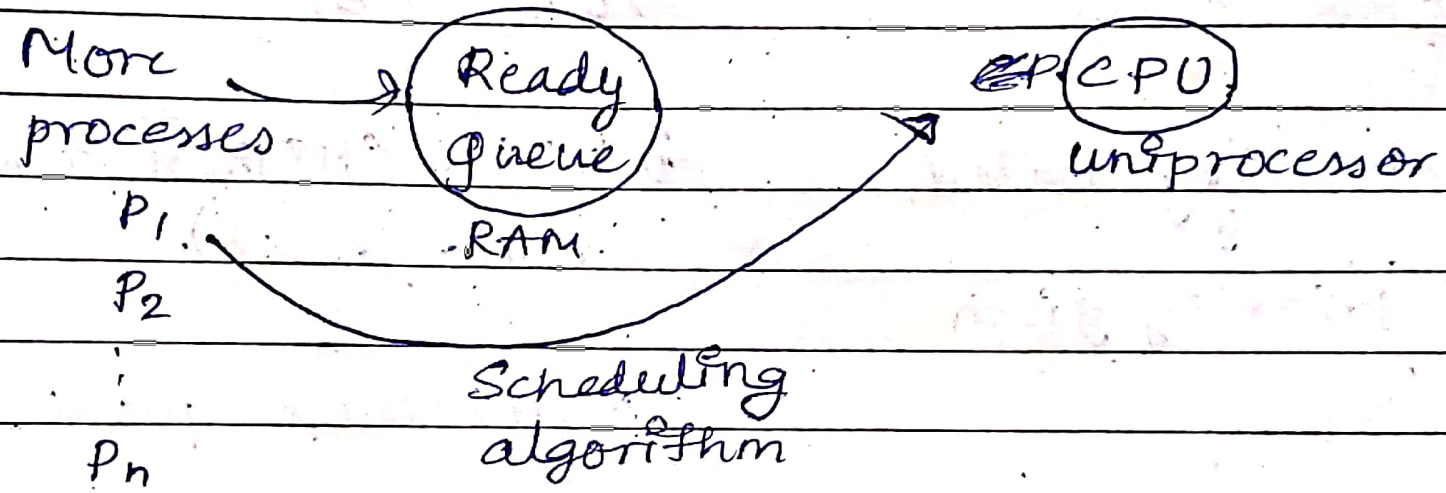
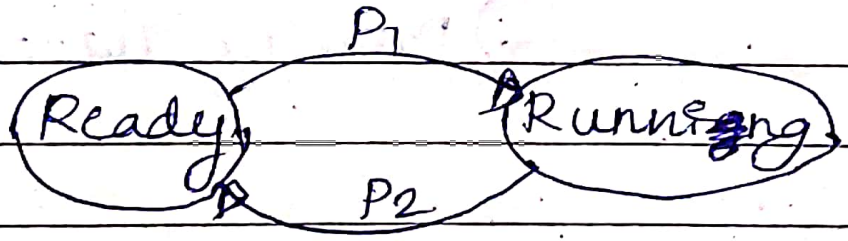
Process Scheduling Algo:

A way of selecting algo from ready queue & putting it on CPU.





Reason:
1) Time quantum
2) Priority of process

Pre-emptive : If a process is taken out from ready queue or RAM & is put in running process (queue) (CPU). Then we can stop the process P_1 & give another process P2 & send P_1 to ready queue. It provides responsiveness.

Non-preemptive : If a process is taken out from ready queue it would be processed till its burst time, after then only new process can be introduced

Pre-emptive	Non-pre emptive
1) SRTF → BT	1) FCFS → AT
2) LRTF → BT	2) SJF → BT
3) RR → TQ	3) LJF → BT
4) Priority based (2 priority given)	4) HRRN (Highest response ratio next)
	5) Multi-level queue
	6) Priority based

(AT) Arrival Time : Time at which process enters the ready queue or stack.

(BT) Burst Time : Time req^d by a process to get executed on CPU

(CT) Completion time : Time at which process completes execution

(TAT) Turn-Around-Time = CT - AT

(WT) Waiting time = TAT - BT

(RT) Response time = (Time at which process gets CPU) - AT

Arrival time → Process exec^n → Completion time
(Point of time) duration (Point of time)

First Come First Serve (FCFS)

→ Executes queued requests & processes in order of their arrival
→ Process which requests the CPU first gets the CPU allocation first
→ Managed by by FIFO queue.

Scanned with OKEN Scanner

Given in question

Take from Gantt

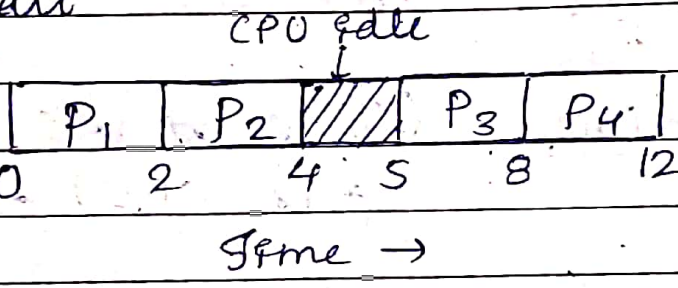
(CT-AT) (TAT-BT)

Got CPU for 1st time	Proc. no.	AT	BT	CT	TAT	WT	RT
0	P_1	0	2	2	2	0	0-0 → 0
2	P_0	1	2	4	3	1	2-1 → 1
5	P_3	5	3	8	3	0	5-5 → 0
8	P_4	6	4	12	6	2	8-6 → 2

Criteria: Arrival time

Mode: Non-pre-emptive

Gantt chart



At $t = 12$ all processes gets executed

Avg. TAT = $\frac{14}{4}$ Avg. WT = $\frac{3}{4}$

Scanned with OKEN Scanner

SJF (Shortest Job First)

→ Process having smallest execution time is chosen for next execution.

→ Can be pre-emptive or non-pre-emptive
→ Reduces WT for other processes
→ Greedy Algo

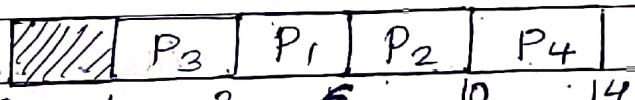
Process	AT	BT	CT	TAT	WT	RT
P1	1	3	6	5	2	2
P2	2	4	10	8	4	4
P3	1	2	3	2	0	0
P4	4	4	14	10	6	6

Starting time

Criteria: Burst time Mode: Non-pre-emptive

Queue: P1 P3 P2 P4

Gannt chart:



→ In time period 0 to 0.99 no process has arrived
→ At time 1, P1 & P3 arrive, P3 has less BT ∴ execute it
→ Now P2 has already arrived at time 2, Compare P1 & P2, P1 has less BT, ∴ P1
→ Now P4 has arrived at time 4, compare P2 & P4 both have equal BT, ∴ Execute P2 first ∵ it has arrived first.

Scanned with OKEN Scanner

When BT is same, check for Arrival time

SRTF (Shortest Remaining time first)

(SJF + Pre-emptive)

Proc. no.	AT	BT	CT	TAT	WT	RT	CPU first time
P_1	0	5	9	9	4	0	0
P_2	1	3	4	3	0	0	1
P_3	2	4	13	11	7	7	9
P_4	4	1	5	1	0	0	4

Criteria: 'BT'

Mode: Pre-emptive

Grannt Chart:

P_1	P_2	P_2	P_2	P_4	P_1	P_3

0 1 2 3 4 5 8 13

Avg. TAT $= \frac{24}{4} = 6$

Avg. WT $= \frac{11}{4} = 2.75$

Avg. RT $= \frac{7}{4} = 1.75$

0 → P_1

1 →

0 → P_1 (5)

1 → P_1, P_2 (4, 3)

2 → P_1(4), P_2(2), P_3(4)

3 → P_1, P_2, P_3(4, 1, 4)

~~4 → P_1, P_2, P_3, P_4(~~

4 → P_1, P_3, P_4 (4, 4, 1)

5 → P_1, P_3 (4, 4)

B: Execute P_1 first

∵ P_4 arrived first

Page No:
Date: / /

Round Robin (RR)

→ Pre-emptive.

→ Once a process is executed for a given time period, it is pre-empted & other processes execute for given time.

Proc. ng.	AT	BT	CT	TAT	WT	RT	CPU first time
P_1	0	5	12	12	7	0	0
P_2	1	4	11	10	6	1	2
P_3	2	2	6	4	2	2	4
P_4	4	1	9	5	4	4	8

Criteria = "Time Quantum"
Mode: "Pre-emptive"

We have to resume the process, not restart them

TQ = 2

2 (Ready ⇄ Running)

Queue:

P_1	P_2	P_3	P_1	P_4	P_2	P_1

Gannt chart:

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0 2 4 6 8 9 11 12

→ At time 0 only P_1 is in queue & ready queue, so run it for tq=2, from 0 to 2

→ In the mean while at t=1 & t=2, P_2 & P_3 resp. arrive, now as P_1 has ran for a tq=2 put it back into ready queue.

Scanned with OKEN Scanner

→ Now run P_2 from 2 to 4, but at 4 P_4 arrives so put it in the ready queue.
→ Now after P_2 has completed its tq put it behind P_4 in ready queue
→ P_3 runs from 4 to 6 & completes itself
→ Now again run P_1 from 6 to 8. & as there's no new process put it at the end of ready ~~queue~~ queue

Mix Burst Time (CPU & I/O both)

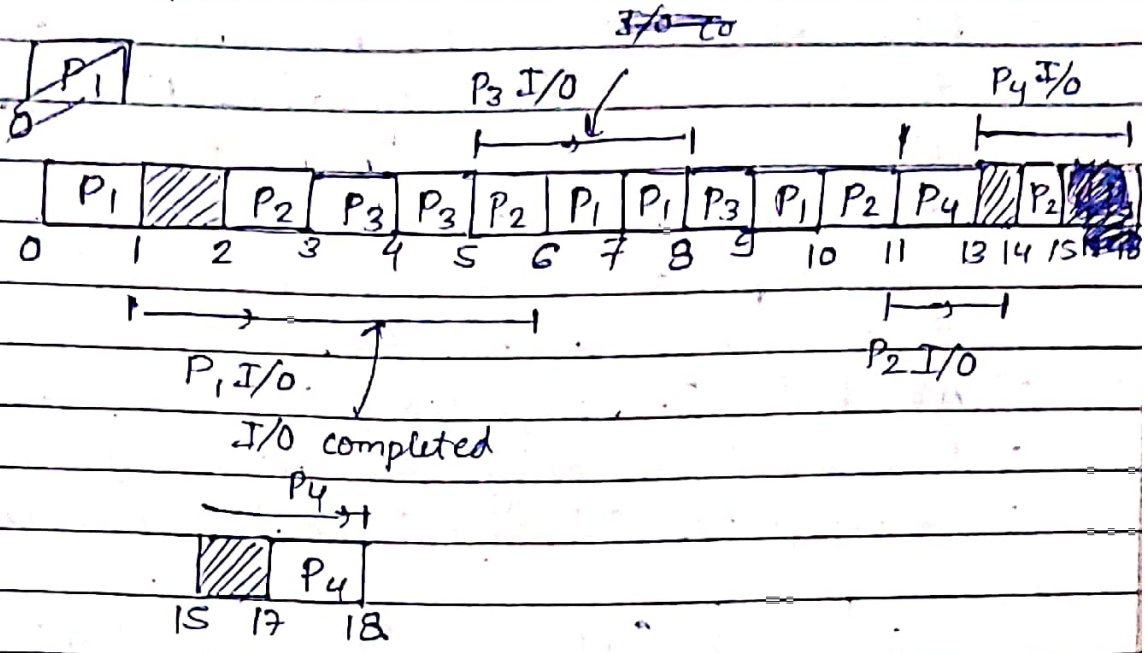
Process	AT	Priority	CPU	I/O	CPU
P_1	0	2	1	5	3
P_2	2	3	3	3	1
P_3	3	1	2	3	1
P_4	3	4	2	4	1

lower the no. higher the priority
Mode: 'Pre-emptive'
Criteria: 'Priority based'

Find CT of Q. P_1, P_2, P_3, P_4

Scanned with OKEN Scanner

Gannt Chart



→ At t=0, P_1 runs for tq=1 & after that starts its I/O process, now CPU of 1 ~~becomes~~ changes from 1 to 0

→ At t=2, P_2 arrives & runs from tq=1 & its CPU changes from 3 to 2

→ Now at t=3, B.P_3 & P_4 have arrived, but P_3 has higher priority than P_4 so run it for tq=1, again run P_3 for tq=1, ∵ it has the highest priority & its CPU changes from 2 to 0

→ Now we have P_2 & P_4 left & P_2 has higher priority so run it for tq=1

→ Now ~~at~~ at t=6 as P_1 has completed I/O it comes back to ready queue, now as out of P_1, P_2, P_4, we see that P_1 has highest priority so it restarts, & CPU changes from 3 to 2 & again run it ~~from~~ for tq=1, CPU → 2 → to 1

→ Now at t=8 as P_3 has completed its I/O it comes back to ready queue, Now as P_3 has the highest priority run it from 8 to 9 & P_3 has completed its CPU

→ Now P_1 has highest priority so it runs from 9 to 10 & completes its CPU

→ At t=11, P_2 completes & its I/O starts

→ Now as P_4 is only left it runs from 11 to 13 & completes its CPU & its I/O starts

	CT
P_1	10
P_2	15
P_3	9
P_4	18

CPU Idle = 4/18

usage = 14/18

0 → P_1

1 → no process

2 → P_2

3 → P_3, P_4
↳ high priority

4 → P_1, P_2, P_3, P_4

5 →

6 → P_1, P_2, P_4

Scanned with OKEN Scanner

Page No:
Date: / /

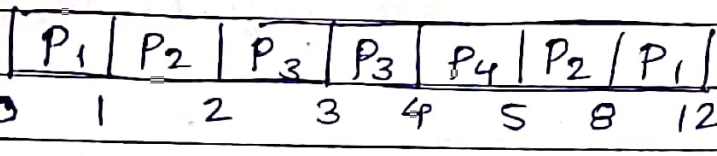
Pre-emptive priority scheduling

The scheduler selects the task to work as per priority

Equal priority: Round Robin or FCFS

Priority	Processing	AT	BT	CT	TAT	WT	RT	CPU first
10	P_1	0	5	12	12	7	0	0
20	P_2	1	4	9	7	3	0	1
30	P_3	2	2	4	2	0	0	2
40	P_4	4	1	5	1	0	0	4

Higher the ng. higher the priority



0 → P_1
1 → P_1, P_2
2 → P_1, P_2, P_3
3 → P_1, P_2, P_3
4 → P_1, P_2, P_3, P_4
5 → P_1, P_2

Scanned with OKEN Scanner

Non-Pre-emptive Priority Scheduling

P.I.D	Priority	A.T	B.T	C.T	T.A.T	W.T	R.T
P_1	2	0	3	3	3	0	0
P_2	6	2	5	18	16	11	13
P_3	3	1	4	7	6	2	5
P_4	5	4	2	13	9	7	11
P_5	7	6	9	27	21	12	18
P_6	4	5	4	11	6	2	7
P_7	10	7	10	37	30	18	27

Mode: "Non-Pre-emptive" Criteria: "Priority"

P_1	P_3	P_6	P_4	P_2	P_5	P_7
0	3	7	11	13	18	27	37

Lower the no. higher is priority

0 → (P_1)

3 → P_2, (P_3)

7 → P_2, P_4, P_5, (P_6), P_7

11 → P_2, (P_4), P_5, P_7

Scanned with OKEN Scanner

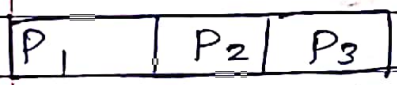
CPU Scheduling

1. CPU Utilization → High
2. TAT → less
3. Response → min.
4. Waiting time → less
5. Throughput → max.
   ↳ no. of o/p per unit time

FCFS (Convoy Effect)

	AT	BT	TAT	WT
P_1	0	50	50	0
P_2	1	1	50	49
P_3	1	2	52	50

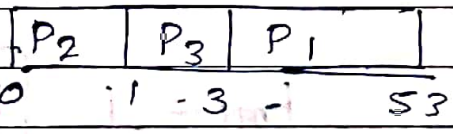
$99 \div 3 = 33$



	AT	BT	TAT	WT
P_1	0	50	50	0
P_2	1	1	50	49
P_3	1	2	52	50

	AT	BT	TAT	WT
P_1	1	50	52	2
P_2	0	1	1	0
P_3	0	2	3	1

$3/3 = 1$



When a process having higher burst time comes first other processes have to wait for a large amount of time, this is called convoy effect.

Predict burst time based on process type & process-id?:

→ Process Type
→ Process Size

$P_{old} = 100KB$ — 5 units
$P_{new} = 102KB$ — ≃ 5 units

As the process size is the same, it would take approx. the same BT.

Burst time also depends on the nature or type of the process.

→ Simple Averaging:

For e.g. 5 processes have finished execution, then we can predict the BT of 6th process by simple averaging method

P_1 P_2 P_3 P_4 P_5

$$\frac{t_1 + t_2 + t_3 + t_4 + t_5}{5}$$

$$P_{n+1} = \frac{1}{n}$$

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

→ Exponential Averaging

$$\boxed{E_{i+1} = \alpha A_i + (1-\alpha) E_i}$$

Scanned with OKEN Scanner

E_{i+1} = Expected time for process $i+1$

A_i = Actual BT of process i

$$0 < \alpha < 1$$

α = Smoothening factor

$\alpha \neq 1$ or 0 in real life

E.g.

$\alpha = 0.5$, $E_1 = 5$

Process	BT
P_1	4
P_2	8
P_3	5
P_4	6

$E_5 \Rightarrow$

$E_5 = \alpha A_4 + (1-\alpha) E_4$

$E_5 = 0.5 \times 6 + (0.5) E_4$

$= 5.81$

$\boxed{E_5 = 5.81}$

$E_4 = \alpha A_3 + (1-\alpha) E_3$

$= 0.5 \times 5 + (0.5) E_3$

$= 5.625$

$E_3 = 0.5 \times 8 + (0.5) E_2$

$= 6.25$

$E_2 = (0.5) \times 4 + 0.5 \times 5$

$= 4.5$

Scanned with OKEN Scanner

PAGE NO.:

SJF having CPU time & I/O time

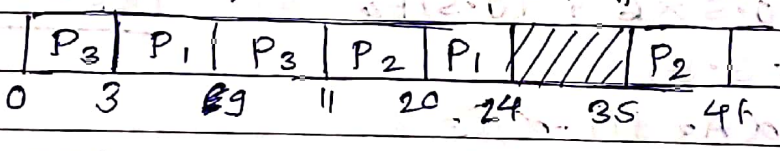
Q1 ~~Then Tr~~ Three processes P_1, P_2, P_3 with process times 20, 30, 10 respectively. Each process uses the first 30% of the process time on CPU, the 50% on I/O & last 20% on CPU: Find avg. WT, TAT, & RT ~~of~~ If system follows SJF scheduling.

	Process time
P_1	20
P_2	30
P_3	10

Process time = CPU time + I/O time

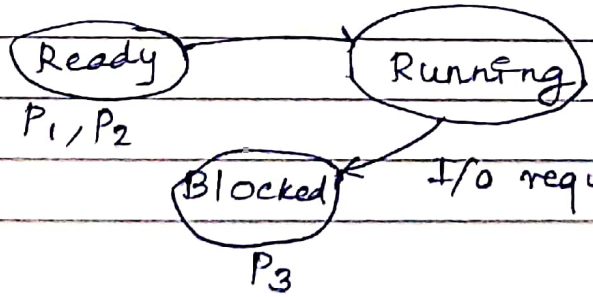
	Process time	AT	CPU BT	I/O BT	CPU time
P_1	20	0	6	10	4
P_2	30	0	9	15	6
P_3	10	0	3	5	2

Choose the process with min. process time first



In SJF,
If the process wants to do some I/O operation then the process can leave the CPU on its own.

P_3 will return at 8



Scanned with OKEN Scanner

P_1 will return at 19

Ready = P_2, P_3

Now we'll allocate the CPU to P_2 ∵ it has less CPU time than P_3

Ready : P_2

P_2 will return at 35

Ready : P_1

Now at t=24 we don't have any process in ready queue & P_2 would come at 35

		TAT (CTAT)	WT (TAT − BT)	RT
P_1				
P_2	P_1	24 − 0 = 24	24 − 10 = 14	3
P_3	P_2	41 − 0 = 41	41 − 15 = 26	11
	P_3	11 − 0 = 11	11 − 5 = 6	0

In BT ~~only~~ don't consider I/O BT.

∴ CPU time for P_1 = 6 + 4 = 10

Avg. TAT = $\frac{76}{3}$ = 25.3

Avg. WT = $\frac{46}{3}$ = 15.3

CPU utilization = $\frac{\text{Expected}}{\text{Actual}} = \frac{10+15+5}{41}$ (See in Gantt chart, & the rightmost point)

Scanned with OKEN Scanner

$= \frac{30}{41} = 0.7317 = 73.17\%$

Don't consider I/O BT in Actual,

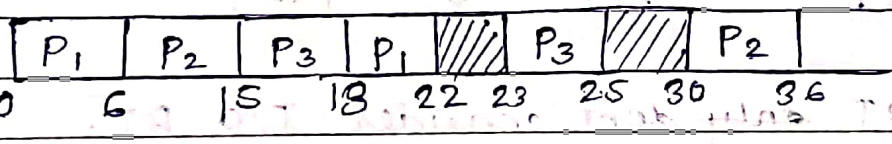
Idle % of CPU = $\frac{\text{Total Idle time}}{\text{Actual}}$

$= \frac{11}{41} = 0.2682$

$= \boxed{26.82\%}$

Same sum with FCFS scheduling

	Process time	AT	CPU BT	I/O BT	CPU BT	CT	TT	WT	RT
P_1	20	0	6	10	4	22	22	12	0
P_2	30	0	9	15	6	36	36	21	6
P_3	10	0	3	5	2	25	25	20	15



P_1 will come back at 16 unit of time

Ready : P_2 & P_3
Blocked : P_1

P_2 will return at 30
Ready : P_3
Blocked : P_1, P_2.

P_3 will return at 23

Scanned with OKEN Scanner

Ready : P_1

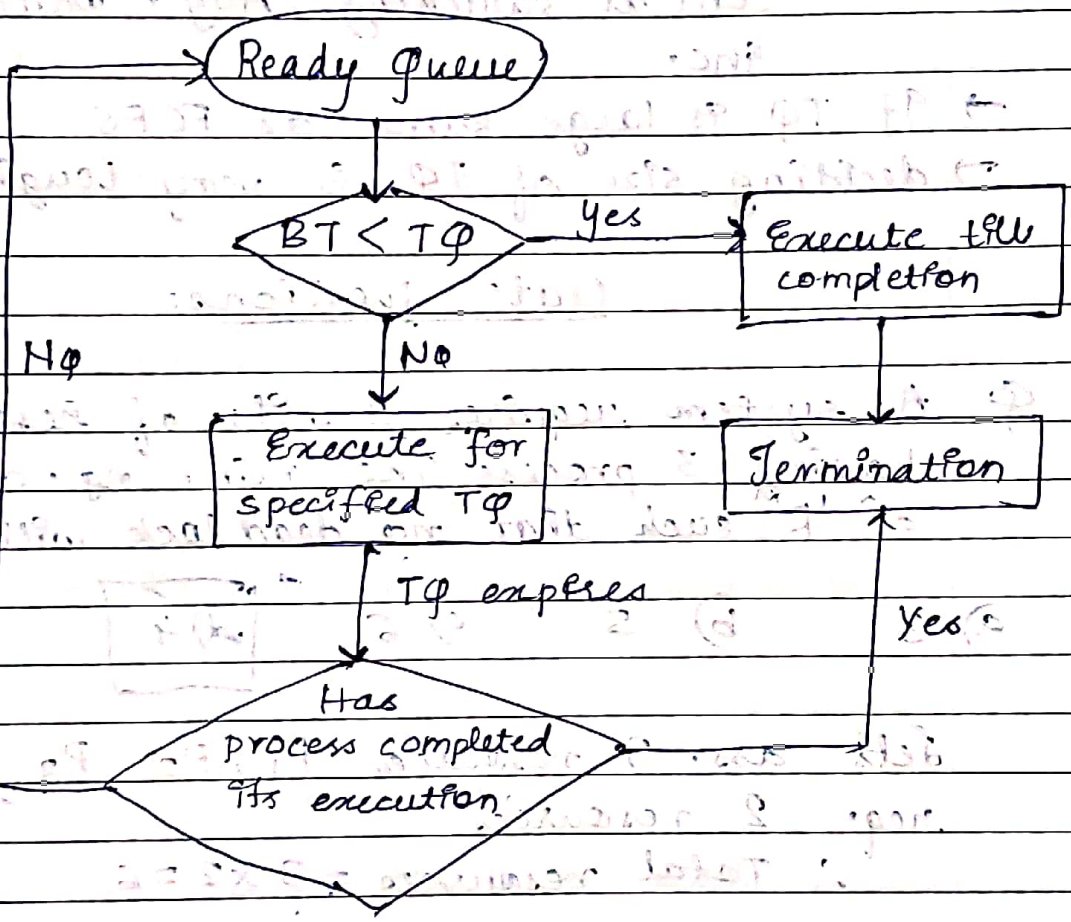
Blocked : P_2, P_3

CPU utilization = $\frac{\text{Expected}}{\text{Actual}} = \frac{10+15+5}{36} = \frac{30}{36} = 0.833$

$= 83.3\%$

CPU idle % = $\frac{6}{36} \times 100 = 16.68\%$

Flowchart for Round-Robin



Advantage : → Easy & simple to implement

→ Each process gets a fair share of CPU

→ No starvation

Scanned with OKEN Scanner

→ No convoy effect

→ deterministic response time

→ Priority is same for each process

Disadvantage:

→ Throughput depends on TQ

→ If TQ is small → more overhead of context switching, Avg. waiting time inc.

→ If TQ is large same as FCFS

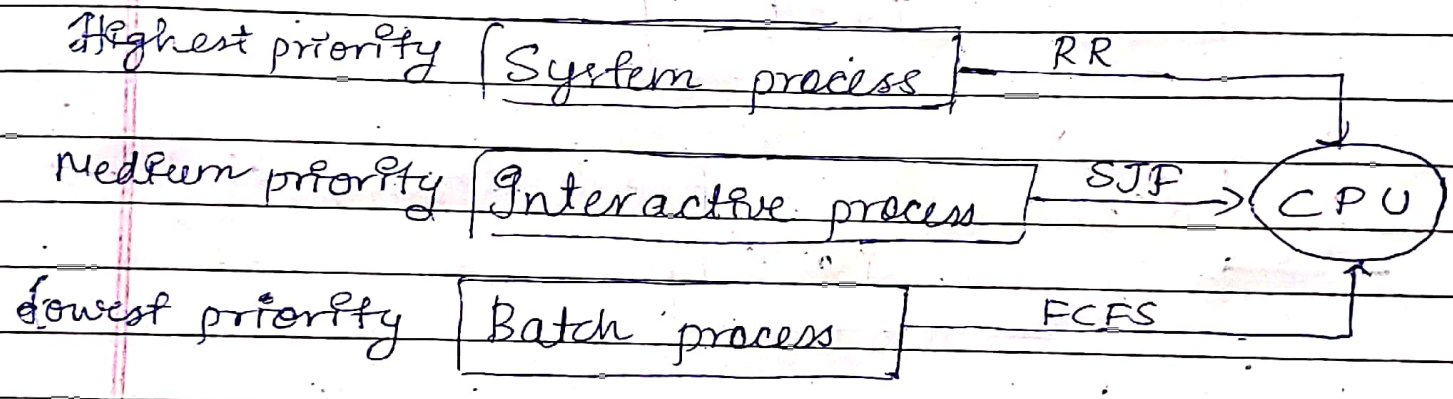
→ deciding size of TQ is very tough

Multi-level Queue scheduling

In every scheduling algo. till now, we've taken only one queue.

→ There must be different queue for diff. process
→ Every process can have their own algorithm

1) System calls - Round Robin
2) Interactive calls - SJFS

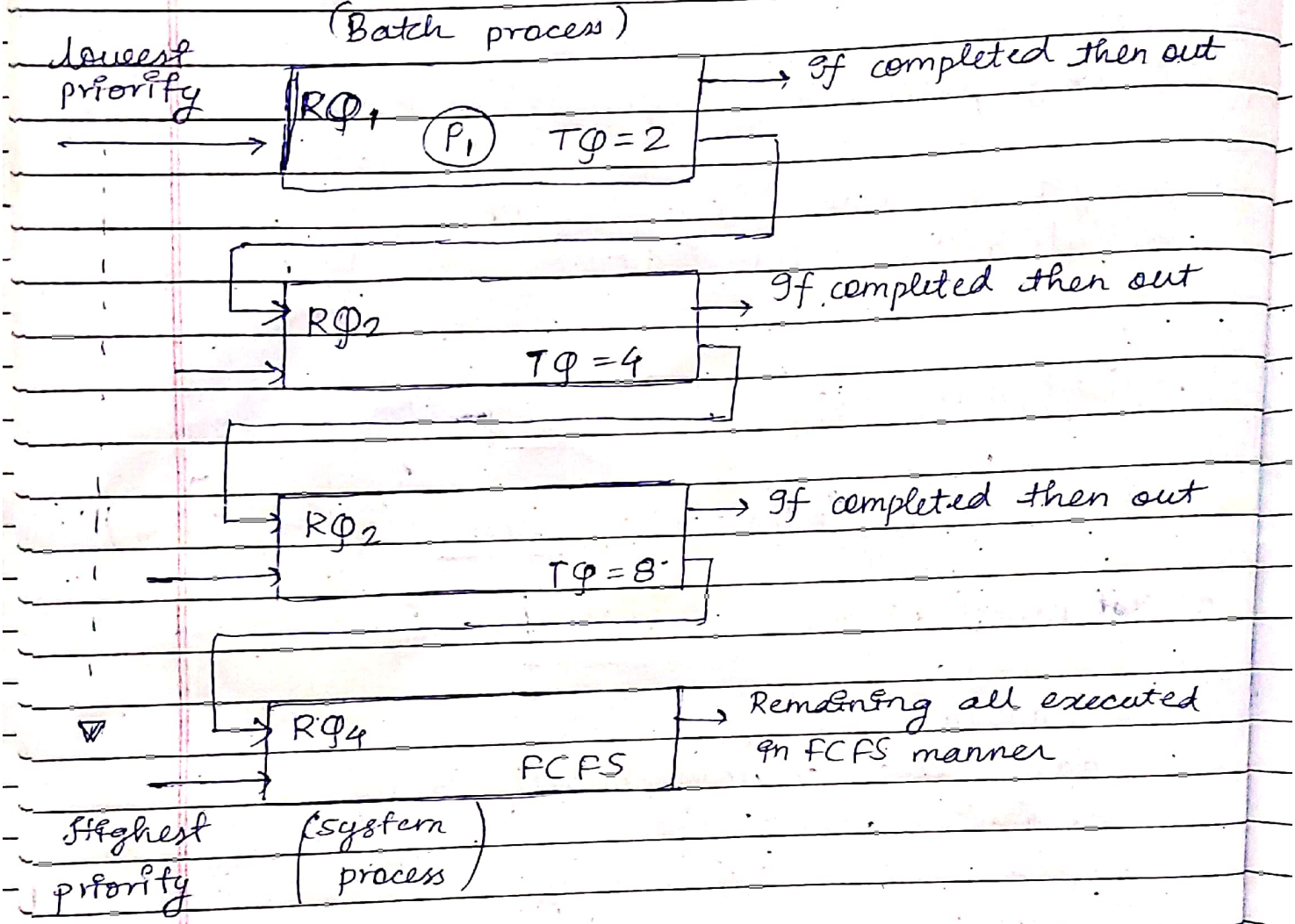


- Problem:
Lower level or lower priority queues may experience starvation due to more no. of calls on high priority.

Resolved using multi-level feedback queue

Multi-level feedback queue

More system process - lowest priority waits
lowest priority - feedback



P_1	19	17	13	5
		TQ=2	TQ=4	TQ=8