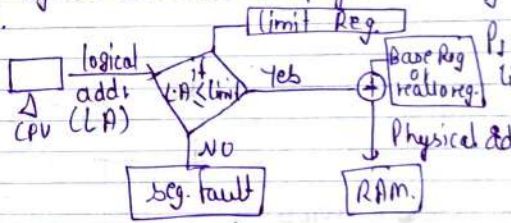
P_2 when swapout & P_2 swap in at 500 to 900. so, now i can't swap in P_1. → difficult to reallocate or swap in/out b/c each process will have to get same m/m partition, every time which may not be available

* Address_binding_types :-

1) Compile_time Binding : Compiler assigns physical addrs to m/m location.

2) Load time Binding: Compiler assigns logical addrs but loader assigns physical addrs during loading a process. or

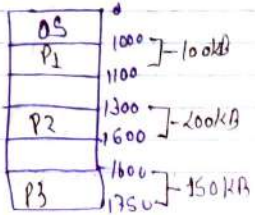
3) Execution time Binding: Compiler & loader assigns logical addr. during run time logical converted into physical using some Hard ware.



* Memory Allocation :- contiguous & non-contiguous

1) Contiguous Memory Allocation : When a whole process is kept in a single chunk (a single m/m partition).

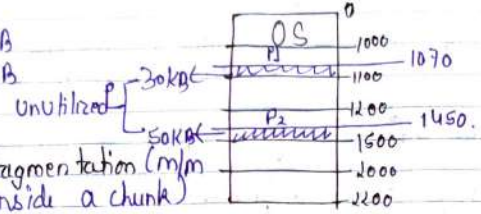
eg. $P_1 \longrightarrow 100$ kB
$P_2 \longrightarrow 200$ kB
$P_3 \longrightarrow 150$ kB



Partitioning.

→ (i) Static Partition: • m/m chunks are fixed.

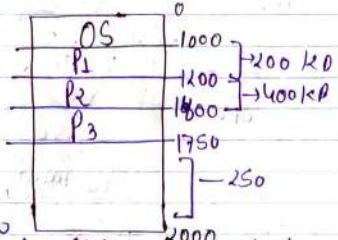
$P_1 \rightarrow$ 70KB
$P_2 \rightarrow$ 260KB



• Internal fragmentation (m/m wasted inside a chunk) is available.

(ii) Dynamic Partition: • m/m chunks are not fixed, they are created a/c to the process requirement.

$P_1 \rightarrow$ 200 KB
$P_2 \rightarrow$ 400 "
$P_3 \rightarrow$ 150 "



Now Let $P_4 \rightarrow$ 600KB comes and P_2 terminated, so now we have space of 250 + 400 (P_2) = 650KB but this is in two different blocks and even having m/m space we can't allocate P_4.

• When process terminates, the m/m is freed which will create a hole in b/w. which leads to external fragmentation.

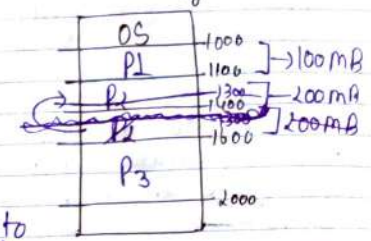
• External frag: when enough m/m is available but it is not contiguous, so can't be allocated to a process.

• Advantage is no internal frag.

• Solution for ext. frag. is called Compaction.

→ Compaction:
$P_1 \rightarrow$ 100mB
$P_2 \rightarrow$ 200mB

Let $P_3 \rightarrow$ 500mB.



So, shifting of P_2 is done to above block and then Below P_2, P_3 can settle
↳ 1100 − 1300

So time req. for doing this → shift

1B → 4nsec = 4×10^{-9} sec

200mB → $4 \times 200 \times 10^6 \times 10^{-9}$
= 800×10^{-3} sec
= 800 msec.

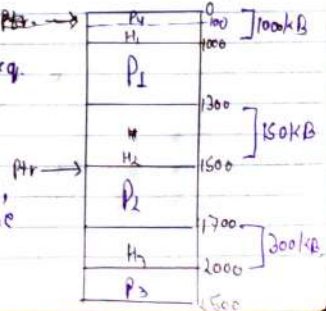
So for 800 msec the CPU is idle (No work), that is why it is costlier in terms of time.

*Memory Allocation Algos: first fit, next fit, Best fit, Worst ", Quick ". Hole

Let P_4 = 100KB

1) first fit: P_4 is allocated to H_1. (whichever first block of req. size or more is available)

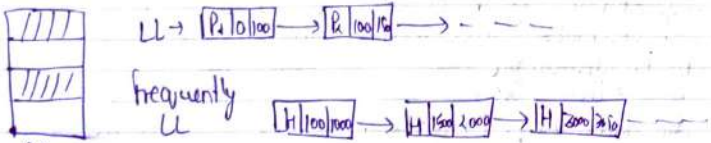
2) Next fit: P_4 is allocated to next available hole (block/chunk), where ever the pointer is last time allocation will be next to it.



3) Best fit:- Best suitable block or smallest possible block which can accomodate a process.
eg:- P_4 is allocated to H_3 i.e 150KB.

4) Worst fit:- largest block is allocated to a process
eg:- P_4 is allocated to H_1.

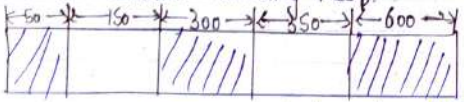
5) Quick fit:- In this we have a Linked list pointing to the frequently used m/m blocks. It will take the smallest m/m block pointer from that LL & allocate it to process.



P_n → processes, m → 2mb or 4mb (frequently)
n-m → various

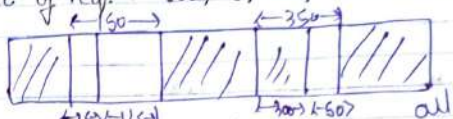
→ General notes:-
- first fit is most widely used.
- Best " " not always give best result.
- Quick fit is having overhead to manage extra LL to frequently used m/m blocks.
- Best fit & worst, searching is time consuming. (complete searching)

Ques (1994) → Consider following heap.



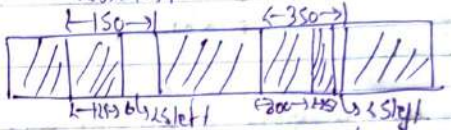
the sequence of req. → 300, 25, 125, 50. which formalized.

→ first fit



all allocated.

→ Best fit

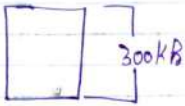


Now 50 is left and we have 50 space as well but not contiguous.
Ans → first fit but not best fit.

* Overlays:- If we can't accomodate whole process in m/m, we can divide the mutually exclusive code in different pass & run it.

P_1 →
Pass 1 → 100 KB
Pass 2 → 200 KB
common routines → 50 KB
Symbol tables → 40 KB
Pass driver → 10 KB. (which pass execute first)

total m/m → 100 + 200 + 50 + 40 + 10
= 400 KB

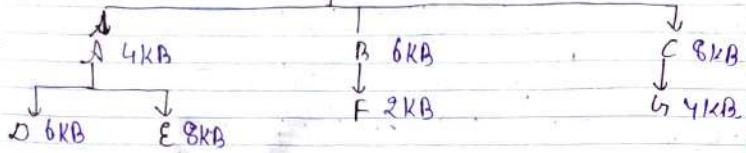


Now both pass can't run together so,

Pass 1 → 100KB + 50 + 40 + 10 = 200KB
" 2 → 200 + 50 + 40 + 10 = 300 KB.

So we can run P_1 in 300 KB

Ques (1998) Root 2KB



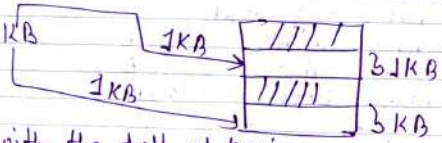
What is will be the size of partition (in physical m/m) req. to load (and run) this prog.

a) 12KB b) 14KB c) 10KB d) 8KB.

Ans → Path → A Root → A → D = 2 + 4 + 6 = 12 KB
" → A → E = 2 + 4 + 8 = 14 KB
" → B → F = 2 + 6 + 2 = 10 KB
" → C → G = 2 + 8 + 4 = 14 KB

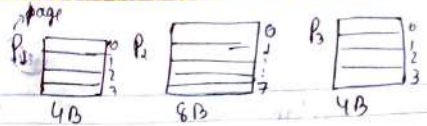
---

* Non-Contiguous m/m allocation :- Process can be allocated at different blocks.

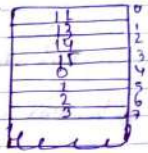


This is done with the help of paging.

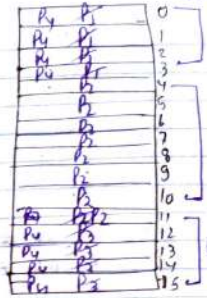
* Paging :-
- m/m is divided into frame.
- process " " " page.
- page size = frame size.



Now P1 & P2 terminated & P4 come with 8B so there we use non-contiguous m/m alloc.

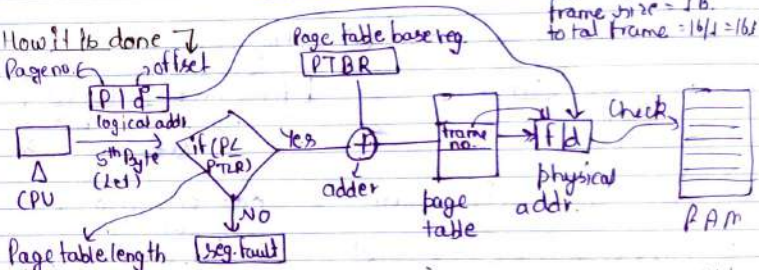


Page table will be created. It act as a index.



Ram = 16B
frame size = 1B.
total frame = 16/1 = 16

How it is done ↓



Let Logical addr = 5
page no. = $\left[\frac{5}{4}\right]$ = 1 $\xrightarrow{\text{Calculate frame no.}}$ = (2) & offset = 5%4 = 1.

(2) in diagram which is not made

Physical addr = frame no. × frame size + offset
= 2 × 4 + 1
= 9.

Now

4addr = 2^2 → bit

0 → 00 → 1st
1 → 01 → 2nd
2 → 10 → 3rd
3 → 11 → 4th
4 → 100 → 5th

8addr → 2^3
RAM addr = 32 bit.
Size of RAM = 2^{32} B.

2^{10} B = 1KB ≒ 10^3 B	
2^{20} B = 1MB ≒ 10^6 B	
2^{30} B = 1GB ≒ 10^9 B	
2^{40} B = 1TB ≒ 10^{12} B	

① Logical addr / or Virtual addr

① Logical or Virtual addr. ⓜ Physical addr.

page no(p)	offset(d)
← 3bit →	← 2bit →
← 5bit →	

frame no(f)	offset(d)
← 4bit →	← 2bit →
← 6bit →	

∵ 3 bit can generate 8addrs = 2^3 = 000, 001, 010, 011, 100, 101, 110, 111

∴ max no. of page = $2^3 = 2^p$ (∵ each addr is a page)

∴ max size of page = $2^d = 2^2$ (∵ offset is of 2 bit, total addr = $2^2 = 4$.)

∴ max no. of frame = $2^f = 2^4$

∴ size of frame = $2^d = 2^2$ → Same always b/c of offset

∴ Max size of process = $2^p \times 2^d = 2^{p+d}$

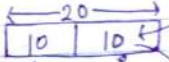
" " " m/m = $2^f \times 2^d = 2^{f+d}$

a process can

eg① Page Size = 1KB. Logical addr = 20 bits.
a) draw virtual addr format b) How many max pages can have?
c) If there are total 2^{16} frames in m/m. Draw Physical addr format.

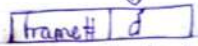
Ans page size = 1KB = 2^{10} B. 10 bits = 2^8

a) logical addr format.



b) $2^p = 2^{10}$

c)



$2^{16} \Rightarrow f = 16$

→ total.

① m/m size = 2MB, page size = 512B
a) How many pages are allocated to a process P1 of 5000 B.

Ans $\left\lceil \frac{5000}{512} \right\rceil$ = 10 page are allocated.

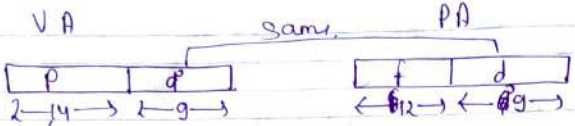
allocated m/m = 10 × 512 = 5120 B.

Contiguous → can be any internal or external fragm.
non-contiguous → only internal fragmentation.

b) max size of process = 6MB
Draw logical & phy. format.

∵ m/m size = 2MB = 2^{f+d} = 2MB = $2^{20} \times 2 = 2^{21}$
∵ page " = 512 B = 2^d
$2^d = 512$ B
$2^d = 2^9$ B ⇒ d = 9

∵ f+d = 21
⇒ f = 21 − 9 = 12



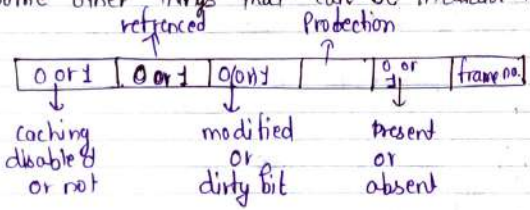
Now calculate p.

max size of process = 8 MB = 2^{p+d} = 8 MB
$= 2^{p+d} = 2^3 \times 2^{20} = 2^{23}$

$\because p+d = 23 \Rightarrow p = 23 - g = 14$. ✓

c) Max. no. of frames = $2^f = 2^{12}$.

Some other things that can be included in page table

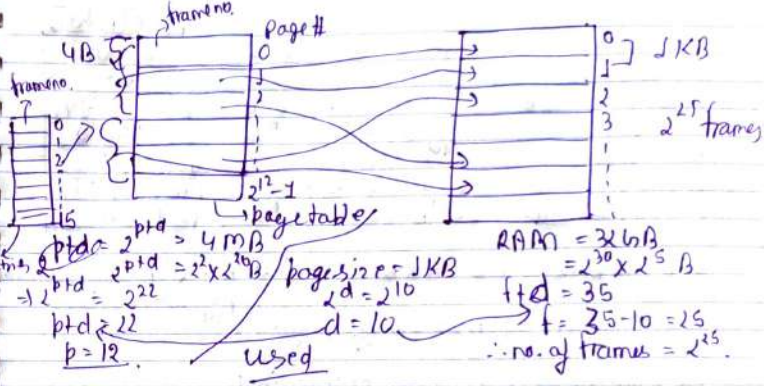


* 2-level paging :

eg → Logical add = 22 bits = p+d.
page size = 1 KB
$2^d = 2^{10} \Rightarrow d = 10$
$\Rightarrow p = 22 - 10 = 12$.
max no. of pages = 2^{12}

Now, Let say each page table entry is of 4 B.
size of page table = $2^{12} \times 4B = 2^{14} B$ or 16 KB.

max size of process = $2^{p+d} = 2^{22} = 4$ MB.



$p+d = 2^{p+d} = 4$ MB
$2^{p+d} = 2^2 \times 2^{20}$ B
$\Rightarrow 2^{p+d} = 2^{22}$
$p+d = 22$
$p = 12$.

Now, $\because 2^{12}$ pages are there in page table

$2^{12} \times 4 = 2^{14} B = 16$ KB (one entry is of 4 bytes)

so another page table is req. to store this one page table, and have 16 entries.

total size of new page table = no. of entries × 4 B
$= 16 \times 4 = 64$ B < 1 KB

Ques

ptd

[P3 | P2 | P1 | d]

	Virtual Addr	Page size	Page table entry size	PT1 size	PT2 size	PT3 size	Logical addr format
1	48 bits	16 KB	4B	2^{36} B	2^{24} B	2^{12} B	10 | 12 | 12 | 14
2	64 "	1 MB	4B	2^{46} B	2^{28} B	2^{10} B	8 | 18 | 18 | 20
3	72 "	1 GB	4B	2^{58} B	2^{16} B	—	14 | 28 | 30
4	72 "	256 MB	4B	2^{46} B	2^{20} B	—	16 | 26 | 28
5	72 "	16 MB	4B	2^{50} B	2^{28} B	2^{6} B	4 | 22 | 22 | 24

1. V.A = 48 bits, P.S = 16 KB $= 2^4 \times 2^{10}$ B $= 2^{14}$ B $= 2^d \Rightarrow d = 14$ bits
$\therefore p = 48 - 14 = 34$

Entry = 4B, max no. of pages $= 2^{34}$

page table size = total entries × 4
= $\frac{\text{total size}}{\text{page "}} \times 4 = \frac{2^{48}}{2^{14}} \times 4 = 2^{36}$ B
$= 64$ GB

> page size.

P.T.2 size $\rightarrow \frac{2^{36}}{2^{14}} \times 2^2 = 2^{24}$ B
> page size.

P.T.3 size $= \frac{2^{24}}{2^{14}} \times 2^2 = 2^{12}$ B < page size ✓

No. of entries in PT3 $= \frac{2^{12}}{2^2} = 2^{10}$

2. V.A = 64 bits, P.S = 1 MB $= 2^d = 2^{20}$ B $\Rightarrow d = 20$.
$p = 64 - 20 = 44$

max no. of pages $= 2^{44}$

P.T.1 size $= \frac{2^{64}}{2^{20}} \times 2^2 = 2^{46}$ B > page size.

P.T2 size $= \frac{2^{46}}{2^{20}} \times 2^2 = 2^{28}$ B > page size (2^{20})

P.T3 size $= \frac{2^{28}}{2^{20}} \times 2^2 = 2^{10}$ B < " " "

No. of entries in PT3 $= \frac{2^{10}}{2^2} = 2^8$

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

(iii) V.A = 72 bits, P.S = 1 GB $= 2^d = 2^{30}$ B $\Rightarrow d = 30$.
$\Rightarrow p = 72 - 30 = 42$.

$\therefore$ max no. of pages $= 2^{42}$.

P.T1 size $= \frac{2^{72}}{2^{30}} \times 2^2 = \frac{2^{74}}{2^{30}} = 2^{44}$ > P.S.

P.T2 size $= \frac{2^{44}}{2^{30}} \times 2^2 = \frac{2^{46}}{2^{30}} = 2^{16}$ < P.S. ✓

No. of entries in P.T2 =

Ques → Page size = 2KB, P.T.E = 4B
Outer page table size = 2KB, levels of Paging = 3.
Virtual add" space (max process size) = ?

Ans Outer page table = 2KB = 2^{11} B.
total entries = $\frac{2^{11}}{2^2} = 2^9 \rightarrow$ 9 bits
→ P3.

P3	P2	P1	d
9	9	9	11

∵ Outer is filled → P2 & P1 is also filled.
$\therefore p = 9 + 9 + 9 = 27$

$\therefore$ Virtual addr space $= 2^{p+d} = 2^{27+11} = 2^{38} = 256$ GB

- if outer p.table size = 128B
  ⇒ Not completly filled (PT3).

outer P.T size = 128B
→ No. of entries = $\frac{2^7}{2^1} = 2^5$ entries

PT3	PT2	PT1	d
5	9	9	11

← 34 →

∴ No. of entries in PT2 = $\frac{2^{11}}{2^2} = 2^9$

∴ virtual Addr = 2^{34} = 16 GB

* Effective Access time (EAT):-

memory access time = t_m

∴ for 1 level paging

effective access time = time to access page table (t_m) + time to access = $2t_m$. frame (t_m)

for 2 level paging

effective access time = time to access page table 2. (t_m) + time to acess page table 1. (t_m) + time to access frame (t_m)
= $3t_m$.

∴ **K-level paging = $K \cdot t_m + t_m$**

Let t_m = 100ns

1-level paging = $2t_m$ = 100×2 = 200ns.
2- " " = $3t_m$ = 3×100 = 300ns.
3- " " = $4t_m$ = 4×100 = 400ns.

} Very large for bulk data.

So we use "translation lookaside buffer" (TLB). and we use 'cache' m/m for this. and EAT gets reduced. in this some of the pages out of that page table are Hi bring up in cache and others left overe there.

Hit ratio :- if page is found in TLB.
Miss ratio :- " page not " " ".

Effective Access time on using translation look aside buffer is:- In 1 level.

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

- t_{TLB} → time to access TLB
- t_m → m/m access time for instr"
- t_{TLB} → time to access TLB
- t_m → to search not found page in page table

Hit — same.

miss.

eg → t_{TLB} = 20ns, t_m = 100ns. Hit ratio = 80% = 0.8
miss " = 20% = 0.2

EAT = 0.8(20 + 100) + (0.2)(20 + 100 + 100)
= 140ns.

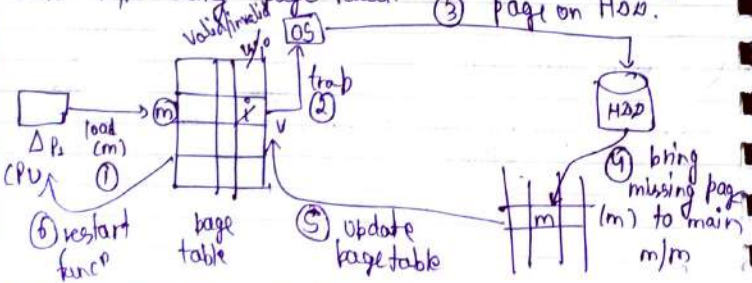
∴ for k-level paging E.A.T = $h*(t_{TLB} + t_m) + (1-h)(t_{TLB} + k \cdot t_m + t_m)$

2-level paging → $0.8(20+100) + (0.2)(20 + 2\times100 + 100)$
= 160 ns.

* Virtual memory: A part of secondary m/m treated as main m/m to increase the size of physical m/m is called virtual m/m.
(Programms)

* Demand paging: Processes generally resides on HDD, instead of loading all pages of a process or program in m/m, we can load whichever page is required, then & there itself on demand.

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



→ Page fault service time (PFST): time taken from ① to ⑥.

→ " " frequency (PFF): no. of page fault per n instr.

Effective memory Access time (EMAT): $PFF*(PFST + t_m) + (1-PFF)*t_m$

PFF → page fault freq probability
PFST → service time
t_m → m/m access time.
1-PFF → prob. of no page fault.

Qu: Gate: 2011,
↳ Let the page fault service time be 10ms in a computer with avg. m/m access time being 20ms. If one page fault is generated for every 10^6 m/m accesses, what is the effective access time for the m/m?

Ans PFST = 10ms, t_m = 20ns, PFF = 1 in 10^6 = $1/10^6$

EMAT = PFF (PFST + t_m) + (1-PFF)(t_m).
= PFF × PFST + PFF × t_m + t_m − PFF · t_m
= PFF × PFST + t_m
= $\frac{1}{10^6} \times 10 \times 10^6 + 20$

1 sec = 10^9 ns
1 sec = 10^9 ns
1 m Sec = 10^6 ns.

= 30 ns.

2) Note 2000.
Suppose the time to service a page fault is on the average 10ms, while m/m access takes 1 microsecond. Then a 99.9% hit ratio results in avg. m/m access time of

Ans PFST = 10ms, tm = 1 usec.
PFF = (100 − 99.99) = 0.01% page faults in %
↳ freq → 0.0001 or $\frac{1}{10^4}$ → no in frac
in frac

$EMAT = PFF \times PFST + tm$
$= \frac{1}{10^4} \times 10 + 10^{-3} = \frac{10}{10^4} + 10^{-3} m = 2 \times 10^{-3} ms$
$= 2 \mu sec.$

* Page Replacement Algorithm:

(i) First in first out (fifo): which comes first will be replaced first.

(ii) Least recently used (LRU): which is used Least recently

eg. [2 | 3 | 1], 4 (upcoming)
least, most
replaced.

(iii) Optimal page replacement Algo: we focus on future requirements. the page will be replaced which will be used later in future.

Ques) Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 on a demand paged virtual m/m.

system running on a computer system that has main m/m size of 3 page frame which are initially empty. Let LRU, FIFO and OPTIMAL denote the no. of page faults under the corresponding page replacement policy. Then
A) Optimal < LRU < FIFO
B) Optimal < FIFO < LRU
C) Optimd = LRU
D) optimal = FIFO.

Ans fifo:

	1	2	3	2	4	1	3	2	4	1
1	✓	✓	✓	✓	✗4	✓	✗3	✓	✗1	✓
2					✗1	✓	✗2	✓	✗4	✓
3					✗2	✓	✗4	✓	✗3	✓
					replace with 4		replace with 2			

Page fault = 6 (6 times we have written)

LRU:-

	1	2	3	2	4	1	3	2	4	1
1	✓	✓	✓	✓	✗4	✓	✗3	✓	✗4	✓
2			✗1	✓	✗2	✓	✗4	✓	✗1	✓
3		✗1	✓	✗2	✓	✗4	✓	✗3	✓	✗4
		Lead		Lead						
				end						
					1	3	2	4	1	3

page fault = 9.

optimal:

	1	2	3	2	4	1	3	2	4	1
1	✓	✓	✓	✓	✗4	✓	✓	✓	✗4	✓
2					✗2	✓	✓	✓	✗1	✓
3					✗3	✓	✗2	✓	✗3	✓
					2		3			

Page fault = 5

Optimal < FIFO < LRU

miss ratio ↳	FIFO	LRU	Optimal
	$\frac{6}{10}$	$\frac{9}{10}$	$\frac{5}{10}$

Note 2014:- page frame = 3, string 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6
use optimal replacement algo.

Ans

1	2	3	4	2	1	5	3	2	4	6
1 2 3	✓	✓	✓	✓	✓	✓	↓ 4	✓	✓	✓	✓
			↓ 3				↓ 2				

1 2 3	6	1	2	3	4	2	1	5	3	2	4	6
1 2 4		✓	✓	✓	↓ 3	✓	✓	↓ 1	↓ 5	✓	✓	↓ 3
1 3 4	2 6											

P.f = 7.

Note → 2007- P.f = 3, string 1, 2, 1, 3, 7, 4, 5, 6, 3, 1.

Optimal page algo →

Ans

1	2	1	3	7	4	5	6	3	1
1 2 4	5 6	✓	✓	✓	✓	↓ 2	↓ 7	↓ 4	↓ 5	✓	✓
3						↓ not coming afterwards					

P. fault = 7.

LRU →

1	2	1	3	7	4	5	6	3	1
1 4 3	✓	✓	✓	✓	↓ 2	↓ 1	↓ 3	↓ 7	↓ 4	↓ 5
2 4 6										
3 8 1										

p.f = 9
↳ 2 more than optimal

* Belady's Anamoly :- In fifo if we increase no. of frames, there is a chance of increase in page fault, this problem is called same. but in LRU & Optimal nothing such happens. in that either page fault will be equal or less on increasing no. of frames.

eg → Ref string 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4.

a) Page frame = 3

0	1	2	3	0	1	4	0	1	2	3	4
0 3 4	✓	✓	✓	↓ 2	↓ 0	↓ 1	↓ 2	↓ 3	✓	↓ 4	✓	↓ 5
1 0 2											
2 1 3											

page fault = 9

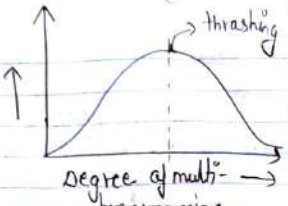
b) Page frame = 4.

0	1	2	3	0	1	4	0	1	2	3	4
0 1 4	✓	✓	✓	✓	✓	↓ 2	✓	✓	↓ 3	↓ 4	↓ 5
1 0 2											
2 1 3											
3 2 4											

page fault = 10.

General points:
- Sometimes FIFO suffers from Belady's Anamoly.
- LRU & Optimal follows stack algo but FIFO does not follow it always, because of this, FIFO suffers Belady's Anamoly
- Optimal algo always gives best results because it can transform itself into LRU, MRU based on situation.

* Thrashing :- As ~~the degree~~

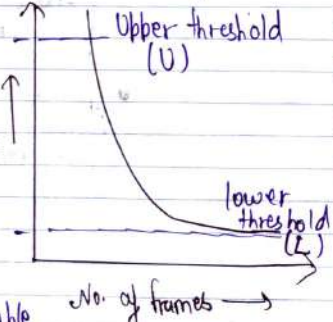


As degree multi-prog. increases, each process gets lesser m/m frames, Thrashing starts or is a situating where mast processes do not have their locality in a m/m. & therefore they frequently do page faults & when no. of page faults are so high then, CPU% decreases.

Problem :
→ CPU % is very low
→ Hard disk usage is very high → time consuming
→ Very large no. of processes are running but no-one is completing.

* Measures taken by O.S to remove thrashing :

a) Page fault freq. base monitoring :- OS will monitor PFF of each process. If



→ If $PFF_i > U$, then OS will try to give more frames from other processes
→ If $PFF_i < L$, OS will try to take/snatch frames from process P_i
→ If there are no frames available then OS will swap out process P_i.

b) Working set Algo :- → It assumes that nearest future is the close approximation of recent past.
→ It is used to predict the no. of frames for a process dynamically. (Specially in case of blocked processes).
→ Working set is the no. of pages it can look at a time in recent past.
→ Window is the max size of the working set.

eg. ref. string → 1, 2, 3, 1, 2, 4, 1, 4, 2, 1, 5, 1, 2, 4, 2, 1
window size (Δ) = 4, working set = W.

W	Δ
$W = \{1\}$	1
$W = \{1,2\}$	2
$W = \{1,2,3\}$	3
$W = \{1,2,3\}$	3
$W = \{1,2,3,4\}$	4
	12

W	Δ
$W = \{1,2,3,4\}$	4
$W = \{1,2,4\}$	3
$W = \{1,2,4\}$	3
$W = \{1,2,4\}$	3
$W = \{1,2,4,5\}$	4
	17

W	Δ
$W = \{1,2,4,5\}$	4
$W = \{1,2,5\}$	3
$W = \{1,2,4,5\}$	4
$W = \{1,2,4\}$	3
$W = \{1,2,4\}$	3
	17

avg frame req = $\frac{12+17+17}{16} = \frac{46}{16}$
(no. of string)

Memory Management Completed

Unit - IV

File Management

* **File** :- A file is an abstract data structure which holds the info. about a collection of data.

→ A file typically consist of name, type, location, size, protection, time & date.

→ **Operations** :- Create, delete, Remove, write, read, move copy etc.

* **Accessing a file** :- (P_1 process which is stored) P_1 → line 70 (read) abc.txt 100 lines

P_2 → line 68. (read) abc.txt 100 lines

1) Per process table (local to a process) :-

P_1 process table

filename	Pointer	Permission
abc.txt	7adf	R

P_2 process table

filename	Pointer	Permission
abc.txt	68df	WR

2) Global table :- for all the open files related to all process

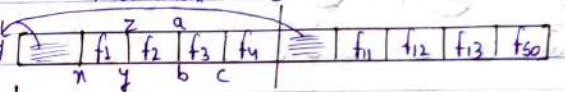
file name	file location	file open count
abc.txt	addr^{s} to HDD location	2 ~~1~~ 0 → file will get closed when this count is 0

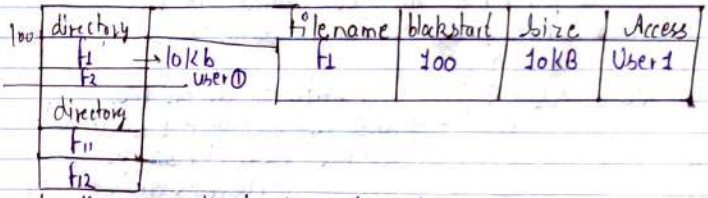
∴ initially 2 → b/c 2 process wants to access "abc.txt", then P_1 terminated count = 1 the P_2 used and terminated count = 0 → file closed.

info. req. to access file → file pointer, file open count, disk location of a file. Access rights.

* **HDD (Hard Disk Drive)** :- Permanent storage devices.
  - → It is divided into Blocks.
  - → m/m (RAM) is divided into frames.
  - → Process " " Pages.
  - → Block size = frame size or page size

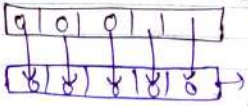
* **Directory** :- It is also an Abstract Data struct. like file
  - → It holds the info. about collection of files.





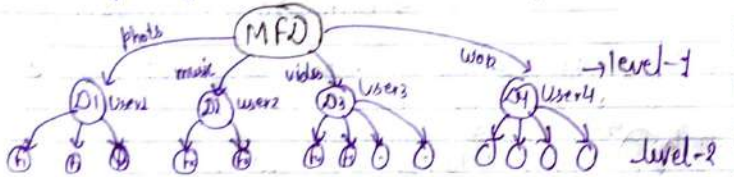
→ Operations on directory :- → Search → Remove
→ Create → List
→ delete → Traverse.

* **Single level directory** :- files at level-1.
  - → advantage → • Implementation is very simple
    - • Searching is easy.
    - • Best for small sized files



→ disadvantages → • Not suitable for large files.
• Naming is difficult. → No two files with same name.

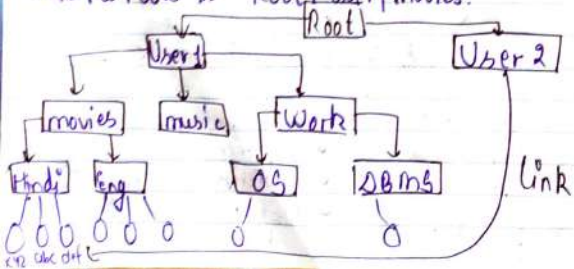
* 2-level directory: • At first level, Master file Directory (MFD).
• MFD contains the info. of various directory at level-2 & their access rights.
• At level-2, various other directories having various data of single user in each or one type is available.



disadvantages → • Cannot make directory under a directory.
• Sharing of file is not possible eg: d1 has print file can't be shared to d2.
• Sharing of individual data is not possible.

* Tree Directory: path → a route to access file.
→ absolute: Root\user\movies\Hindi\xyz.
→ relative: PWD\Hindi\xyz.
↳ present working director.

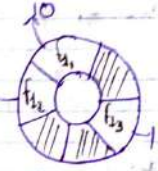
Here for PWD is Root\user\movies.



→ Sharing of files can be done by using links (Linux) ms-dos doesn't use links.
→ Because of these links a cycle is formed called acyclic graph.
→ Deletion is difficult in the links → counter of links is used to keep track of all links when counter = 0 → delete.

* Accessing Methods:

1) Sequential access: We access in sequential manner i.e used for application like video or music or compilers.

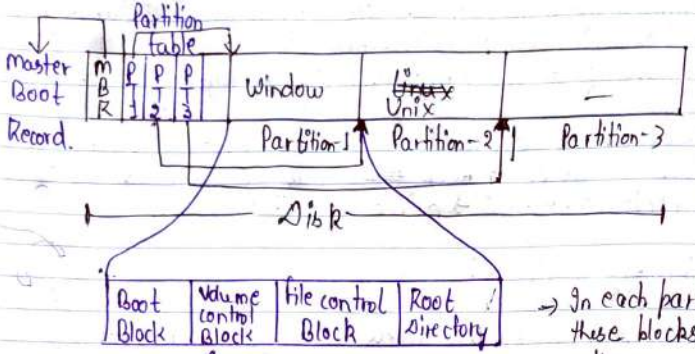


f1 → 2 KB
sequence → 10 → 12 → 15.

2) Direct Access: Used for apps like databases. accessing a particular block using its logical block addr called direct access

3) Indexed Access: In this we have a index table on basis of which we access our data.

* File System: → It is a method or Data Struct. that O.S. uses to keep track of files on disk
→ It allows the access to program & data stored on disk.
→ helps to keep track of files and their location
→ Allocation & de-allocation of space on disk.



→ In each partition these blocks are there.

→ MBR → it is info. loaded by BIOS (Basic I/p O/p System)

→ Boot block → It is used to load OS present in that Volume and is a section available in each volume.
→ In Unix it is called boot block, it is
→ In NTFSC New table file system) it is called partition boot sector.

→ Volume Control Block: → 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 organise the files.
→ In Unix, this includes file names & associated node numbers.

→ File Control Block: → It stores details about a file. like file permissions, date & time (Created, modified), size etc.

* Allocation Methods: Allocating space to files on disk efficiently, so that files can be accessed faster.

1) Contiguous Allocation:

Directory:

file name	Starting block No.	Length
abc.txt	5	4
Xyz	1	2
PQR	3 or 9, or 8 or 6	4

→ Problem: Is we do not have enough contiguous block, but we have 6 block left → (External fragmentation). or some times can be internal ") can occur.

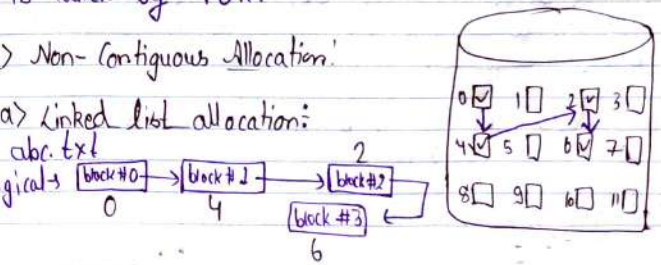
advantages: → Easy to implement. Reading is faster.

disadvantages → External & internal fragmentation
→ Can't grow the file size dynamically.

- means if abc req. 5 to 8 blocks and suppose PQR need 9 to 11 blocks, if in future abc req. more blocks then we can't assign to it because next contiguous space is taken by PQR.

2) Non-Contiguous Allocation:

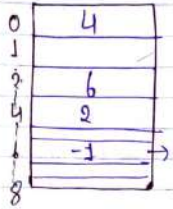
a) Linked list allocation:



Advantages :→ No external fragmentation.

Disadvantages :→ Random Access is slow.
→ Holding or storing the pointer to next block is an overhead.

• File Allocation Table (FAT): to speed up Random access. (To overcome disadvan. of L-L allocation to access randomly little faster.



disadvan → also have some size & it can be very large.

eg→ Block size = 1KB
HDD " = 1TB

∴ no. of entries in FAT = $\frac{1TB}{1KB} = \frac{10^9}{10^3} = 10^6$ entries

if each entry is of 4B
size of FAT = $4 \times 10^6 B = 4MB$

GATE-2014:- A FAT based sys. is used, total overhead of each entry in FAT is 4B. Given 100×10^6 bytes disk. data block size is 1KB. max. size of a file that can be stored on this disk in units of 10^6 bytes is ____.

entry size = 4B , Block size = $10^3 B$.
HDD " = $100 \times 10^6 B$

FAT Size = no. of entries in FAT × 4B.

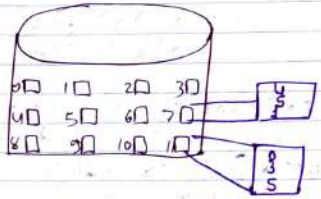
$= \frac{100 \times 10^6}{10^3} \times 4 = 0.4 \times 10^6$

∴ max size of fat = $100 \times 10^6 - 0.4 \times 10^6$
$= 99.6 \times 10^6$
∴ Ans = 99.6 ✓

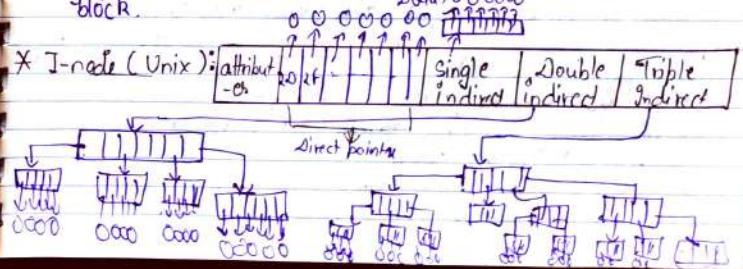
(b)→ Indexed Allocation → Solves the problem by bringing all the pointers together into one location: The index block

Directory

FileName	Index value
abc	7
xyz	11



→ If 1-block is not sufficient to hold all indexes of blocks, we can use multiple Index node.
→ Multiple Index nodes can be linked together using linked list, or can all indexes can be stored again in one more index block called outer index block.



GATE-2004: A Unix-style I-node has 10 direct pointer and one single, one double and one triple indirect pointers. Disk block size is 1 Kbyte, disk block addr is 32 bit, and 48-bit integers are used. what is the max possible file size?

Ans Size because of 10 direct pointers = $10 \times 1\,KB = 10\,KB$

" " " 1 single indirect pointer
= block size / each entry size = $\frac{1024}{4} = 2^8$
$= 2^8 \times 1\,KB$

" " " 1 doubly " " = $2^8 \times 2^8 \times 1\,KB$

" " " 1 triple " " = $2^8 \times 2^8 \times 2^8 \times 1\,KB$

max size of file = $10KB + 2^8 \times 1KB + 2^8 \times 2^8 \times 1KB + 2^8 \times 2^8 \times 2^8 \times 1KB$

$= 10KB + 2^8 \times 1KB + 2^{16} \times 1KB + 2^{24} \times 1KB$

On scaling/on comparing 2^{24} with $10, 2^8, 2^{16}$ we can neglect others $= 2^{24} \times 1KB$

$= 2^{24} \times 2^{10} B = 2^{34} B$

---

* Disk Layout: diagram in screenshot. /lecture 68.

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

→ If in studies some latency & delays:-

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

2> Rotational " : The time taken to come on right sector under head.

3> avg rotational latency = $\frac{1}{2}$ of Rotational latency

3> Data transfer latency: The time taken to transfer the data in or out of the disk. It majorly depends on rotational latency.

Ques → GATE - There are 10 platters in a disk, each platter have 20 tracks, each track have 50 sectors, each sectors can store 512 Bytes. find total size of disk.

Ans Size of 1 track = $50 \times 512\,B$

Size of 1 platter = $50 \times 512 \times 20\,B$

" " 10 " = $10 \times 20 \times 50 \times 512\,B$

if in ques is each platter has 2 surfaces then the answer will be Answer X2.

*** More Ques on lecture 69.

* Disk Scheduling Algo :→ Accessing different-different cylinders needs proper scheduling

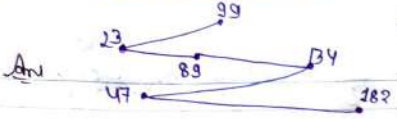
1> FCFS (First Come First Serve) :→ Simplest of all, easy to implement.

→ Perform operation in order of arrival

→ No starvation: every request is served.

Eg → A disk Queue with request for I/O to blocks on cylinders 23, 89, 134, 47, 182 with disk head initially at 99. find total no. of movement.

· 89 comes b/w 134 & 23 (short-cut)



Ans

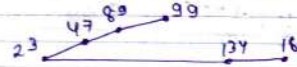
total head movement = (99-23) + (134-23) + (134-47) + (182-47)
= 76 + 111 + 87 + 135
= 409 movements

2) Shortest Seek time first (SSTF) :-

→ Select the disk i/o request that require the least movement of the disk arm from its current position, regardless of direction.
→ It reduce the total seek time compared to FCFS.

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

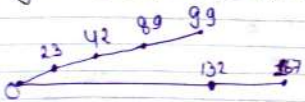
eg → same above ques → 23, 89, 134, 47, 182, initially 99



total head movements = (99-23) + (182-23)
= 76 + 159 = 235

3) SCAN (Elevator Algo) :→ Move from current position to the dir of movement given & in the opposite dir, till the all cylinders are accessed.

eg → 23 89, 132, 42, 187 initial → 99 & moving towards 0.

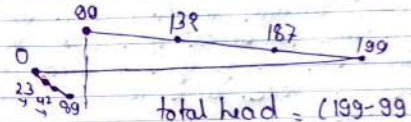


total head movement
= (99-0) + (187-0)
= 286.

4) C-SCAN (Circular Scan) :

eg → 93, 89, 132, 42, 187, initial at 99, moving towards higher cylinders. ~~higher cylinders~~

total cyli. = 200 i.e 0-199

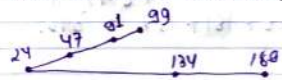


total head movements = (199-99) + (199-0) + (89-0).
= 100 + 199 + 89 = 388 movem.

→ Move from current position to the last cylinder (in dir) & jump to the opposite side cylinder (last) & without servicing any request

5) LOOK : → Just like SCAN.
→ But we do not go till the end instead revert the dir at the last cylinder request-ed of that side.

eg → 24, 91, 134, 47, 189, initial = 99, moving towards 0.

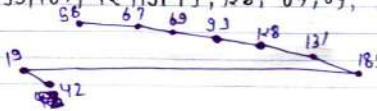


total head movement = (99-24) + (189-24)
75 + 165 = 240.

6) C-look :→ Just like C-Scan.

eg → 93, 189, 42, 131, 19, 128, 67, 69, initial = 58. moving towards higher

eg



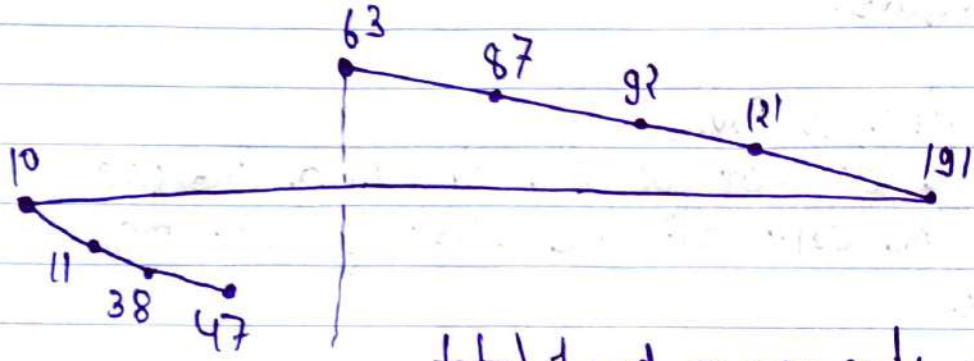
→ total head movement = (189-58) + (189-19) + (42-19) = 393

GATE-2016 :-

Consider a disk queue with request for I/O blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-look scheduling algo is used. The head is initially at cylinder no. 63, moving towards larger cyl. no. on its servicing pass. The cyl. are numbered from 0 to 199. The total head movement (in no. of cyl.) incurred while servicing these requests is: Note: Ques was asked as numerical answer type.

A) 346 B) 165 C) 154 D) 173

Ans → 47, 38, 121, 191, 67, 11, 92, 10 , initial = 63
move to higher ✓



total head movements = (191 - 63) + (191 - 10) + (47 - 10)
= 128 + 181 + 37 = 346.

Syllabus Done