Process Scheduling

→ Basis of multi-programming OS.

→ By switching the CPU among processes, the OS can make the computer more ~~productive~~ productive.

→ Many processes are kept in memory at a time, when a process must wait or time quantum expires, the OS takes the CPU away from that process and gives the CPU to another process and this pattern continues.

→ Process scheduling is the process of managing and prioritizing the execution of multiple processes on a computer system.

→ It's essential for efficient resource allocation, preventing conflicts. and ensuring that all processes get a fair share of the CPU time.

CPU scheduler

→ Whenever the CPU becomes idle, OS must select one process from the ready queue to be executed

→ Done by short term scheduler

Scheduling → Non-Preemptive scheduling

Scheduling → Preemptive scheduling

Non-preemptive scheduling

→ Once CPU has been allocated to a process, the process keeps the CPU until it releases CPU either by terminating or by switching to wait-state.

→ Starvation occurs as a process with long burst time may starve less burst time process

→ Low CPU utilization

→ In other words non-preemptive scheduling is a CPU scheduling method where once a process is allocated the CPU it retains control until it either voluntarily releases the CPU or until it terminates.

Preemptive scheduling

→ CPU is taken away from a process after time quantum expires along with terminating or switching to wait-state.

→ Less starvation

→ High CPU utilization

→ Preemptive scheduling is a scheduling method in which OS can interrupt a currently running process and allocate CPU to another process.

Goals of CPU scheduling

(a) Maximum CPU utilization

(b) Minimum Turnaround time (TAT)

(c) Minimum wait time

(d) Minimum response time

(e) Maximum throughput of system

Definition

Throughput : Number of processes completed per unit time.

Arrival time (AT) : Time when process is arrived at the ready queue

Burst time (BT) : The time required by the process for its execution.

'OR'

amount of time a process needs to execute on the CPU.

Turnaround time (TAT) : Time taken from first time process enters ready state till it terminates.

TAT = Completion time − Arrival time

OR

TAT = Waiting time + Burst time

Wait time (WT) : Time process spends waiting for CPU

wait time = turnaround time − burst time

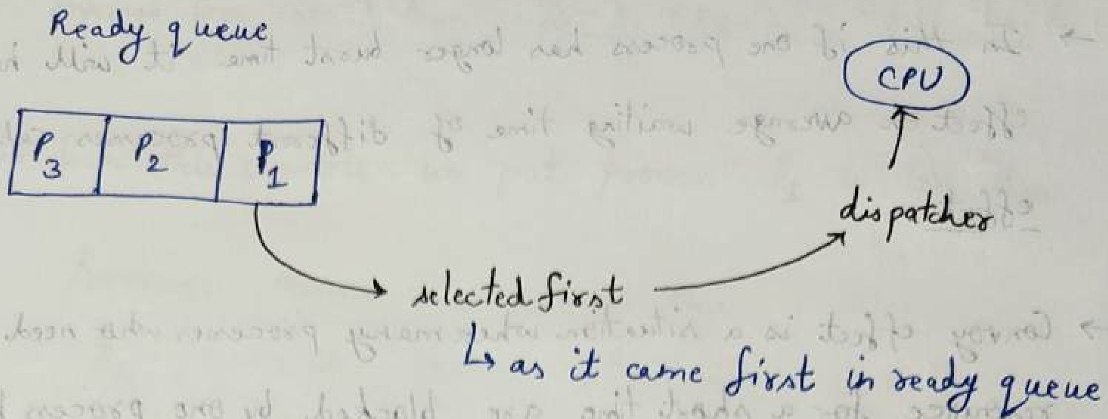
Response time : Time duration between process getting into ready queue and process getting CPU for the first time.

Completion time (CT) : Time taken till process gets terminated.

① FCFS (First come First serve)

→ Whichever process comes first in the ready queue will be given CPU first.

Ready queue



eg :- Processes Burst time

Processes	Burst time
P_1	10
P_2	5
P_3	7

Gantt chart

P_1	P_2	P_3

t=0 10 15 22

Processes	Burst time	Waiting time	Turnaround time
P_1	10	0	10
P_2	5	10	15
P_3	7	15	22

Average waiting time $= \frac{0+10+15}{3} = \frac{25}{3} = 8.33$ time units

Average turnaround time $= \frac{10+15+22}{3} = 15.66$ time units

→ In this if one process has longer burst time it will have major effect on average waiting time of different processes called convoy effect.

→ Convoy effect is a situation where many processes who need to use a resource for a short time are blocked by one process holding that resource for a long time.

→ This causes poor resource management.

eg:-

Processes	Arrival time	Burst time
P_1	0	20
P_2	1	2
P_3	2	2

Gantt chart

P_1	P_2	P_3
t=0 – 20	20 – 22	22 – 24

Processes	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P_1	0	20	20	20	0
P_2	1	2	22	21	19
P_3	2	2	24	22	20

$$\text{Average waiting time} = \frac{0+19+20}{3} = \frac{39}{3} = 13 \text{ time units}$$

Average turn around time = $\frac{20+21+22}{3} = 21$ time units

→ So if in this example we put process P_1 in last then.

Processes	Arrival time	Burst time
P_2	0	2
P_3	1	2
P_1	2	20

Gantt chart

P_2	P_3	P_1

t=0 2 4 24

Processes	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P_2	0	2	2	2	0
P_3	1	2	4	3	1
P_1	2	20	24	22	2

Average waiting time = $\frac{0+1+2}{3} = \frac{3}{3} = 1$ time units

Average turnaround time = $\frac{2+3+22}{3} = \frac{27}{3} = 9$ time units

→ Hence there is a difference in average waiting time this is due to convoy effect.

② SJF (Shortest Job First) {Non-preemptive}

→ Process with least burst time will be dispatched to CPU first.

→ Must do estimation for burst time for each process in ready queue beforehand {correct estimation of burst time is an impossible task (ideally)}.

→ Run lowest time process for all time then choose job having lowest burst time at that instance.

→ This will suffer from convoy effect as if the very first process which came in ready queue is having a large burst time.

→ Process starvation might happen (as if a process is with large burst time and all the other processes are of small burst time then process with large burst time might not get CPU).

eg :- {all process arrived at the same time}

Process	Burst time
P_1	5
P_2	9
P_3	3

Gantt chart

P_3	P_1	P_2
t=0 – 3	3 – 8	8 – 17

Processes	Burst time	Completion time	Turnaround time	Waiting time
P_1	5	8	8	3
P_2	9	17	17	8
P_3	3	3	3	0

Average waiting time $= \frac{0+8+3}{3} = \frac{11}{3} = 3.66$ time units

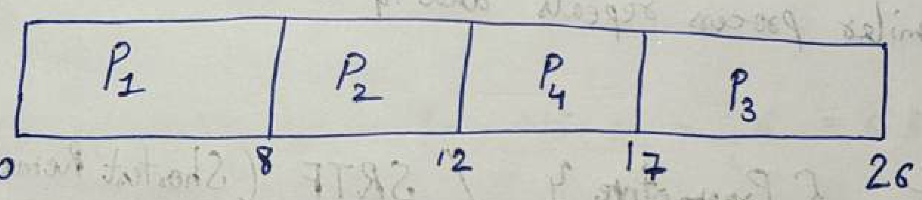
Average turn around time $= \frac{8+17+3}{3} = \frac{28}{3} = 9.33$ time units

→ In case of different arrival criteria for SJG algos is <u>arrival time + burst time</u>.

eg :-

Process	Arrival time	burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantt chart



Process	Arrival time	burst time	completion time	Turnaround time	Waiting time
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	26	24	15
P_4	3	5	17	14	9

$$\text{Average waiting time} = \frac{0+7+15+9}{4} = \frac{31}{4} = 7.75 \text{ time units}$$

$$\text{Average turnaround time} = \frac{8+11+24+14}{4} = \frac{57}{4} = 14.25 \text{ time units}$$

→ Can also understand this in ~~another~~ way like

- When arrival time = 0 or can say in ready queue only P_1 is present ~~without~~ due to 0 arrival time.
- So during this time CPU can only be alloted to P_1 as only 1 process is present.
- Then when P_1 is terminated (as its non-preempted so P_1 will be execution until it has completed it's execution) P_2, P_3, P_4 are also in ready queue (as they have arrived at 1, 2, 3 {arrival time})
- Then based on burst time they are differentiated and the one with lowest burst time is alloted the CPU (i.e P_2)
- Then similar process repeats and P_4 is selected then P_3.

(3) SJF {Preemptive} / SRTF (Shortest Remaining Time First)

→ Works same as SJF but only new feature is preemption.

→ Less starvation

→ No convoy effect

→ Gives average waiting time less for a given set of processes as scheduling short job before a long one decreases the waiting time of short job more than it increases the waiting time of the long process

eg:-

Process	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantt chart

P_1	P_2	P_4	P_1	P_3

t=0   1   5   10   17   26

Process	Burst time	Arrival time	Completion time	Turnaround time	Waiting time
P_1	8	0	17	17	9
P_2	4	1	5	4	0
P_3	9	2	26	24	15
P_4	5	3	10	7	2

$$\text{Average waiting time} = \frac{9+0+15+2}{4} = \frac{26}{4} = 6.5 \text{ time units}$$

$$\text{Average turnaround time} = \frac{17+4+24+7}{4} = \frac{52}{4} = 13 \text{ time units}$$

→ P_1 was preempted after 1 time units as

P_1 remaining burst time = 8 - 1 = 7 time units

P_2 burst time = 4 time units.

Therefore as P_2 had less burst time so CPU was allocated to it.

{ Can say main issue in SJF is calculating accurate burst time. }

→ In this also starvation might occur as a large burst time process might not get CPU due to many small burst time processes.

(4) Priority scheduling {Non-preemptive}

→ Priority is assigned to a process when it is created.

eg :-

Process	Priority	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
1	2	0	4	4	4	0
2	4	1	2	25	24	22
3	6	2	3	23	22	19
4	10	3	5	9	6	1
5	8	4	1	20	16	15
6	12	5	4	13	8	4
7	9	6	6	19	13	7

Gantt chart

P_1	P_4	P_6	P_7	P_5	P_3	P_2
t=0 – 4	4 – 9	9 – 13	13 – 19	19 – 20	20 – 23	23 – 25

$$\text{Average waiting time} = \frac{0+22+19+1+15+4+7}{7} = \frac{68}{7} = 9.71 \text{ time units}$$

Average turnaround time = $\frac{4+24+22+6+16+8+13}{7} = \frac{93}{7} = 13.28$ time units

{ Greater priority = Higher priority in above example }
{ like priority 9 > priority 7.

{ In this we have assumed that our algo checks every 1 second to compare }
{ priority

→ SJF is a special case of general priority scheduling with priority inversely proportional to burst time.

(5) Priority scheduling {Preemptive}

→ Currently running job will be preempted if next job has higher priority.

→ Working same as non-preemptive priority scheduling, here preemption is added.

eg:-

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

Gantt chart

P_1	P_2	P_3	P_4	P_6	P_4	P_7	P_5	P_3	P_2	P_1

t=0 1 2 3 5 9 12 18 19 21 22 25

Average waiting time $= \dfrac{21+19+16+4+14+0+6}{7} = \dfrac{80}{7} = 11.42$ time units

Average turnaround time $= \dfrac{25+21+19+9+15+4+12}{7} = 15$ time units

* → Average waiting time here is more than that of ~~pro~~ non-preemptive priority scheduling in above example.

→ Overhead in this as too much context switching.

Note :-

→ Convoy effect can be there in both preemptive and non-preemptive as high priority job with large burst time can hold resources for a long period of time from other processes/jobs.

Drawback of Priority scheduling {in both preemptive and non-preemptive}

⇒ Indefinite waiting or extreme starvation

kind of extreme convoy effect

• Jobs with high priority will keep getting CPU but jobs with low priority won't get CPU due to which they will be waiting for an indefinite amount of time.

Solution : Ageing

→ Gradually increase priority of process that wait so long

eg :- increase priority by 1 every 15 mins.

⑥ Round Robin (RR)

→ Most popular

→ Like FCFS but preemptive

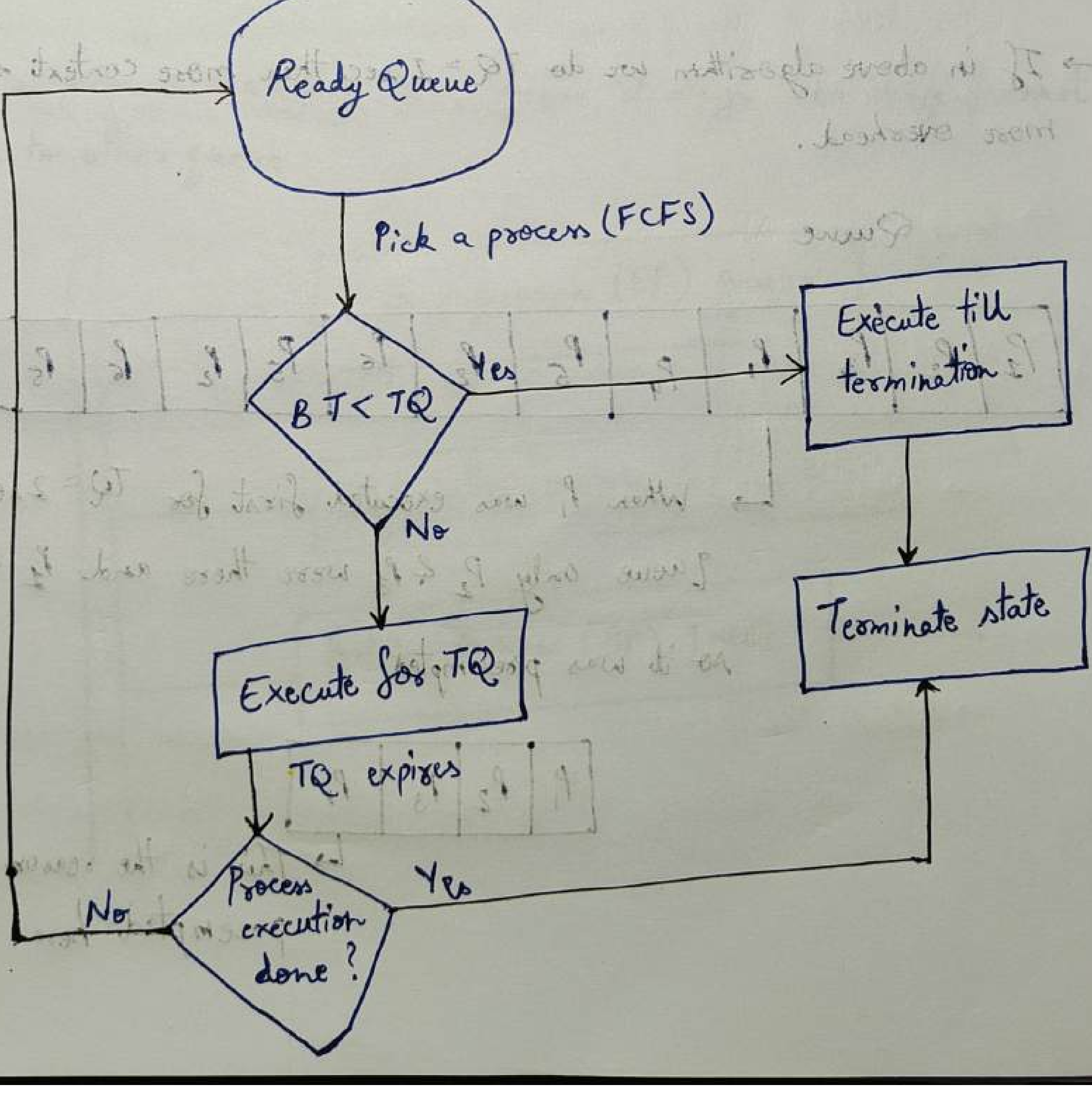
→ Designed for time sharing systems

→ Criteria : AT + time quantum (TQ), Doesn't depend on BT

→ No process is going to wait forever hence very low starvation {no convoy effect}

→ Easy to implement

→ If TQ or time quantum is small more will be the context switch (more overhead).



eg :-

Process	AT	BT
1	0	4
2	1	5
3	2	2
4	3	1
5	4	6
6	5	3

Gantt chart

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5	P_2	P_6	P_5

t=0 2 4 6 8 9 11 13 15 17 18 19 21

$TQ = 2 sec$

→ If in above algorithm we do $TQ = 1$ sec then more context switching so more overhead.

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

↳ When P_1 was executed first for $TQ = 2 sec$ then in Queue only P_2 & P_3 were there and P_1 didn't terminate so it was preempted.

P_1	P_2	P_3	P_1

↳ This is the reason why it was preempted here.

① Multi-Level Queue scheduling (MLQ)

→ There are 3 types of processes :

{Highest priority} (i) System process : Created by OS

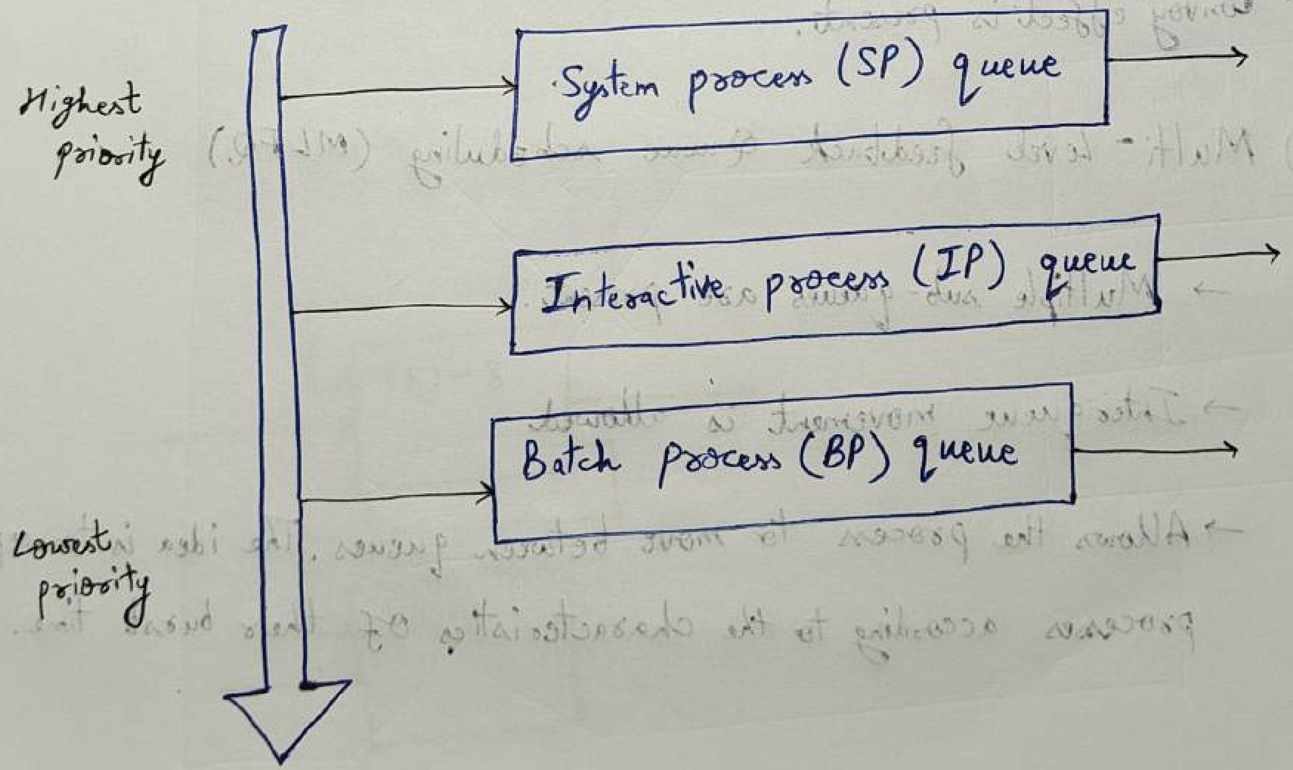
(ii) Interactive process (foreground process) : User input required.

{Lowest priority} (iii) Batch process (Background) : No I/P, No user input. They run in the background.

→ Ready queue is divided into multiple queues depending upon priority

→ A process is permenently assigned to one of the queues based on some property of process, memory size, process priority or process type.

→ Once a process is assigned to a queue it stays there only, cannot switch to other queue



→ Each queue has its own scheduling algorithm.

eg :- SP → Round Robin (RR)

IP → RR

BP → FCFS

→ Scheduling among different sub-queues is implemented as fixed priority preemptive scheduling

→ If an interactive process come and batch process is currently executing then batch process will be preempted.

→ Problem : Only after completion of all the processes from the top-level ready queue the further level ready queues will be scheduled. This came starvation for low priority process.

→ Convoy effect is present.

② Multi-Level feedback Queue scheduling (MLFQ)

→ Multiple sub-queues are present.

→ Interqueue movement is allowed

→ Allows the process to move between queues. The idea is to separate processes according to the characteristics of their burst time.

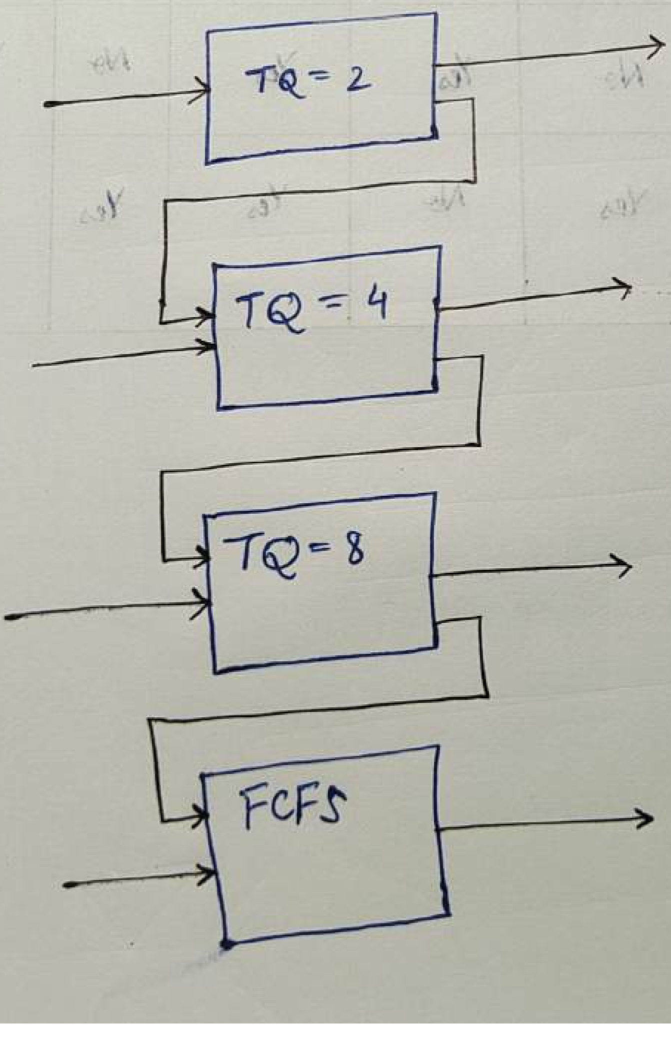
If a process uses too much CPU time, it will be moved to lower priority queue. This scheme leaves I/O bound and interactive processes in the higher priority queue. {As we want that kind of OS that listens to us} => {eg: If we launch an application but CPU is processing other processes.}

→ In addition a process that waits too much in a lower priority queue may be moved to a higher priority queue. This ~~form of~~ form of ageing prevents starvation.

→ Less starvation than MLQ.

→ It is flexible.

→ Can be configured to match a specific system design requirement.



Design of MLFQ

1. Number of Queues
2. Scheduling algorithm in each queue
3. Method to upgrade a process to a higher queue.
4. Demote a process to ~~convert~~ Lower queue.
5. Process P_1 → which queue will it be pushed.

Comparison

	FCFS	SJF	SRTF	Priority	P-Priority	RR	MLQ	MLFQ
Design	Simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Preemption	No	No	Yes	No	Yes	Yes	Yes	Yes
Convoy effect	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Overhead	No	No	Yes	No	Yes	Yes	Yes	Yes