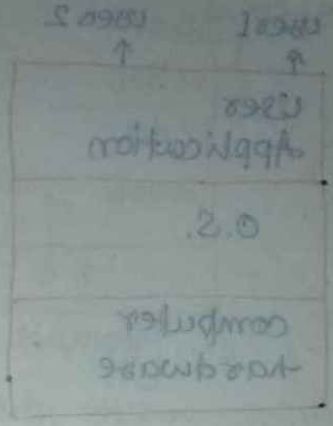
Meena Kanwar
Lodhi Rajput

OPERATING SYSTEM NOTES

- Introduction and Background :

* process Management
  - → process Concept
  - → CPU Scheduling
  - → Synchronization
  - → Concurrent programming
  - → Deadlocks
  - → Threads.

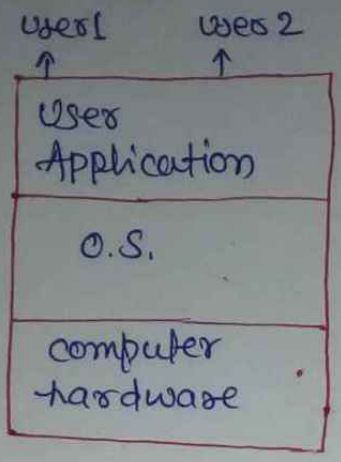


* Memory Management :
  - → RAM chip implementation
  - → Loading, Linking & Address Binding
  - → Technique → paging
    - → Multilavel paging
    - → Saugmentation
    - → Segmented paging
  - → Virtual Memory.

* File & Device Management
* protection & security

* Introduction & Background.

operating System :- Interface between users and the computer hardware.




```
main ()
{
int x;
printf ("Hello");
scanf ( — );
}
```

Monitor

recognise → keyword

→ printf internally called the write ( ) system call in order to communicate with the monitor.

System call:→ System call is a request mode by the users programmers to the O/S in order to get any kind of services.

* Operating is also called as Resource allocation (Responsible for alsating computer of a system)

Resource

- H/W Type — Eg. Memory Device
- S/W Type — Eg. file

* Goals for an operating system :-

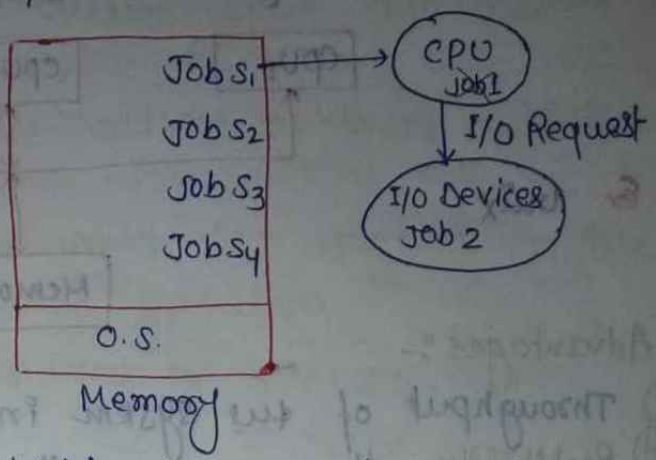
i) The primary goal of an O/S is convienience ↑ easy to user

ii) Secondary goal of an O/S is efficiency → stability of an O/

[Unix is more Efficient O/S than window xp]

* Type of Operating System :-

i) Batch O.S. → Eg. - IBM OS/2
group



Time
↓ ↓
CPU Time I/O Time

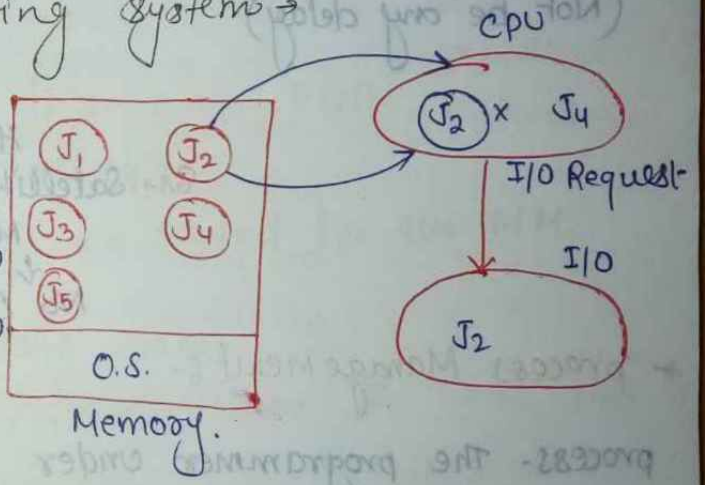
Strategy -

→ If the job is completed completely than only another job will be scheduled on to the CPU.

- problem - Inverse CPU idleness.
Throughput of the operating system will decrease.

Throughput → No. of job completed for a unit time.

ii) Multiprogramming operating system →



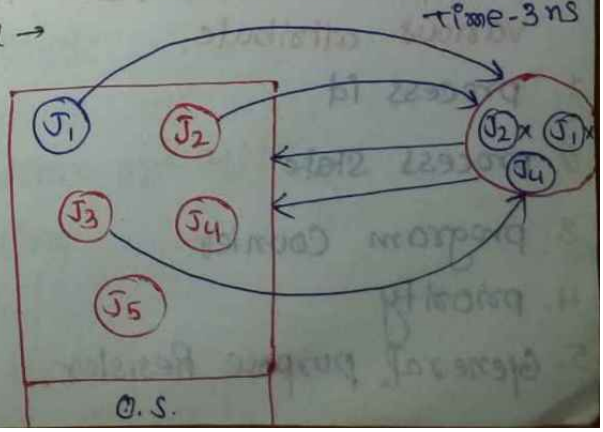
→ If the job is leaving the CPU to problem I/O operation than another job which is ready for execution will be scheduled on to to the CPU.

→ Increased CPU utilization.

→ Throughput of the system will also increase.

Ex- Windows, Unix, Linux.

iii) Multitasking operating system →

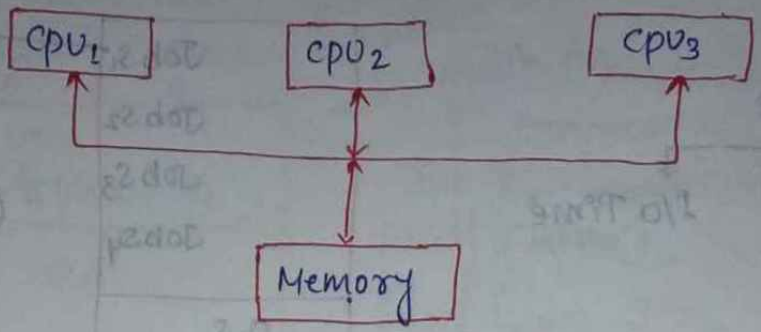


→ Multitasking is an extension to multiprogramming O.S.

→ Job will be executed on the CPU in the time sharing mode.

Ex- Windows, Unix, Linux.

iv) Multiprocessor operating system :- (parallal system)
Suggustable only for more no. of progr.



Ex. Unix

Advantages :-
i) Throughput of the system increased.
ii) Realibility - If one cpu fails than system will run with the help of remaining CPU.
(Fault tolerant system), Exceptance.
iii) Economical - The cost will be less.

v) Real time Operating system :- The system which are strict deadly time bound are called as Real time system.
(Not be any delay)

Hard Real time
Ex→ Satellite system, Missile system
No Delay - RT O.S.

Soft Real time
Ex. Banking sector
(Minor delay can be accepted)

* process Management :-

process - The programmer under execution is called as a process.

→ compulsory
→ It should be reside in M.M.
→ Occupy the CPU to execute the instruction.

→ The process will have the various attribute.

1. process id
2. process state
3. program counter
4. priority
5. General purpose Resister.
6. List of open file
7. List of open device
8. protection information

process id - unique identification no. which is assigned by the O.S. at the time of process.

process state - It contain the current state information of a process where it is residing.

program counter - It contain the address of next instruction to be executed.

priority - It is a parameter which is assigned by the O.S. at the time of process genration.

General purpose Resister - Resister used by the processor will be maintain.

All the attribute of a process is called as a context of the process.

* Context of the process will be stored in the PCB

Pid	P.S.
P.C.	priority
GPR	LOF
LOD	protection

} PCB Block context

* Every process will have it's own PCB.

PCB_7 → process (7)

* PCB of the process will be stored in the MM.

The process will have various state -

i) New
ii) Ready
iii) Run
iv) wait & Block
v) Termination & completion
vi) Suspend Ready
vii) Suspend wait or suspend Block.

Various operation -

i) Creation
ii) Scheduling
iii) Dispatching
iv) Execution
v) Termination or Killing
vi) Suspending
vii) Resuming

* Process State Diagram :-

(i) Initially the process will be in the new state. It means the process in under creation or being created.

(ii) Once the process is created it will be removed to ready state.

(iii) In the ready state there will be multiple no. of process.

(iv) One of the process will be selected from the ready state and that will be dispatched on to the running state.

(v) In the running state there will be only one processor at any point of time.

(vi) If the running process require only I/O operation it will be removed to wait or Block state.

(vii) In the wait state there will be multiple no. of process it means multiple process will perform I/O operation simultaneously.

(viii) When the process is ready running & wait state it is residing in the MM.

Multiprogramming O.S.

(formally to do) disturb by priority, Time quatum

Non-premptive Multiprogramming

preemitive Multiprogramming or Multitasking or time sharing

- But not all multiprogramming is multitasking.

(ix) When the resources are not sufficient to manage the process in the ready state, then some of the process will be suspended than they will be moved to Suspended ready state. → secondary memory. Backing store.

(x) When the process is in the suspend ready it is residing in the backing store.

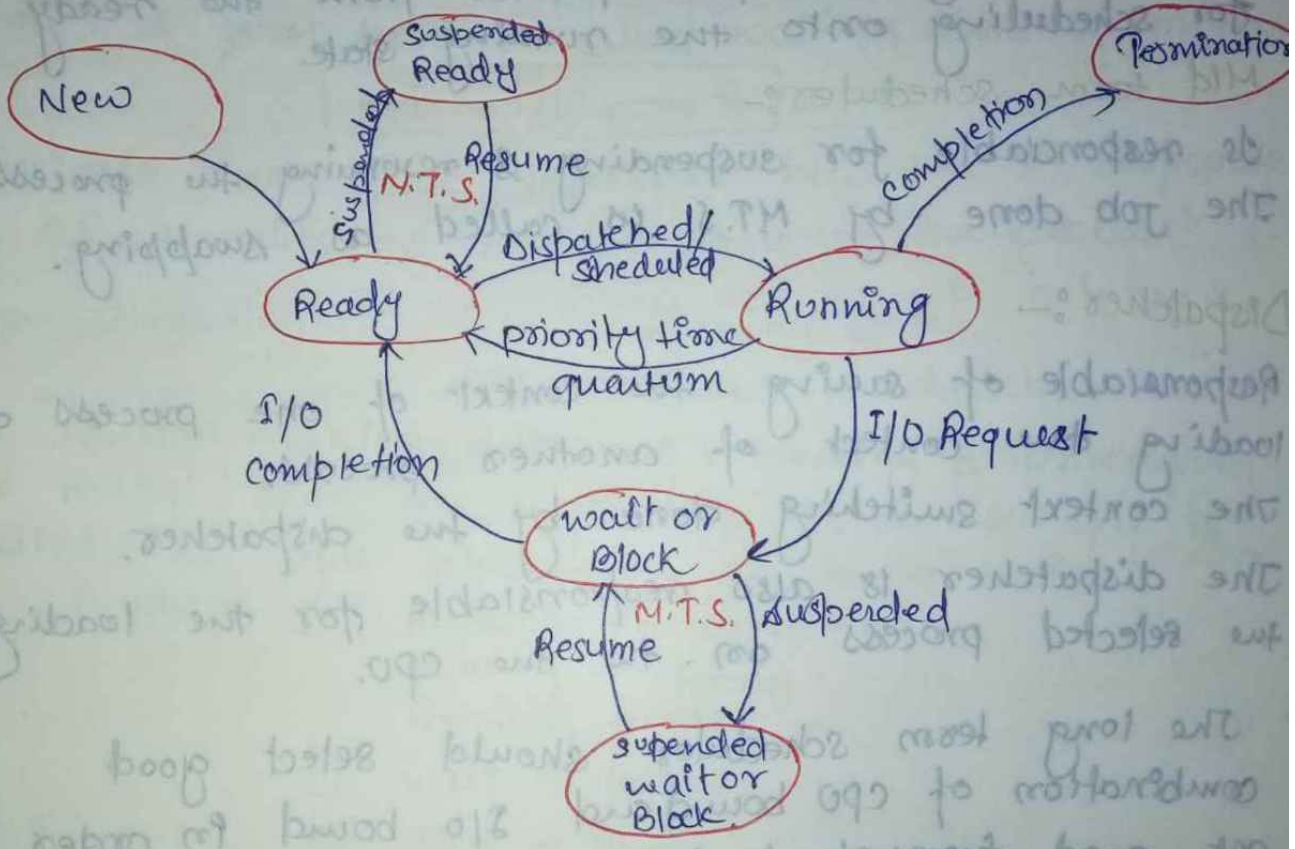
(xi) The process with respect to their execution time are of 2 type.

1. CPU Bound process
2. I/O Bound process.

1. CPU Bound process :- The process which requested more amount of CPU time are called as CPU Bound process. These process will be spend more time in the running state.

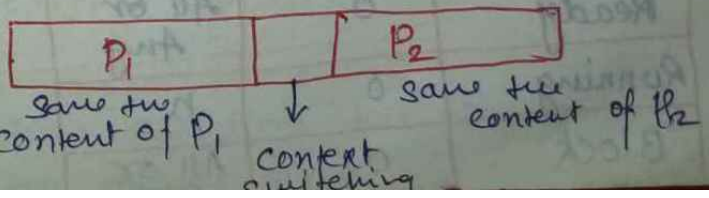
2. I/O Bound process :- The process which require more amount of I/O time is called I/O bound process. These process will spend more time in the wait state.

(1) Degree of Multiprogramming :- The no. of process present in the MM at any point of time is called as degree of multiprogramming.

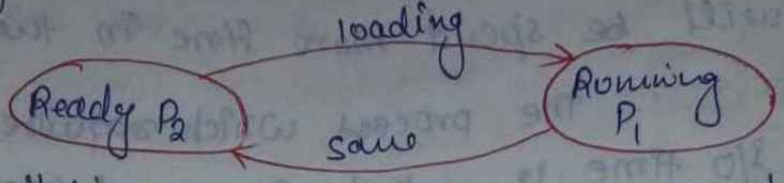


(1) Each and every time when the process is moving from one state to another state the context of the process will change it means that context switching will happened.

Context switching - Saving the context of one process and loading the context of another process is called as context switching.



If the context of the process is more than the context switching time will also increase which is undecidable.



Context switching time is consider overhead for the system.

In the O.S. these are three different Scheduler.

1. Long term scheduler :- Responsible of creating and beginning of new process into the system.
2. Short term scheduler :- (CPU Scheduler) - Is responsible for selecting one of the process from the ready state for scheduling onto the running state.
3. Mid term scheduler :-

Is responsible for suspending & resuming the process. The job done by M.T.S. is called as swapping.

(Dispatcher :-

- Responsible of saving the context of one process and loading the context of another process.
- The context switching done by the dispatcher.
- The dispatcher is also responsible for the loading the selected process on the the CPU.

→ The long term scheduler should select good combination of CPU bound and I/O bound in order to get good throughput for the system.

→ The L.T.S. control the degree of multiprogramming.

Qus → Consider a system which has n-CPU processor than what is the minimum & max. no. of processes that may present in the ready, running & block state.

	Min.	Max.
Ready	0	All or Any
Running	0	n
Block	0	All or any

* The process will have various different Time →

i) Arrival Time → The time when the process is arrival into the ready state is called as arrival time.

ii) Burst time → Required by the process for its execution.

iii) Completion Time → Time when the process is completed its execution.

iv) Turn around Time → Time difference b/w the completion time and arival time is called (TAT) of process.

$$T.A.T. = C.T. - A.T$$

v) Waiting Time → Time difference b/w the turn around time & burst time is called waiting time of process.

$$W.T. = T.A.T. - B.T.$$

vi) Response Time → Time difference b/w the 1st response & A.T. is called as reponse time of the process.

* CPU Scheduling- where → Running State
who → Short term scheduler

The decision making difference selecting a process in the ready state for scheduling on to the CPU is called as CPU Scheduling.

when- i) Running - Termination
Running → Ready
Running → wait
ii) wait → Ready
iii) New → Ready.

* Goal of the CPU scheduling →

i) Max. the CPU utilization and throughput of the system.

ii) To minimize the avg. waiting & T.A.T of the process.

* Scheduling Algorithm:-

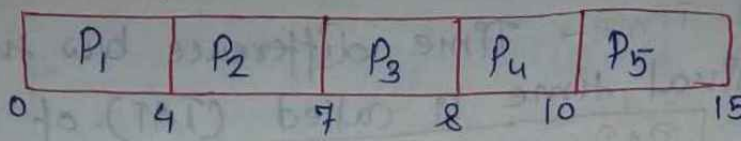
1) FCFS → First come First served.
Criteria → Arrival time
Mode → Non-pre-Emptive selecting the process based on arrival time.

Qu→

P.No.	A.T.	B.T.	C.T.
1	0	4	4
2	1	3	7
3	2	1	8
4	3	2	10
5	4	5	15

~~T.A.T = 15-0~~
~~= 15/5 = 3~~

Grant Chart.



Avg T.A.T → $P_1(4-0) + P_2(7-1) + P_3(8-2) + P_4(10-3) + P_5(15-$

$= 4+6+6+7+11 / 5 = \frac{34}{5} = 6.8.$

Avg W.T. $= \frac{0+3+5+5+6}{5} = \frac{19}{5} = 3.8.$

If the arrival time of the process are M acting then scheduled the process which has the lowest process id.

* Convey Effect :- In the FCFS the first process is having large burst time (CPU burst time) then it will have major effection waiting time of the process. This effect is called as convey effect.

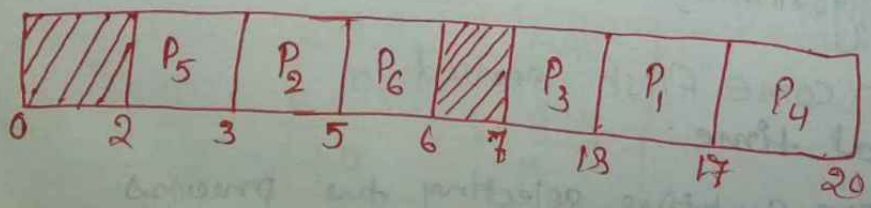
P.No.	A.T.	B.S.	C.T.	T.A.T.
1	8	4	17	9
2	3	2	5	2
3	7	6	13	6
4	10	3	20	10
5	2	1	3	1
6	3	1	6	3

Avg T.A.T = $\frac{9+2+6+10+1+3}{6}$

$= 5.1$

Avg W.T. = $\frac{5+0+0+7+0+2}{6}$

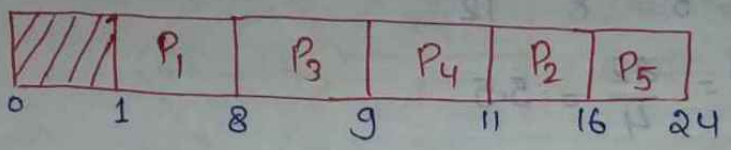
$= 2.3$



2) Shortest job first (SJF) :- Non- pre-Emptive.

Que :-

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11



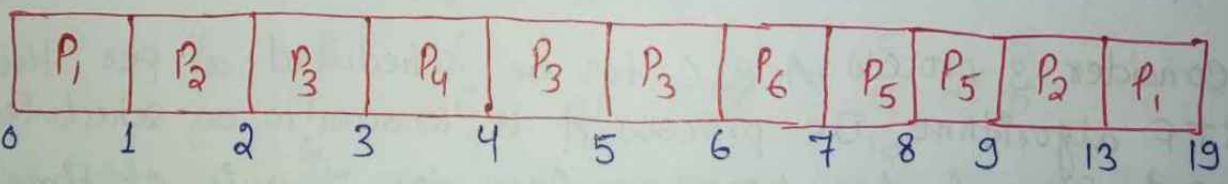
Avg W.T. = $\frac{30}{5} = 6$

Avg. T.A.T. = $\frac{53}{5} = 10.6$

3) Shortest Reamaining time First :- Criteria - Burst time
Mode - pre-Emptive

Que :-

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
1	0	7			
2	1	5			
3	2	3			
4	3	1			
5	4	2			
6	5	1			



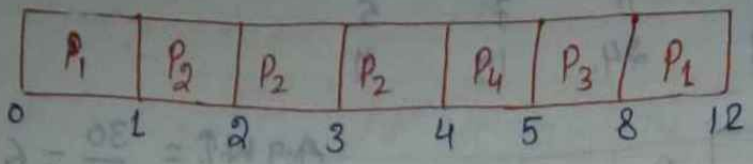
* When all the process are arrived then no need of checking (one unit) then you can continue with the shortest burst time.

Avg. W.T. = $\frac{24}{6} = 4$

Avg. T.A.T. = 7.1

Qus:-

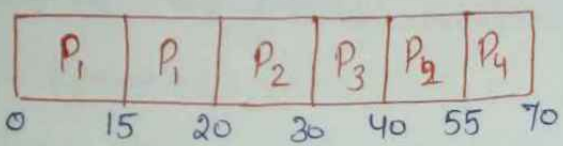
P.No.	A.T	B.T	C.T	T.AT	W.T
1	0	5	12	12	7
2	1	3	4	3	0
3	2	3	8	6	3
4	4	1	5	1	0



$$\text{Avg. T.A.T} = \frac{22}{4} = 5.5$$

Qus.

P.No.	A.T.	B.T.	C.T.	T.AT.	W.T.
1	0	20	20	20	0
2	15	25	55	40	15
3	30	10	40	10	0
4	45	15	70	25	10



* If the process is changing than it have the C.S. & if the process are some No. context switch.

Qus. Consider 3 process A, B, C to be scheduled as per the SRTF algorithm. The process A is known to be scheduled first and when A has been running for 7 unit of time, the process C has arrived. The process C has run for 1 unit of time, than the process B has arrived and completed running in 2 units of time than what could be the minimum Burst time of process A & C.

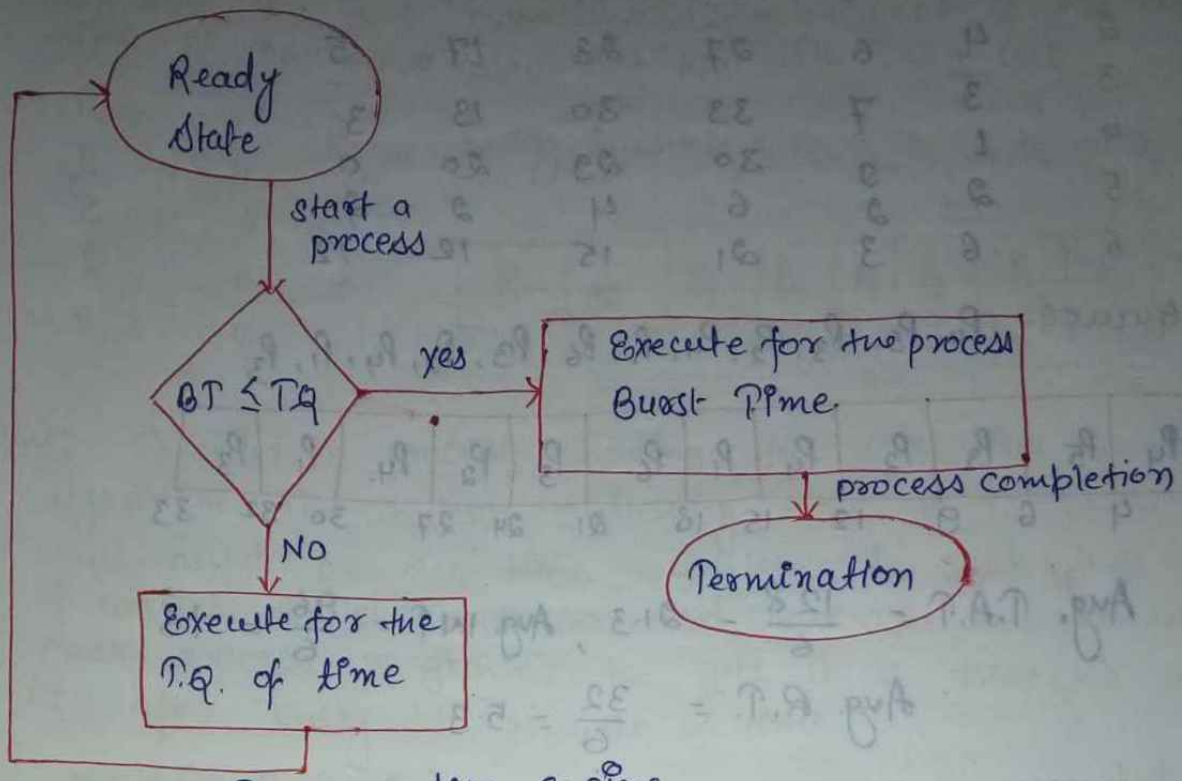
P.No.	A.T.	B.T.
A	7	
B		
C		

A → 7 + 5 = 12

B → 2

C → 1 + 3 = 4

④ Round Robin :- Criteria → Time quantum or Time Slice
Mode → preemtive

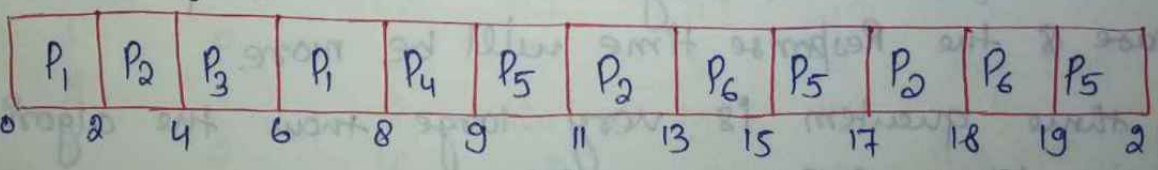


Qus :-

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.	R.T.
1	0	4	8	8	4	0
2	1	5	18	17	12	1
3	2	2	6	4	2	2
4	3	1	9	6	5	5
5	4	6	21	17	11	5
6	6	3	19	13	10	7

T.Q. = 2

Ready Queue :- $P_1, P_2, P_3, P_1, P_4, P_5, P_2, P_6, P_5, P_2, P_6, P_5$



Avg. T.A.T. = $\frac{65}{2} = 10.8$, Avg. W.T. = $\frac{44}{6} = 7.3$

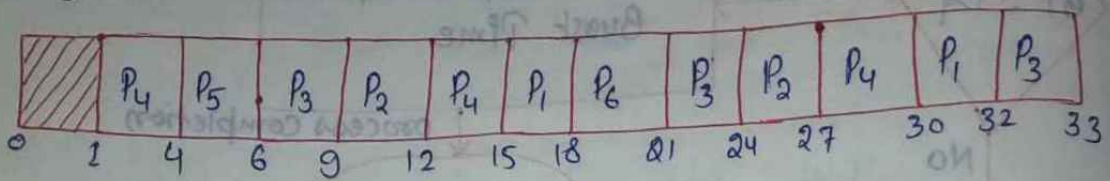
Avg. Response Time = 1st Response – A.T.

$= \frac{20}{6} = 3.33$

Qu:-

P.No.	A.T.	B.T.	C.T.	T.A.T	W.T.	R.T.
1	5	5	32	27	22	10
2	4	6	27	23	17	5
3	3	7	33	30	13	3
4	1	9	30	29	20	0
5	2	2	6	4	2	2
6	6	3	21	15	12	12

Ready Queue:- $P_4, P_5, P_3, P_2, P_4, P_1, P_6, P_3, P_2, P_4, P_1, P_3$



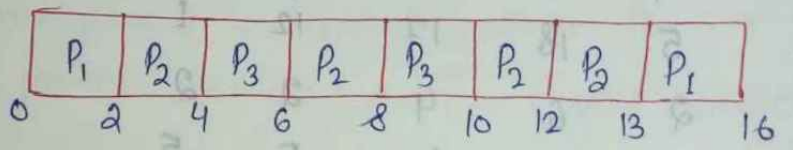
Avg. T.A.T = $\frac{128}{6} = 21.3$, Avg. W.T. = $\frac{96}{6} = 16$

Avg. R.T. = $\frac{32}{6} = 5.3$

Qu:-

process	A.T.	B.T.
P_1	0	5
P_2	1	7
P_3	3	4

T.Q = 2



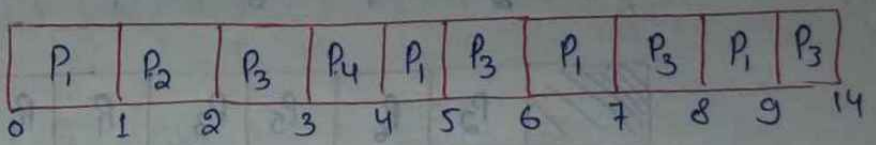
Ready Queue :- $P_1, P_2, P_3, P_2, P_3, P_2, P_2, P_1$

* 1) In the Round Robin if the time quantum is less then the no. of context switches will be increased time will be more decrease & the Response time will be more.
2) If Time Quantum is large than the no. of context will decrease & the Response time will be more.
3) If time quantum is very large than the algorithm behaves likes FCFS scheduling.

Qu:- Consider the 4 process P_1, P_2, P_3, P_4 arriving in the same order at the time 0. The burst time requirement of this job are 4, 1, 8, 1 respectively then what is the completion time of process P_1 assuming R.R. with T.Q = 1NS.

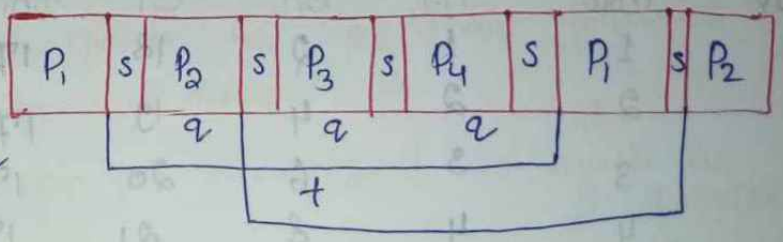
P.No.	A.T.	B.T.
P_1	0	4
P_2	0	1
P_3	0	8
P_4	0	1

Reddy Queue :- $P_1, P_2, P_3, P_4, P_1, P_3, P_1, P_3, P_1, P_3$



Qus:- Consider n process sharing the CPU in Round Robin algorithm. The context switching time is 's' units. then what must be the time quantum 'q' such that the no. of context switches are reduced but at the same, time each process is granted to gets it is turn/job at the CPU after every 't' seconds of time.

n = 4 (P_1, P_2, P_3, P_4)



$t = n * s + (n-1) q$

$(n-1) q = t - ns$

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

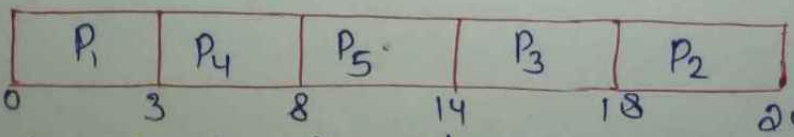
* longest Job first (LJF) :- Efficiency of the system will decrease
Criteria → Burst Time
Mode → Non-pre-Emptive

Qus:-

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
1.	0	3	3	3	0
2.	1	2	20	19	17
3	2	4	18	16	12
4.	3	5	8	5	0
5	4	6	14	10	4

Avg. T.A.T = $\frac{53}{5}$ = 10.6

Avg. W.T. = $\frac{33}{5}$ = 6.6



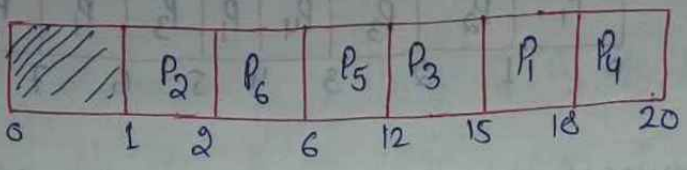
If the burst time of the process are matching than schedule the process which has lowest arrival time.

Qu:

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
1	3	3	18	15	12
2	1	1	2	1	0
3	2	3	15	13	10
4	4	2	20	16	14
5	6	6	12	6	0
6	2	4	6	4	0

Avg T.A.T. = $\frac{56}{6}$

Avg. W.T. = $\frac{36}{6}$ =



L.R.T.F - Longest Remaining Time First :-

Criteria → Burst Time

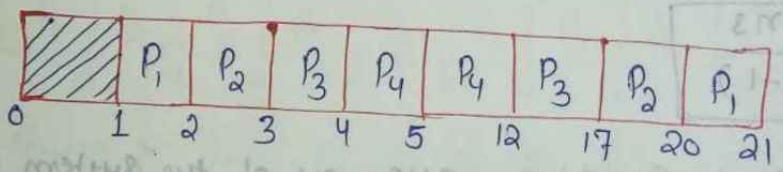
Mode → pre-Emptive

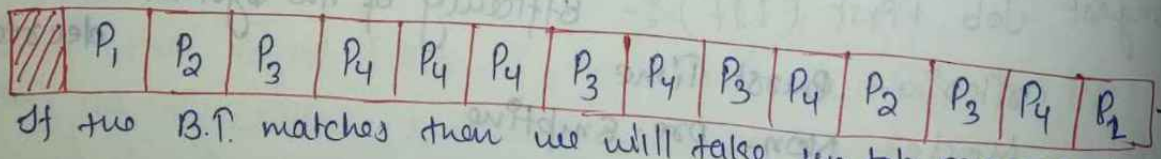
Qu:-

P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
1	1	2	18	17	15
2	2	4	19	17	13
3	3	6	20	17	11
4	4	8	21	17	9

Avg T.A.T = 17.

Avg W.T. = $\frac{48}{4}$ = 12.

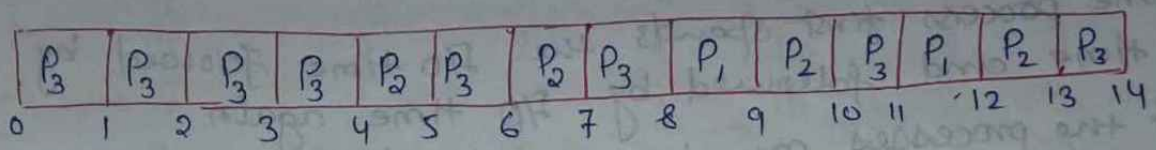




If two B.T. matches then we will take the top processor.

Ques:-

P.No.	A.T.	B.T.	CT	T.A.T.	W.T.
1	0	2	12	12	10
2	0	4	13	13	9
3	0	8	14	14	6



Avg. T.A.T = 13, Avg. W.T. = $\frac{25}{3}$ = 8.3

Ques:-

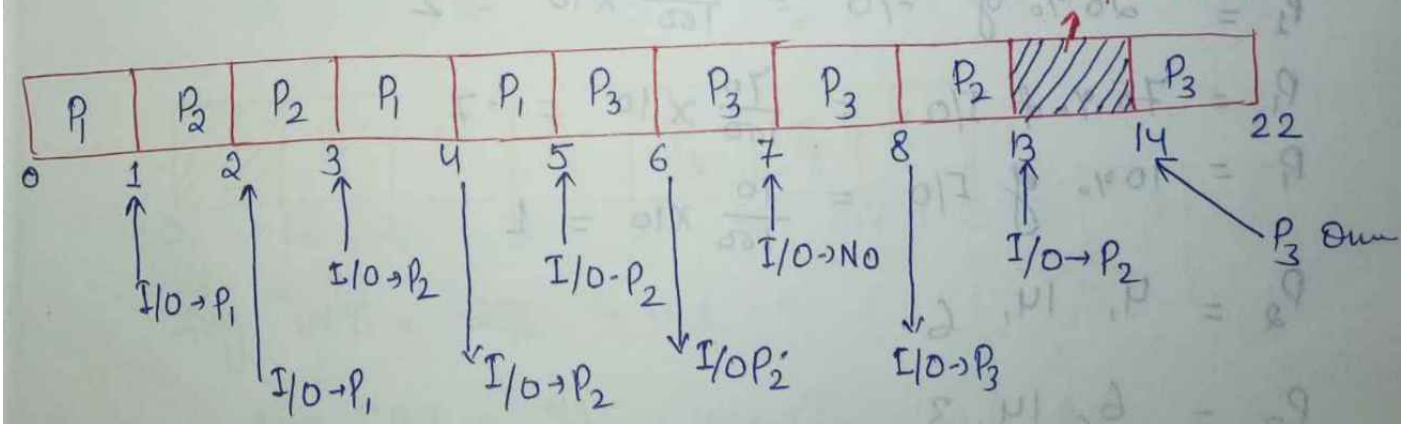
P.No.	A.T.	CPU Time	I/O Time	CPU Time
1	0	1	2	2
2	1	2	4	5
3	2	3	6	8

What is the completion time of process P_1, P_2, P_3 using SRTF Algo.

Note→ i) The process first spends the cpu time followed by I/O time & followed by cpu time.

ii) I/O of the process can be overlapped as much as possible.

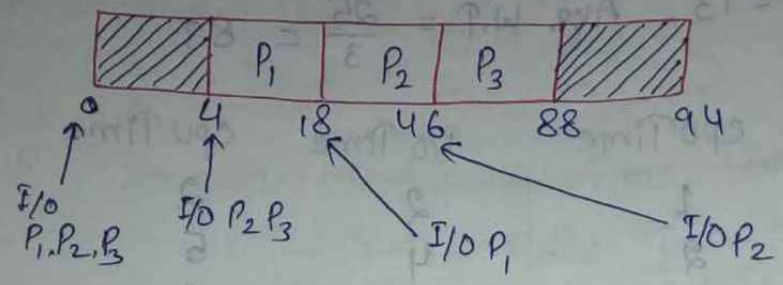
* The white time in the Grant Chart is the CPU Time
* if we start I/O time first than cpu will be idle.



Qu:-

P.No.	A.T.	I/O Time	CPU Time	I/O Time
1	0	4	14	2
2	0	8	28	4
3	0	12	42	6

Note:- The process first spends the I/O time followed by CPU time and followed by I/O time again. I/O of the processes can be overlapped.



$P_1 = 18+2 = 20$

$P_2 = 46+4 = 50$

$P_3 = 88+6 = 94$

Qu:-

P.No.	A.T.	I/O	CPU	I/O
P_1	0	2	7	1
P_2	0	4	14	2
P_3	0	6	21	3

P_1 = 20% of I/O = $\frac{20}{100} \times 10 = 2$

P_1 = 70% of I/O = $\frac{70}{100} \times 10 = 7$

P_1 = 10% of I/O = $\frac{10}{100} \times 10 = 1$

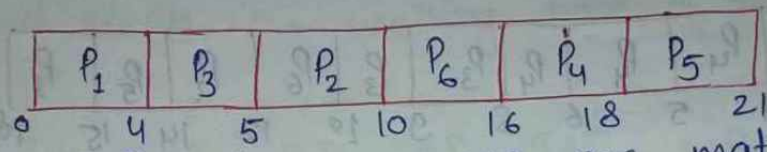
P_2 = 4, 14, 6

P_3 = 6, 14, 3

* Priority Based Scheduling :- criteria → priority
mode → Non-preemptive

Qus:-

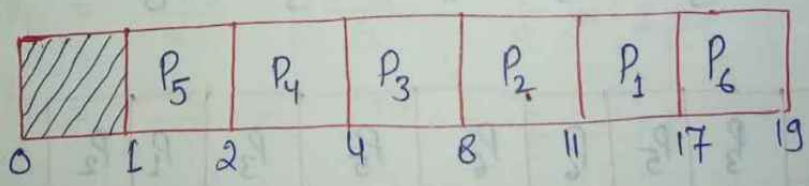
	priority	P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
	4	1	0	4	4	4	0
	5	2	1	5	10	9	4
High →	7	3	2	1	15	3	4
	2	4	3	2	18	15	13
Low →	1	5	4	3	21	17	14
	6	6	5	6	16	11	5



If two priority of the process are matching then schedule the process which has lowest time.arrive

Qu.

priority	P.No.	A.T.	B.T.	CT.	T.A.T.	W.T.
4	1	4	6	17	13	7
5	2	6	3	11	5	2
6	3	3	4	8	5	1
6	4	2	2	4	2	0
7	5	1	1	2	1	0
3	6	2	2	19	17	15



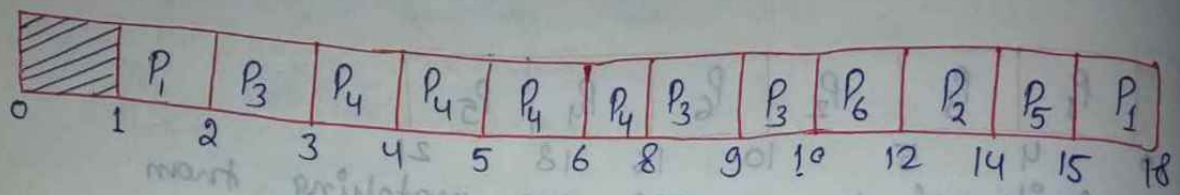
Avg. W.T. = 25/6 = 4.1

Avg T.A.T. = $\frac{43}{6}$ = 7.1

* priority Based scheduling - criteria - priority
mode : preemptive

Q

priority	P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
low → 4	1	1	4	18	17	13
5	2	2	2	14	12	10
7	3	2	3	10	8	5
High → 8	4	3	5	8	5	0
5	5	3	1	15	12	11
6	6	4	2	12	8	6

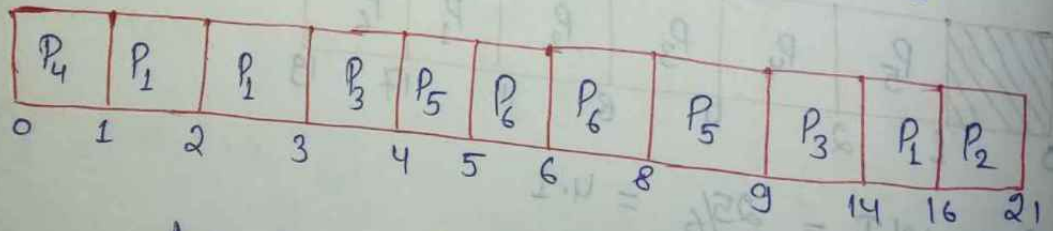


Avg T.AT = 62/6 = 10.3

Avg W.T. = 45/6 = 7.5

Ques

priority	P.No.	A.T.	B.T.	C.T.	T.A.T.	W.T.
5	1	1	4	16	15	11
2	2	2	5	21	19	14
6	3	3	6	14	11	5
4	4	0	1	1	1	0
7	5	4	2	9	5	3
8	6	5	3	8	3	0



Avg T.A.T = 54/6 = 9

Avg W.T. = 33/6 = 5.5

Highest Response time next (HRTN):-

Criteria = Response Ratio

mode = Non-pre emptive

Response Ratio = $W+S/S$

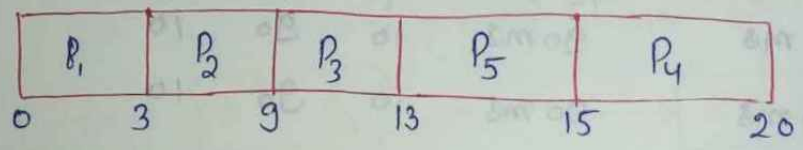
w → waiting time

S → Service time / Burst time

The HR ratio next scheduling algorithm favours the shorter job and limits the waiting time of longer jobs.

Ques →

P.No.	A.T.	B.T.	C.T.	TAT.	W.T.
1	0	3	3	3	0
2	2	6	9	7	1
3	4	4	13	9	5
4	6	5	20	14	9
5	8	2	15	17	15



Avg. T. AT = $\frac{43}{5} = 8.6$

Avg W.T. = $\frac{23}{5} = 4.6$

$R.P_3 = \frac{W+S}{S} = \frac{5+4}{4} = \frac{9}{4} = 2.25$ (9 - AT)

$R.P_4 = \frac{w+s}{s} = \frac{3+5}{5} = \frac{8}{5} = 1.6$

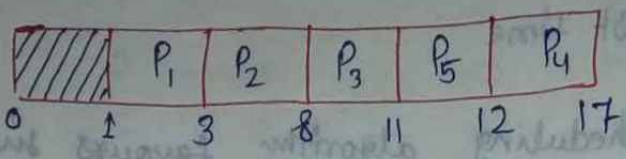
$R.P_5 = \frac{w+s}{s} = \frac{1+2}{2} = 1.5$

$R.P_4 = \frac{7+5}{5} = 2.4$

$R.P_5 = \frac{5.2}{5} = \frac{7}{5} = 3.7$

Q.W.-

P No.	A.T.	B.T.	C.T.	TAT	WT.
1	1	2	3	2	0
2	3	5	8	5	0
3	4	3	11	7	4
4	6	5	12	6	1
5	8	1	17	9	8



$R.P_3 = \frac{4+3}{3} = 7/3 = 2.3$

$R.P_4 = \frac{2+5}{5} = 7/5 = 1.4$

$R.P_5 = \frac{0+1}{1} = 1$

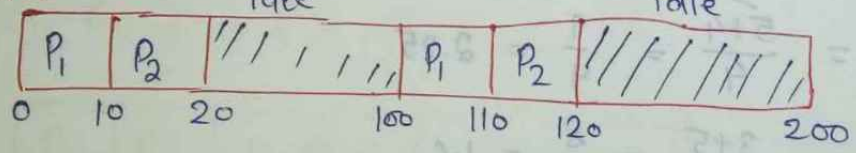
Avg. T.A.T = $29/5 = 5.8$

Avg. W.T. = $13/5 = 2.6$

Q.W:-

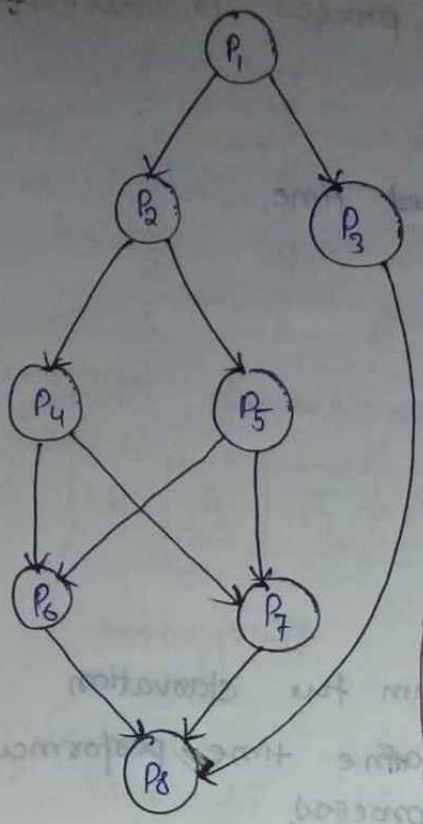
P.No.	A.T.	CPU Burst	I/O Burst	CPU	I/O	CPU
P_1	0	10 ms	90 ms	10	90	10
P_2	0	10 ms	90 ms	10	90	10

1) FCFS



- P

Que:-

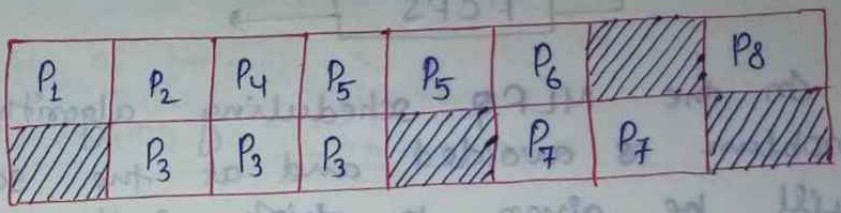


Consider the dependencies graph b/w the process at what time all the process complete their execution by using 2-cpu processor.

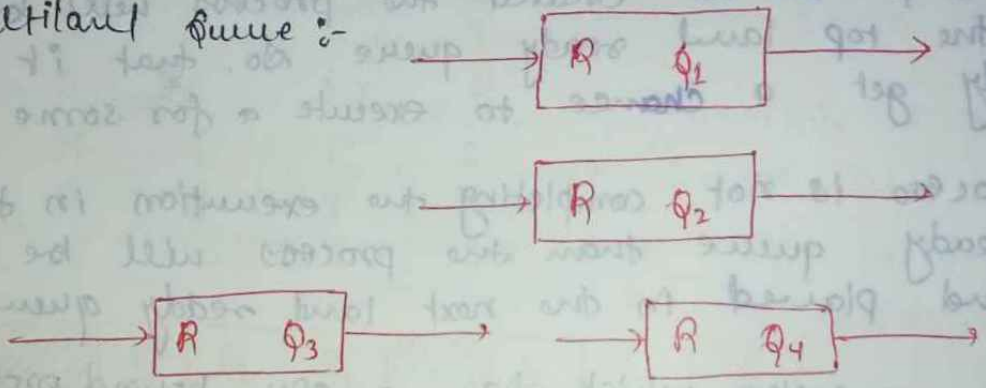
Note - i) Use non-preemptive mode.

ii) one process can not share the 2 cpu at the same time.

Assume A.T. = 0



Multilevel Queue :-



i) Depending on the priority of the process in which perticular ready queue, the process has to be place will be decided.

ii) The high priority process will be placed in the top level ready queue and low priority process will be placed in the bothom level ready queue.

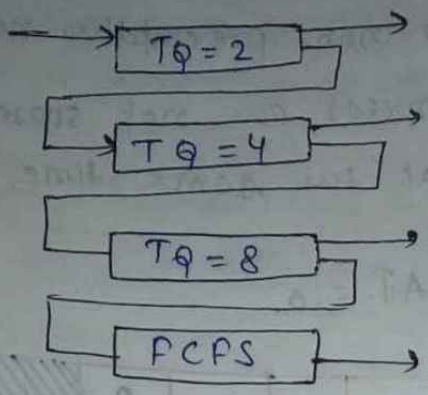
iii) Only after completion of the process from the top level ready queue the further level redy queue process will be scheduled.

iv) If their is the strategy followed than the process which are placed in the bothom level ready queue will suffer from starvation.

Starvation:- The infinite waiting of a process is called as starvation.

* Multilevel feedback Queue:-

Assumption- High process required as burst time.

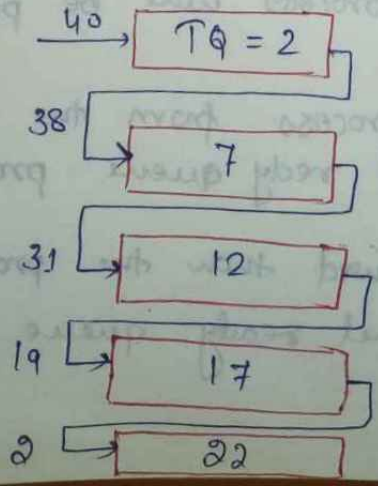


i) In the MLFQ scheduling algorithm the starvation problem is avoided and at the same time performance will be given to high priority process.

ii) Whenever a process is created the process will be placed in the top level ready queue. So. that it will definitly get a chance to execute a for some time (T.Q)

iii) If the process is not completing the execution in the top level ready queue than the process will be interrupted and placed in the next level ready queue.

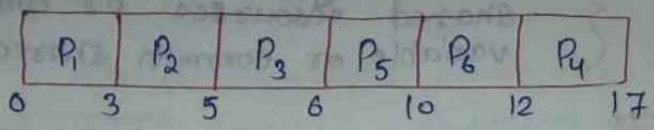
Que:- Consider a system which has a cpu bound process with the burst time of 40 sec. The multilevel feedback queue scheduling algorithm is used and queue time quantom 2 sec. and in each level it incremented by 5 sec. then how many time the process will be interrupted and on which queue the process will terminate the execution.



Ques:-

P.No.	A.T.	B.T.
1	0	3
2	1	2
3	2	1
4	4	5
5	3	4
6	3	2

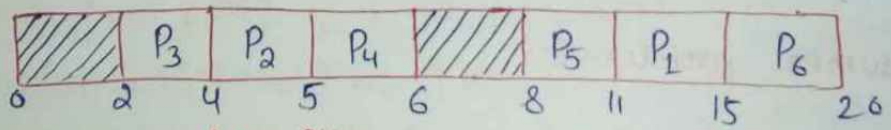
throughput of the system using FCFS scheduling.



$$\text{throughput} = \frac{\text{No. of process}}{\max(C.T.) - \min(A.T.)}$$

Ques:-

P.No.	A.T.	B.T.	C.T.
1	9	1	15
2	3	1	5
3	2	2	4
4	4	1	6
5	8	3	11
6	10	5	20



Algorithm	Starvation
FCFS	No
NP-SJF	yes
SRTF	yes
R.R.	No.
NP-PJF	yes
LRTF	No
NP. priority	yes
pre-priority	yes
HR.R.N.	No
MLQ	yes
MLFBQ	No.

2. Synchronization:- The process with respect to synchronization are of two types.

i) Co-operative process
ii) Independent process.

→ The execution of one process affects as affected by other process than theses processes are said to be co-operative process. Otherwise they are said to be independent process.

when they have common shared resources or common variable or common shared

process

Co-operative
int x = 5

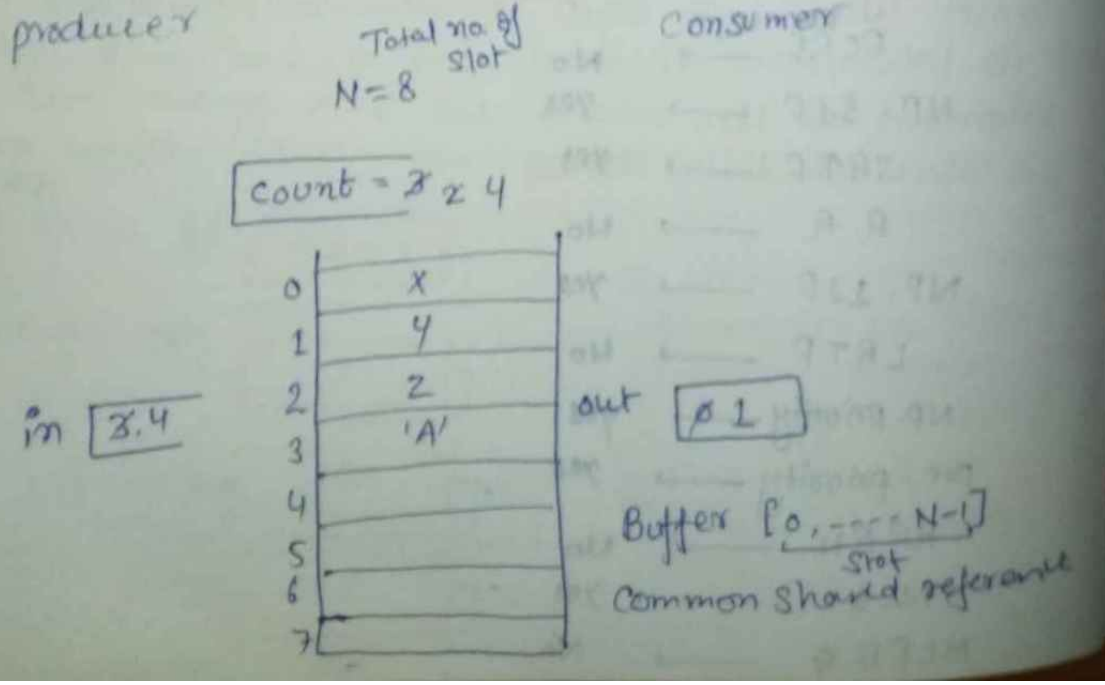
Independent
$P_1 \,||\, P_2$

3. Steps of synchronization-

i) The problem arises not having synchronization b/w the process.
ii) Condition to be followed to achive the synchronization
iii) The solution will include (wrong & right)

Step 1 → The problem arises not having synchronization b/w two process.

The producer - consumer problem-



```
int count = 0;
void producer (void)
{
  int I temp;
  while (true)
  {
produceer - item(i temp)
```

① out → In a variable used by consumer to identify from where it has to consume the item.

In → which slot it has to be placed then item.

② In - Is a variable used by the producer to identify the next empty place/slot in the buffer at any point of time.

```
int count = 0; → global variable
void (producer (void)
                ↳ not passing parameter
{
  int itemp;
  while (true)  ↳ infinite count
  {
produce - item(i temp);
while (count == N);   → Buffer is full count is true the process will stop.
Buffer [in] = I temp;
in = (in+1) mod N;
count = count+1;
```

while it is false then execute the prog

→ Request for producer

```
I : load Rp, m[count)
II : Increm Rp
III : Store m(count), Rp
```

```
void consumer (void)
{ int I temp;
  while (TRUE);
{
false then execute  while (count == 0);   → Buffer is empty cons. is not allow
  Item = Buffer (out);
  out = (out+1) mod N;
  count = count - 1;
produce - item (item c);
}
}
```

```
I1 : load Rc, m[count]
I2 : Decrest Rc
I3 : Store m(count), Rc
```

③ Count :- count is a variable used by bothe producer & consumer to identify the no. of item present in the buffer at any point of time.

④ Shared Resourse :- 1) count variable
2) Buffer

Conditions → i) when the buffer is full the producer is not allowed to produced the item for the buffer
ii) when the buffer is empty the consumer is not allowed to consumer the item from the buffer

Analysis →

producer	consumer
item P = 'A'	item C = X
(3+1) mod 8	(0+1) mod 8
4 mod 8 = 4	1
P ⇒ f_1 , R_P = 3 4	C :- f_1 R_C - 3 / 2
P → f_2 , R_P = 4	C :- f_2 R_C - 2
	C :- f_3 — 2

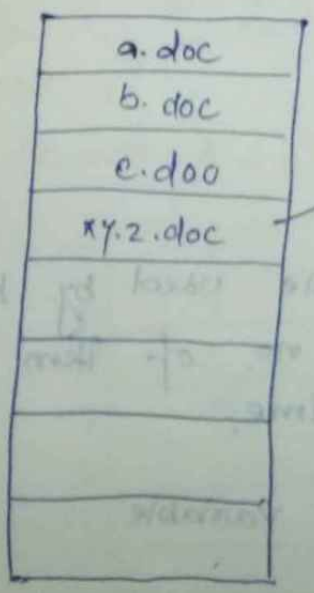
Assuming { Getting preemption that after 2nd pt

P → f_3 : updated in the count variable

1 problem
Inconsistency : { Beoz of Not synchronize }

Statement :- The producer & consumer are not properly synchronized while sharing the common variable count there it is leading to inconsistency

The printer spooler Domain →



Spooler Direction (S.D)

Enter file Register process register [R3 = R0]

I :- load Ri, M[in]

II :- Store SD[Ri], 'f-N'

III :- Increase

IV :- Store in [in], Ri

Every printer has spooler directory to store the document

[filename of a process in SD (spooler Direction)]

IN :- Is a variable used by the printer to identify from where it has to print the document out [o].

Shared Resources :- i) Spooler Directory
ii) 'IN' variable [doubt]

Analysis :- The process is executing the first

fN = 3 $P_1 \rightarrow I$ R_1 [3]

RQ = [P_1 | P_2] $P_1 \rightarrow II$ R_1 [3 4]

xyz.doc → PQR.doc $P_1 \rightarrow III$

P_1 is getting preempted

R_2 [3] $P_2 \Rightarrow I$ [in = 3] not updated

$P_2 \rightarrow II$ (overwritten by the process) P_2 i.e. PQR.doc

I problem is loss of data :- The process are not properly synchronized while sharing the common variable in. Hence it is leading to loss of data.

II problem (Deadlocks) :- If the process are not properly synchronize while sharing the common variable. Resources is also possible for deadlock.

(1) Definitions:-

i) Critical section :- The portion of programme text where the shared Resources or the shared variable will be placed.

Eg.- count = count + 1 : [By produce & consumer problem]

ii) Non-Critical section :- The problem of programme text where the Independent code of the process will be placed.

Eg. In, out :. [By producer & consumer] variable.

iii) Race Condition :- The final output of any variable depends on the execution sequence of the processes. This type of condition is called as race condition.

Eg.

count = 4	count = 2
P → I	C → I
P → I	C → I
C → I	P → I
C → I	P → I
C → I	P → I
P → I	C → I

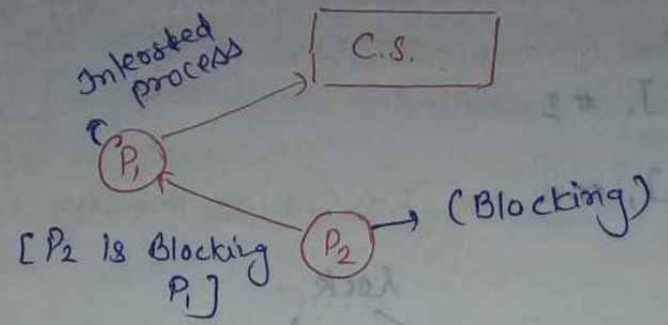
code of producer & consumer code

Step 2:- The condition to be followed to achive the synchronization

1) Mutual Exclusion : (M.E.) → i) No two process may be simultaneously present inside the C.S. at any point of time.

ii) Only one process is allowed into the C.S. at any point of time.

2) progress :- No process running outside the C.S. should block the other interested process from entering into C.S. when C.S. is free.



3) Boonded Waiting :- i) No process should have to wait for ever to enter into C.S.

ii) there should be a boond on getting chance to enter into a C.S.

iii) If the boonded waiting is not satisfied then it is possible for starvation.

4) No Assumption related to hardware and the processor speed.

Step II - solutions - are divided into 4 type -

I - Software Type -

a) Lock variable

b) Strick alteration or Deadlock algo.

c) peterson's Algorithm.

II - Hardware Type

a) Test and set lock (TSL) Instruction set.

III O.S. Type -

a) Counting Semaphore

b) Binary Semaphore

IV. programming Language compiler support type -

a) Monitors.

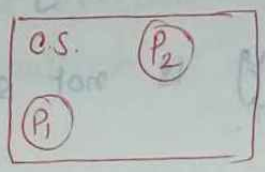
I- Software Type→

a) Lock variable- , Respective process Register [$P_i = R_i$]

I: load Ri, M[Lock]
II: Comp Ri # 0 → To check C.S. is true or not
III: JN2 to step (1) → if non-zero then busy
IV: Store m [lock], #1
V: C.S.
VI: Store m[lock], #0 the process is leaving the C.S
V'

Lock
0 → CS is free
1 → CS is busy

Analysis- Initial value of lock = 0, & 1



R.Q. → [P1 | P2]

lock value

$P_1 \rightarrow I$
Comp. ← $P_1 \rightarrow II$
$P_1 \rightarrow III$

R [0]

After executes the 3rd Inst^n the process is getting preempted. lock value not updated

$P_2 \rightarrow I$
$P_2 \rightarrow II$
$P_2 \rightarrow III$
$P_2 \rightarrow IV$
$P_2 \rightarrow V$ — after 4th Inst^n lock value is updated.

$P_1 \rightarrow IV$
$P_1 \rightarrow V$

R_2 [0]

Mutual Exclusion is not satisfied.

Statement:- We have proved that both the process are simultaneously present inside the C.S. at the same time. Hence the mutual exclusion is not satisfied and solution is bound to be incorrect.

Strict Alteration or Deckers Algorithm:-

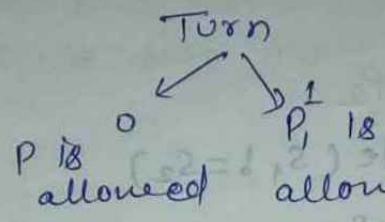
(process takes this to enter into C.S.)

process P_0 code :-

```
while (True)        // Infinite condition
{ Non - C.S. ();
while ((true) != 0);   // If true inside the while loop
C.S.                   // if fals enter into C.S.
turn = 1;
}
```

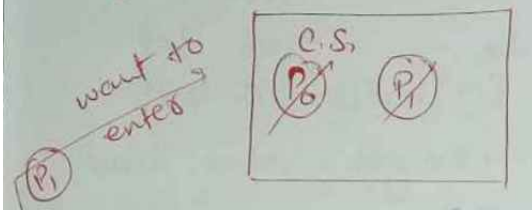
process P_1 code -

```
while (true);(
{ Non - C.S. ();   // independant code
while (true != 0);   // If it true the process will stop here only
C.S.
turn = 0;
}
```



Soln → works only for two process.

Analysis :- mutual value Turn = ϕ / 0



i) If it is exe. from C.S, then turn = 1

Mutual Exclusion is satisfied

for progress ⇒ Unless and until the P_0 is not making turn = 1 then P_1 is not allowed

ii) P_1 is leaving the C.S. & turn = 0 (No body is in C.S.)

2) The progress is not satisfied.

Second Definition of progress → which process will go next to C.S is decided by only those process who want to go into C.S in the decision the process is the remainder section and the process which is not interested to go into C.S. should not take participation.

* The code after critical section is called as remainder section.

Q→

P_1

```
while (S1 == S2)
C.S.
S1 = S2
```

P_2

```
while (S1 != S2)
C.S.
S2 = not (S1)
```

1 → 0 (Ko), 0 → 1 (Ko)

$S_1 = 0$, $S_2 = 0$ | P_1 → the value of S_2 assigned to S_1

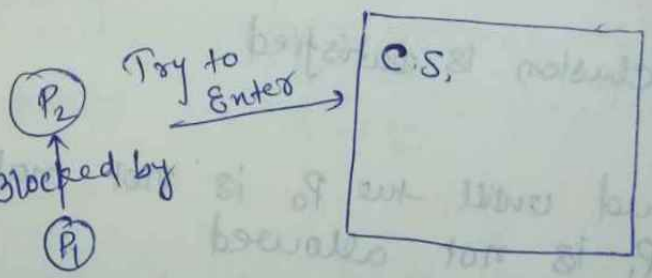
$S_1 = 0$ $S_2 = 1$

$S_1 = 1$ $S_2 = 0$

$S_1 = 1$ $S_2 = 1$

} $S_1 = S_2$

$S_1 = 0, S_2 = 0$



Important points :-

i) The preemption is temporary stop the process will come back and continous the remaining execution.

ii) If there is any chance of the sol^n. Becoming wrong by taking the preemption then only consider the preemption.

iii) If any sol^n is having the deadlock the progress is not satisfied.

If deadlock is there then progress is not satisfied.

* Peterson's Algorithm (2 process solution)

```
# define N 2;
# define True 1
# define false 0
int turn   shared variable used by P0 & P1
int turn
int interested[N]   → 2
void enter - Region ( int processes)
{
1. int other;
2. other = 1 - processes;
3. Interseted [process] = True;
4. turn = process;
5. while (turn == process && interested [other] == True);
}
C.S.
void leave - Region ( int process)
{
interested [process] = false;
}
```

* Turn is a shared variable which is used by both the process P_0 & P_1

process → 0 process P_0, 1 process P_1

interested[2] interested 0 & 1

Initially → interested[0] = ~~false~~ True

interested[1] = ~~false~~ True

2.

P_0 – other P_1
P_1 – other P_0

3. Trying to enter into C.S. then making it is true.

4. turn = process
   turn = 0
   turn = 0

Analysis –

P_0

int process = 0

int other is local variable

other = 1 – process
= 1 – 0

other = 1
T

while (0 == 0 && false == true);
f

Enter into C.S.

P_1

process = 1

other = 1 – process
= 1 – 1 = 0

other = 0

interested[1] = True

while (1 == 1 && True == True)
T T

→ Not enter into C.S

for P_1 – first

interested[0] = ~~false~~ True

interested[1] = false True ~~false~~

Turn ≠ 0

P_0

process = 0

other = 1 – process
= 1 – 0
= 1

while (0 == 0 && True == True);
T T

Not Enter to C.S.

P_1

process = 1

other = 1 – process
= 1 – 1 = 0

other = 0

while (1 == 1 && false == true)
T f

Enter to C.S.

1. Mutual Exclusion is satisfied.
2. progress. progress is satisfied.

turn = 1 process = 0

Interested [0] = false True
Interested [1] = false True

P_0 | P_1

other = 1
preempt occures now P_1 come

other = 0

C.S.
P_1 → Becoz P_0 is also a interested process.

3) Bounded waiting is also satisfied.

Q. Assume that both the process P_0 & P_1 are trying to enter into critical section at the same time that which process will go into C.S. 1st

a) The process which execute the statement 2 first
b) The process which execute the statement 3 first
c) The process which execute the statemat 4 first
d) we cant say.

Test and set lock
TSL Instruction Set :-

TSL Register flag :- copies the current value of flag into register and stores the value of '1' into flag in a single atomic cycle without only preemtion

Entry Section :-

I. TSL, Ri, m[flag] ← [i = respective process register]
II cmp Ri, #0 → to identify C.S. is free or not
III JN2 to step (I) → is busy then go to step I
IV C.S.
V Store m[flag], #0

↓ After releasing the C.S. make the

Unless & until it is not execute the stat. no. 5 than it is consider to be in C.S.

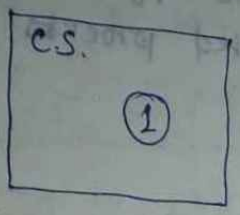
flag

0 → C.S. is free

1 → C.S. is busy

→ C.S. is common there

Analysis → flag = ∅ 1



Mutual Exclusion is satisfied.

R.Q P_1 | P_2

Instruction

R_1 [0]

$P_1 \rightarrow$ I

$P_1 \rightarrow$ II

$P_1 \rightarrow$ III

$P_1 \rightarrow$ IV

$P_2 \rightarrow$ I

$P_2 \rightarrow$ II

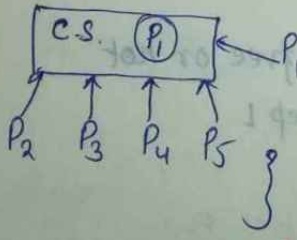
$P_2 \rightarrow$ III

R_2 [1]

II progress → progress is satisfied.

III Bounded waiting → Not satisfied.

flag = ∅ 1



$P_1 \rightarrow$ I

$P_1 \rightarrow$ II

$P_1 \rightarrow$ III

$P_1 \rightarrow$ IV

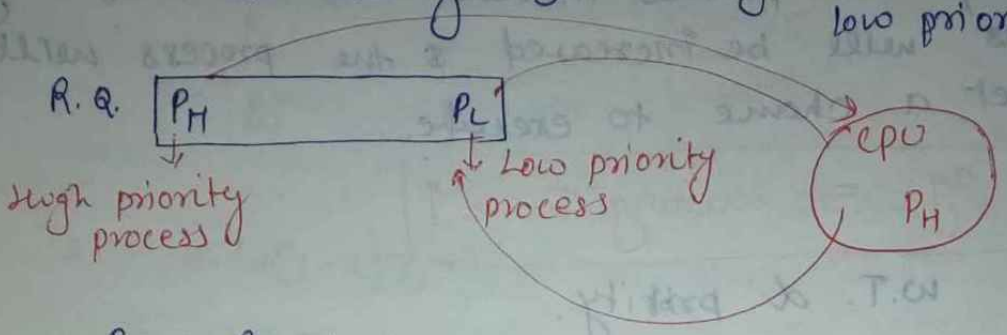
R_1 [0]

There is no garantee that next P_2 will enter may be P_4 or P_5 will be enter.

Solution-	M.E.	Progress	Bounded Waiting
Lock variables	X	✓	X
Strict Alteration	✓	X	✓
peterson Algo.	✓	✓	✓
TSL Inst' set	✓	✓	X

priority Inversion problem:-

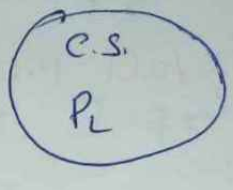
↳ one waiting high priority & one is having low priority.



P_H → Running

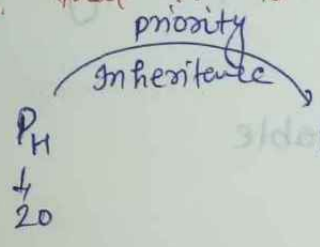
P_L → Reading

Live Lock:-



"High priority process should wait until useless P_L should ---- the C.S. that it is allow in the CPU.

Sol" →



* Now the priority is follow some than execution with the help of arrival time.

CPU Bound. process which require more amount of time

R.R. ⇒ T.Q = 3 assume B.T. = 40

FCFS will forever CPU Bound process and it is a non-preemitive Sheduling.

$n = 3$ $(P_1\ P_2\ P_3)$

$n! = 6$

Notes - Aging is the concept is generally avoid the problem of starvation. or

If the waiting time of the process increase then the priority of the process will be increased.

If the age of the process increase then the priority of the process will be increased & the process will defenatly get a chance to execute.

$$\boxed{\uparrow \text{age} = \text{waiting Time} \uparrow}$$

W.T. $\propto$ priority.

→ T.Q. = Q - usefull time { Execute the int" of the process.

C.S. = T

I/O = R

cpu efficency = ? $\dfrac{\text{usefull time speed by the cpu}}{\text{Total time spend by cpu}}$

$\dfrac{Q}{Q+R} \Rightarrow \dfrac{Q}{Q+T+R}$ → unpriolictable

I/O → Alone
→ Along with cpu
→ Multiple process can perform I/O at the same time

* Notes - In the shortest Job FCFS algo the B.T. of the process will be expected or predicted by using the below formula.

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

T_{n+1} = Next Expected B.T.

T_n = previous expected Burst Time

T_n = previous actual Burst Time.

α = is parameter which control the relation of recent and past history.

$$0 \leq \alpha \leq 1$$

process	B.T.	
P_1	5	
P_2	8	$T_1 = 10$
P_3	3	$\alpha = .5$
P_4	5	

$T_2 = \alpha.T_1 + (1-\alpha)T_1$

$= .5 \times 5 + (1-\alpha)10 = 7.5$

$T_3 = .5 \times 8 + .5 \times 7.5 = 7.75$

$T_4 = .5 \times 3 + .5 \times 7.75 = 5.375$

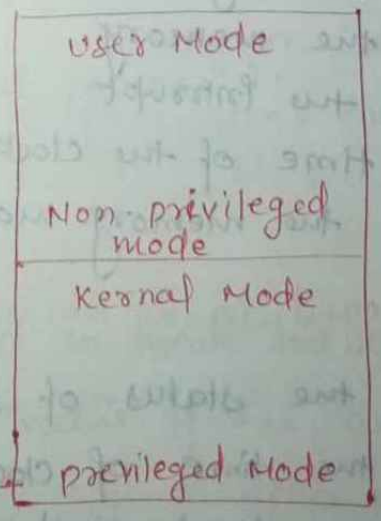
$T_5 = .5 \times 5 + .5 \times 5.375 = 5.1875$

- Dual Mode operation -

Mode bit

0 — Kernal mode

1 — User Mode



Dual mode

Errent User :- False, Hacker, Corrupted User.

1. In the hardware lavel the Inst^{n} are executed in the two different mode i) User ii) Kernal
2. The dual mode operation is used in order to provide the protection & security to the user programm and and also to the O.S. from the Errent User.
3. Some Inst^{n} are executed in the user mode.
& Some Inst^{n} are executed in the Kernal mode.
4. Generally the privileged Inst^{n} are executed in the Kernal mode & non privileged inst^{n} are executed in the user mode.
5. The mode bit is use to identify in which perticular mode the present Inst^{n} in executing.
6. In the boot time the system alloways start in the kernal moder.
7. The operating system always run in the kernal mode

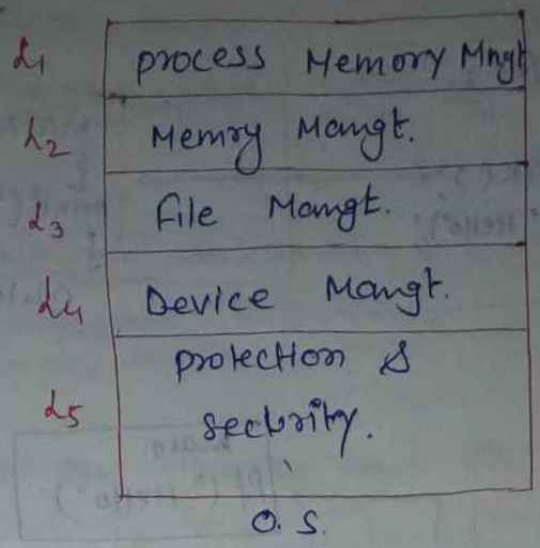
Ex. of privileged Inst^{n} :-

1. I/O operation
2. Context Switching
3. Clearing the memory
4. Disabling the Interrupt.
5. Set the time of the clock.
6. changing the memory map. (location change)

→ Non-privileged in system.

1. Reading the status of the processor.
2. Reading the time of clock.
3. Sending the final print output to the printer.

• Layered Approach :-



{ Modularity :- Easy to change / easy to update }

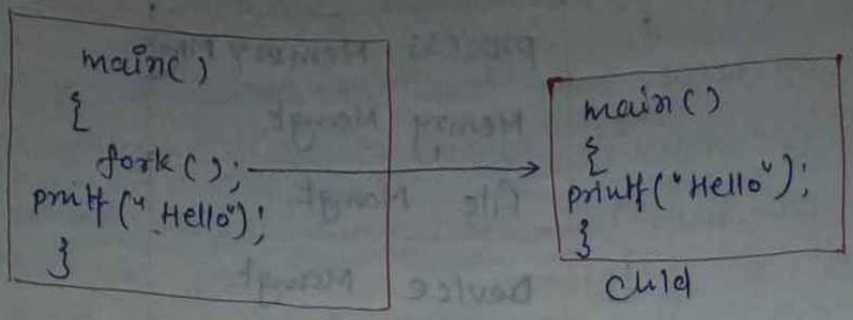
Fork :-

```
main()
{ int pid ;
  pid = fork();
if ( pid <0 ) {
printf ("force failed");
} else if ( pid == 0)
  { printf (" child process");
  }
else {
      printf (" parent process");
  }}
```

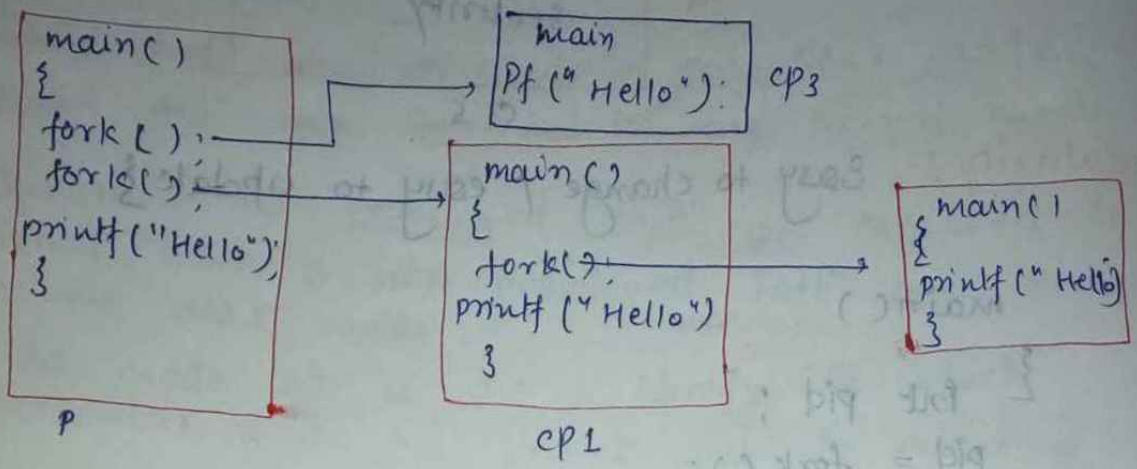
Note :-

1) Fork is the system call used to create the child process.
2) The fork return a negative value if the child process creation is unsuccessfull.
3. The fork return the value zero to the newly created child pro.
4. The fork return (+ve) to the parent process.
5. when the child process is created by using fork system call the new memory location will be allocated the child process.
6. Both parent and child process will have the same virtual address. But the physical address is different.

Ex-



Ex-



→ If the progress is having n fork calls than it will create $2^n - 1$ child process.

* Semaphore (Technique to implement synchronization)

The semaphore is an integer variable which is used by the various process in a mutual Exclusion manner to achive synchronization.

→ The Improper usage of semaphore will also give the Improper result.

Categorized into two types:-

1) Counting semphore - various form
2) Binary semaphore → 0 or 1

→ The two different operation will be performed on the semaphore variable.

1) Down() or wait() or P()
2) up() or signal() or V() or Release()

Counting Semaphore -

```
Down (semaphore S)
{
  S.value = S.value - 1;
  if (S.value < 0)
  {
  Block the process and place
  its PCB in the suspended list();
  }
}
```

```
Up (semaphore S)
{
  S.value = S.value - 1;
  if (S.value ≤ 0)
  {
    select a process from the suspended
      list and wake up();
  }
}
```

After performing the down operation if the process is getting suspended than it is called as unsuccessfull down operation.

- If it is unsuccessfull down operation the process will not continue the execution.
- After performing the down operation if the process is not getting suspended than it is called a successfull down operation.

If it is successfull down operation then only process will continue the execution.

$S = +8, 5^{L}$

- If semaphore value = +9 then we can perform 9 successfull down operation.

⇒ The down operation on the counting semaphore is successfull only when the initial value of semaphore is $S \geq 1$

wake up() → continue the execution.

→ There is no unsuccessfull up operation the process performing up operation will definately continue the execution.

→ The up opration is always successfull.

Qu- Consider a system where the critical section value is initialized to S = +17 the various operation like 23P, 18V, 16P, 14V, 1P are performed then what is the final value of the semaphore

a) +7 b) +8 c) +9 d) +10

value = +17 − 23 + 18 − 16 + 14 − 1 = +9

→ Binary semaphore-

```
Down (semaphore s)
{ if (s.value == 1)
    s.value = 0;
else { Block the process and
place its PCB in the
suspended list ();
}
}
```

```
up (semaphore s)
{ if (suspended list()
is empty)
    s.value = 1;
else {
select a process from the
suspended list and wake
up(); } }
```

→ After performing the down operation if the process is not getting suspended that it is called a successfull down operation.

→ If it is successfull down operation then only the process will continue the oper execution.

→ After performing the down operation if the process is getting suspended then it is called as unsuccessfull down operation.

→ If itis unsuccessfull down operation the process will not continue the execution.

* The down operation on the binary semphore is successfull only if the initial value of the semaphore is =1

$S = 1$

- There is no unsuccessfull up operation. The up operation is allways successfull.

- The process performing up operation will definetely continue the execution.

Qw→ Each process P_i ; i = 1 to 9 execute the following code.

```
Repeat
    P(mutex);
    C.S.
    V(mutex);
forever
```

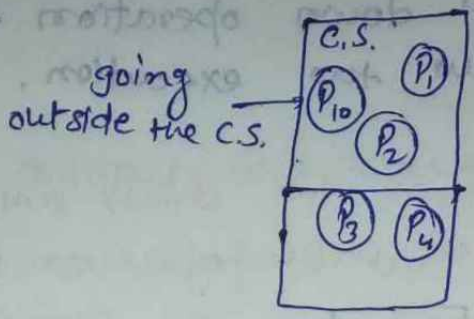
The process 'P_{10}' execute the following code repeat

```
V(mutex);
C.S.
V(mutex);
forever.
```

The initial value of binary semaphor mutex = 1
What is the max. number of process that may present inside the critical section at any point of time.?

a) 2 b) 3 c) 9 d) 10

Sol: Analysis Mutex - 1, 0, 1, 0, 1, 0, 1, 0



Qu → P_i i=1 to 9

```
P(mutex)
C.S.
V(mutex)
```

m = 1, 0, 1, 0, ...

C.S.

P_{10}

```
V(mutex)
C.S.
P(mutex)
```

Max = 3 process count may change in C.S. but Max 3 process