Unit - IV

CPU scheduling & Algorithms

* - In a single processor system, only one process can run at a time.
- others must wait until the CPU is free.
- The main objective of multiprogramming is to have some process running all the time to improve CPU Utilization.
- In multiprogramming, several processes are kept in memory at one time.
- When one process has to wait, the os takes the ~~aw~~ CPU away from that process and gives the CPU to another process.
- This pattern continues all the time.

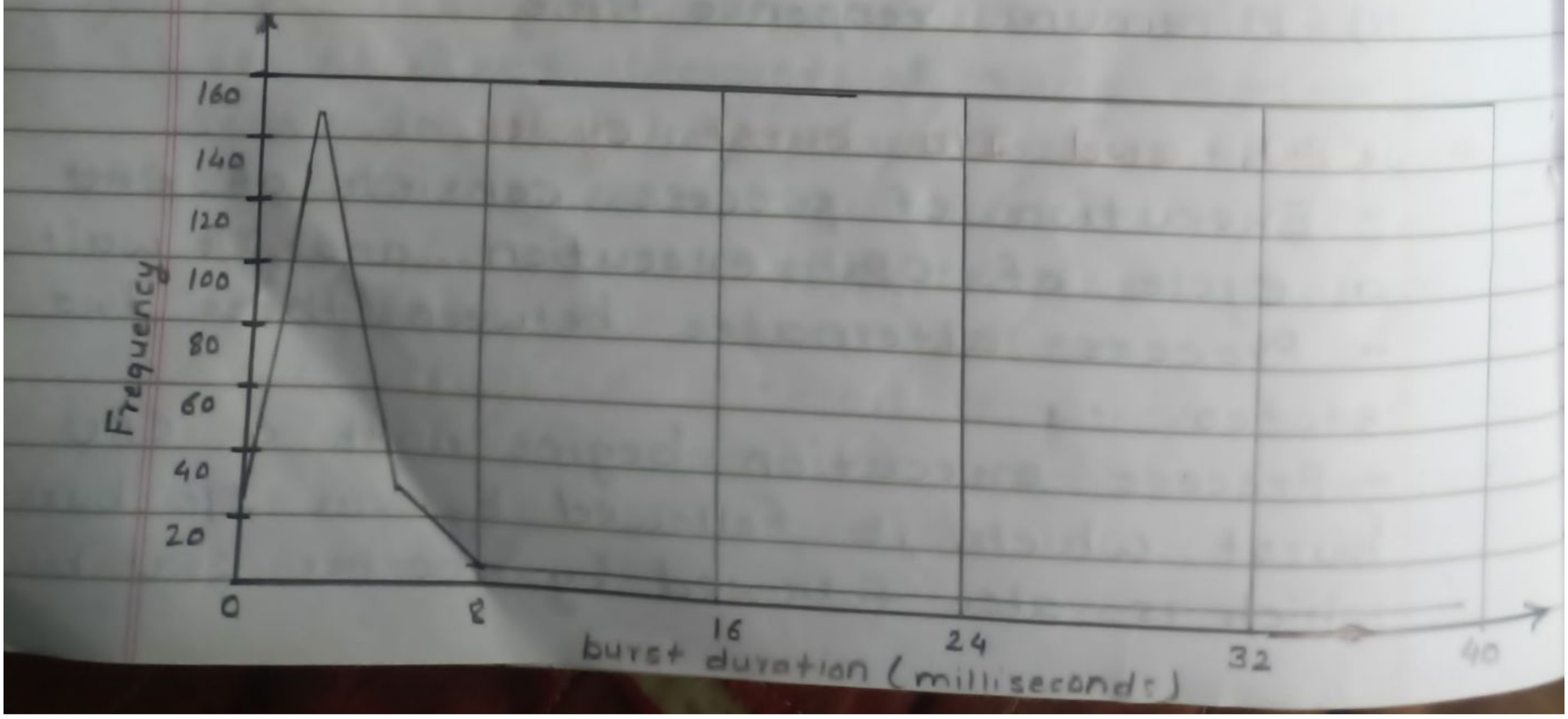
* Scheduling objectives :

1) Maximum CPU Utilization
2) Fair allocation of CPU
3) Maximum throughput
4) Minimum turn around time
5) Minimum waiting time
6) Minimum response time.

* CPU and I/O burst cycles :

- Execution of process consists of ~~CPU~~ a cycle of CPU execution and I/O wait.
- Process alternates between these two states.
- Process execution begins with a CPU burst which is followed by an I/O burst, which is also followed by another CPU burst

and then another I/o burst and so on.
- The final CPU burst ends with a system request to terminate the execution of a process.
- A CPU bound program need more CPU time to complete execution.
- Whereas, I/o bound program spends more time on I/o operation than execution by CPU.
- Depending on this distribution, scheduling algorithm can be selected.
- The duration of CPU bursts have been measured extensively.
- Although they vary from process to process and computer to computer they tend to have a frequency curve.
- The curve is generally charaterized as exponential or hyperexponential, with a large number of short CPU bursts and a small number of long CPU bursts.
- Diagram / Graph :



* Pre-emptive Scheduling :
- Pre-emptive scheduling allow us to take CPU from process during execution.
- If highest priority process arrives in the system, CPU from currently executing low priority process is allocated to it.
- It means that, we can interrupt currently executing process and take CPU from it.

* Non-Pre-emptive Scheduling :
- Once the CPU has been allocated to a process, the process the CPU until it releases the CPU either by terminating or by switching to the waiting state.

* Scheduling criteria :

1) CPU Utilization :
It is amount of time CPU remain busy.

2) Throughput :
Number of processes that are completed per unit time.

3) Turnaround Time :
The interval from the time of submission of a process to the time of completion.

4) Waiting time:
It is the amount of time that a process spends waiting in the ready queue. sum of ~~time~~ periods spent waiting in the ready queue is waiting time.

5) Response time:
Time from the submission of a request until the first response is produced.

* Types of scheduling algorithms:

1) First come first served (FCFS):
- First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time.
- The job which comes first in the ready queue will get the CPU first.
- The lesser the arrival time of the job, the sooner will the job get the CPU first.
- FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.
- Advantages:
i) Simple
ii) Easy
iii) First come, First serve.

- Disadvantages:

i) The scheduling method is non-preemptive, the process will run to the completion.

ii) Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.

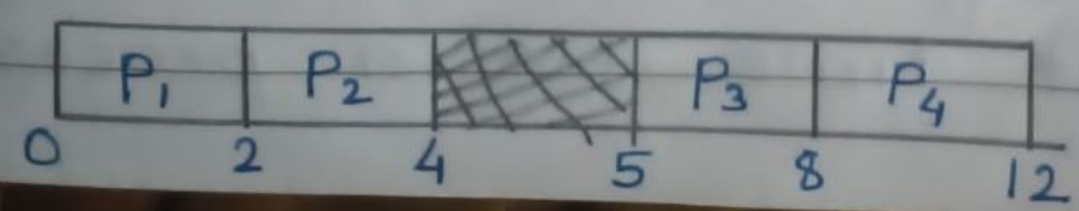
iii) Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

- Example:

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

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT
P_1	0	2	2	2	0
P_2	1	2	4	3	1
P_3	5	3	8	3	0
P_4	6	4	12	6	2

- Gantt Chart:



Average TAT = (2 + 3 + 3 + 6) / 4
= 14 / 4
= 3.5

Average WT = (0 + 1 + 0 + 2) / 4
= 0.75

2) Shortest Job First (SJF):
- Shortest Job First (SJF) algorithm may be pre-emptive or non-preemptive.
- SJF scheduling algorithm, schedules the processes according to their burst time.
- In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.
- However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.
- Advantages:
i) Maximum throughput
ii) Minimum average waiting and turnaround time.

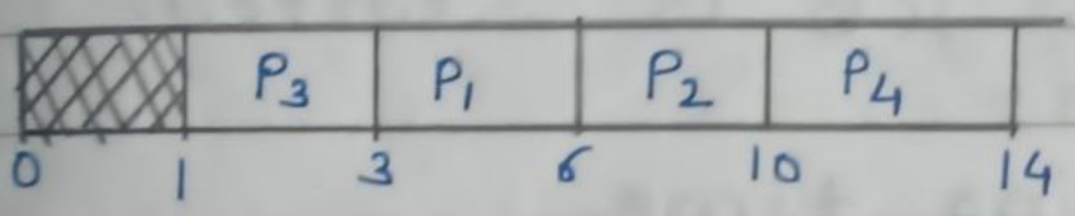
- Disadvantages:
i) May suffer with the problem of starvation.
ii) It is not implementable because the exact Burst time for a process can't

be known in advance.

- Example :

Process No	Arrival Time	Burst Time
~~P_1~~	1	3
P_2	2	4
~~P_3~~	1	2
P_4	4	4

- Gantt chart :



Process No	Arrival Time	Burst Time	Completion Time	TAT	WT
P_1	1	3	6	5	2
P_2	2	4	10	8	4
P_3	1	2	3	2	0
P_4	4	4	14	10	6

Average TAT = (5 + 8 + 2 + 10) / 4
= 6.25

Average WT = (2 + 4 + 0 + 6) / 4
= 12 / 4
= 3

3) Shortest Remaining Time (SRTN):
- In pre-emptive remaining shortest Job First (SJF), the process, whose remaining run time is shortest, is served first.
- When a new process arrives, its total time is compared to the current process remaining run time.
- If the new process needs less time to finish than the current process, the current process is suspended and the new job is started.
- Advantages:
i) Less waiting time.
ii) Quite good response for short processes.

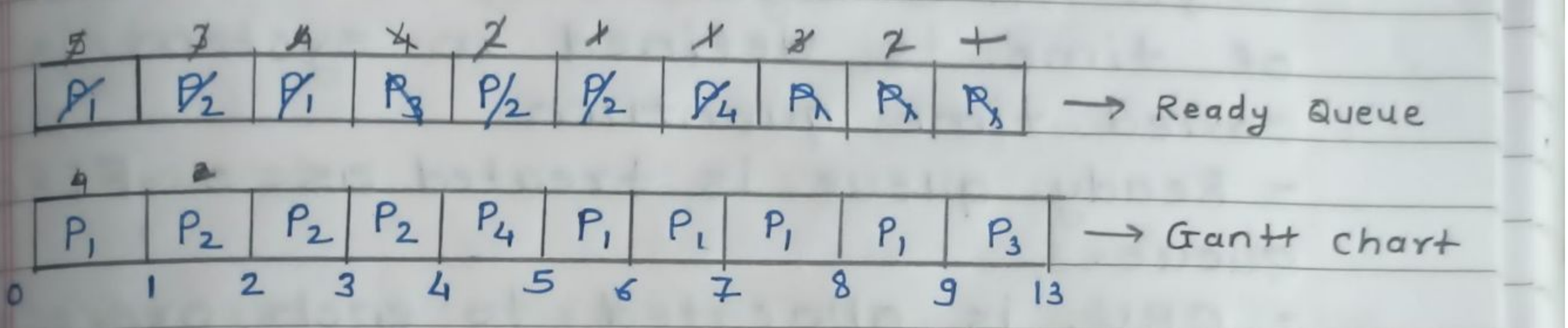
- Disadvantages:
i) Again it is difficult to estimate remaining time necessary to complete execution.
ii) Starvation is possible for long process. Long process may wait forever.
iii) Context switch overhead is there.

- Example:

Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	3
P_3	2	4
P_4	4	1

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	5	9	9	4	0
P_2	1	3	4	3	0	0
P_3	2	4	13	11	7	7
P_4	4	1	5	1	0	0

- Gantt chart :



Average Turnaround Time = (9 + 3 + 11 + 1) / 4
= 24 / 4
= 6

Average Waiting Time = (4 + 0 + 7 + 0) / 4
= 11 / 4
= 2.75

Average Response Time = (0 + 0 + 7 + 0) / 4
= 7 / 4
= 1.75

4) Round Robin (RR) Scheduling :-
- Round Robin scheduling is used in time-sharing systems where many processes gets CPU on time sharing manner.
- This is the preemptive version of FCFS scheduling.
- In this algorithm, a small unit of time is defined in system is called time quantum.
- Ready queue is treated as a FIFO queue.
- CPU is allocated to each process for given time quantum.
- Each process present in the ready queue is assigned the CPU for that time quantum.
- If the execution of the process is completed during that time then the process will terminate.
- else the process will go back to the ready queue and waits for the next turn to complete the execution.
- Advantages :-
i) It can be actually implementable in the system because it is not depending on the burst time.
ii) It doesn't suffer from the problem of starvation or convoy effect.

- Disadvantage :-

i) The higher the time quantum, the higher the response time in system.

ii) The lower the time quantum, the higher the context switching overhead in the system.

- Deciding a perfect time quantum is really very difficult task in the system.

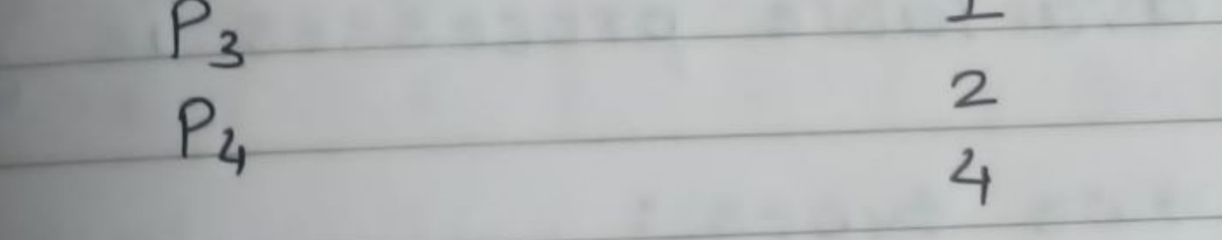
- Example :

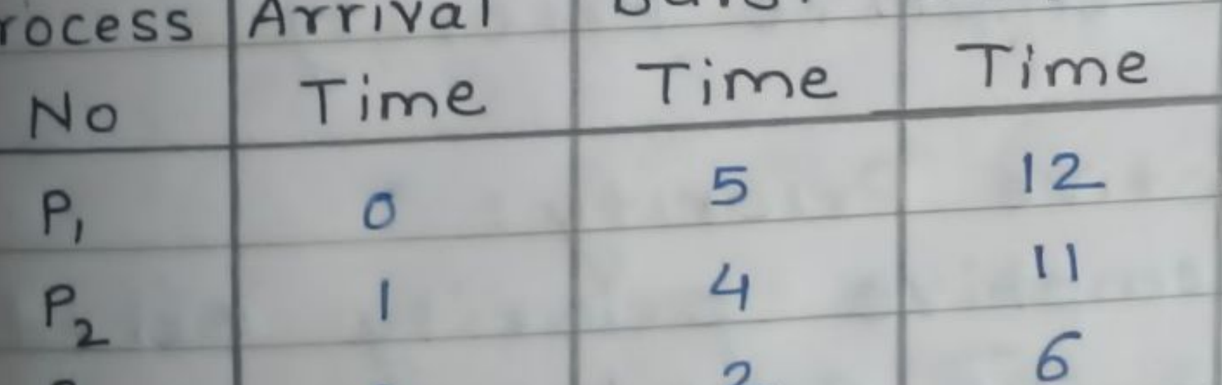
Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	4
P_3	2	2
P_4	4	1

Time Quantum = 2

Process No	Arrival Time	Burst Time	completion Time	TAT	WT	RT
P_1	0	5	12	12	7	0
P_2	1	4	11	10	6	1
P_3	2	2	6	4	2	2
P_4	4	1	9	5	4	4

- Gantt chart :





Average TurnAround Time = (12+10+4+5)/4
= 7.75

Average Waiting Time = (7+6+2+4)/4
= 19/4
= 4.75

5) Priority Scheduling :-
- In Priority Scheduling, there is a priority number assigned to each process.
- In some systems, the lower the number, the higher the priority.
- While, in the others, the higher the number, the higher will be the priority.
- The process with the higher priority among the available processes is given the CPU.
- There are two types :
  i) Non-Preemptive Priority
  ii) Pre-emptive Priority

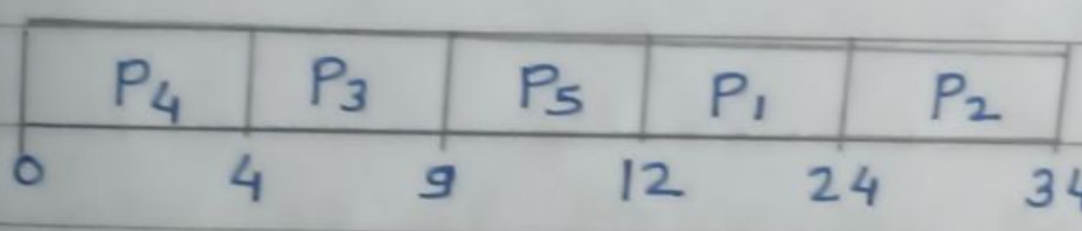
i) Non-Preemptive Priority :
- In non-preemptive priority scheduling, The processes are scheduled according to the priority number assigned to them.
- Once the process gets scheduled, it will run till the completion.
- Generally, the lower the priority number, the higher is the priority of the process.

- Example:

Process	Burst Time	Priority
P1	12	4
P2	10	5
P3	5	2
P4	4	1
P5	3	3

Process No	Burst Time	Priority	TAT	WT	Completion Time
P1	12	4	24	12	24
P2	10	5	34	24	34
P3	5	2	9	4	9
P4	4	1	4	0	4
P5	3	3	12	9	12

- Gantt chart:



Average TurnAround Time = (24 + 34 + 9 + 4 + 12) / 5
= 83 / 5
= 16.6

Average Waiting Time = (12 + 24 + 4 + 0 + 9) / 5
= 49 / 5
= 9.8

ii) ~~Non~~ Preemptive Priority :

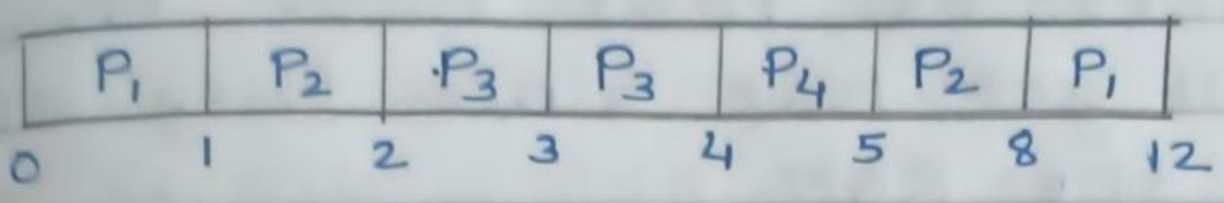
- In Pre-emptive Priority scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time.
- The one with the highest priority among all the available processes will be given the CPU next.
- In this scheduling, the process which is being executed can be stopped at the arrival of the higher priority process.
- Once all the processes gets available in ready queue, the algorithm will behave like non-preemptive priority scheduling.

- Example :

Process No	Priority	Arrival Time	Burst Time
P1	10	0	5
P2	20	1	4
P3	30	2	2
P4	40	4	1

- higher the number, higher the priority.

- Gantt chart:



Process No	Priority	Arrival Time	Burst Time	CT	TAT	WT
P1	10	0	5	12	12	7
P2	20	1	4	8	7	3
P3	30	2	2	4	2	0
P4	40	4	1	5	1	0

- Average Turn Around Time = (12+7+2+1) / 4
  = 22/4
  = 5.5

- Average Waiting Time = (7+3+0+0) / 4
  = 10/4
  = 2.5

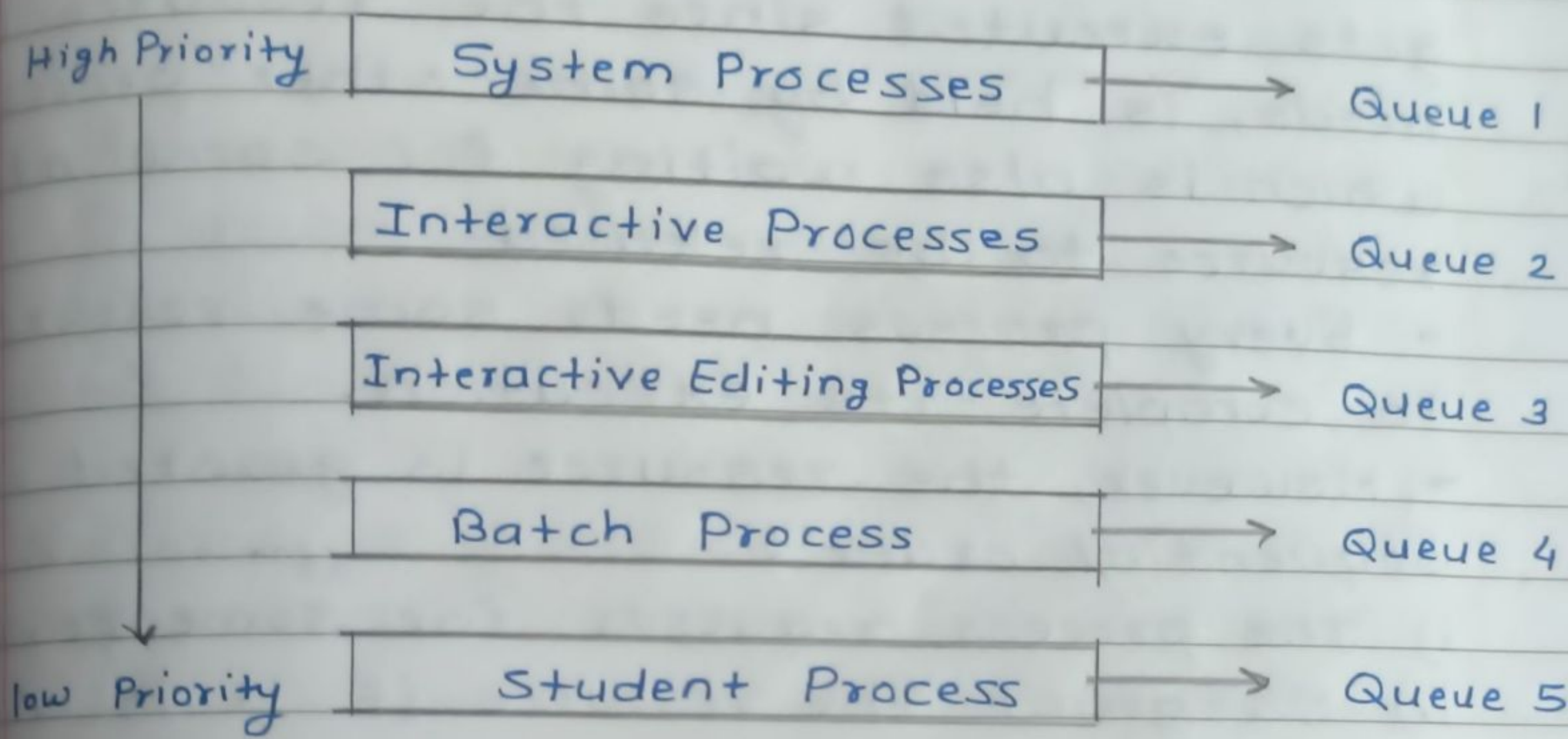
- Advantage of Priority Scheduling:
i) Higher priority processes do not have to wait for their chance to be executed due to the current running process.
ii) We are able to define priority of processes.

- Disadvantage of Priority scheduling:-
i) Since we only execute high priority processes, this can lead to starvation of of the processes that have a low priority.

ii) If the system eventually crashes, all the processes that have low priority will get lost since they are stored in the RAM.

6) Multilevel ~~Priori~~ Queue Scheduling:
- Sometimes, there is a need to catgorize processes into different groups.
- In multilevel Queue Scheduling algorithm, there are multiple ready queues.
- ~~Each~~ The allocation of process to the particular queue depends on property of that process such as memory size, priority of the process, or its types.
- Each queue has its own scheduling algorithm.
- To schedule each queue, there must be scheduling between different queues.
- For that purpose, we use priority preemptive scheduling.
- Every higher priority queue has complete priority over lower-priority queues.
- Unless and until higher priority queues become empty, processes in lowest priority queue cannot execute.
- If process in lower priority queue is executing and at the same time other process belonging to higher priority queue arrives.
- In such cases, the currently executing process in lower priority queue should

be pre-empted.
- In this algorithm, a fixed amount of CPU time is given to each queue and a within this time different processes from this queue are scheduled.
- Diagram :-



- Advantages :-
i) You can use multilevel queue scheduling to apply different scheduling methods to distinct processes.
ii) It will have low overhead in terms of scheduling.

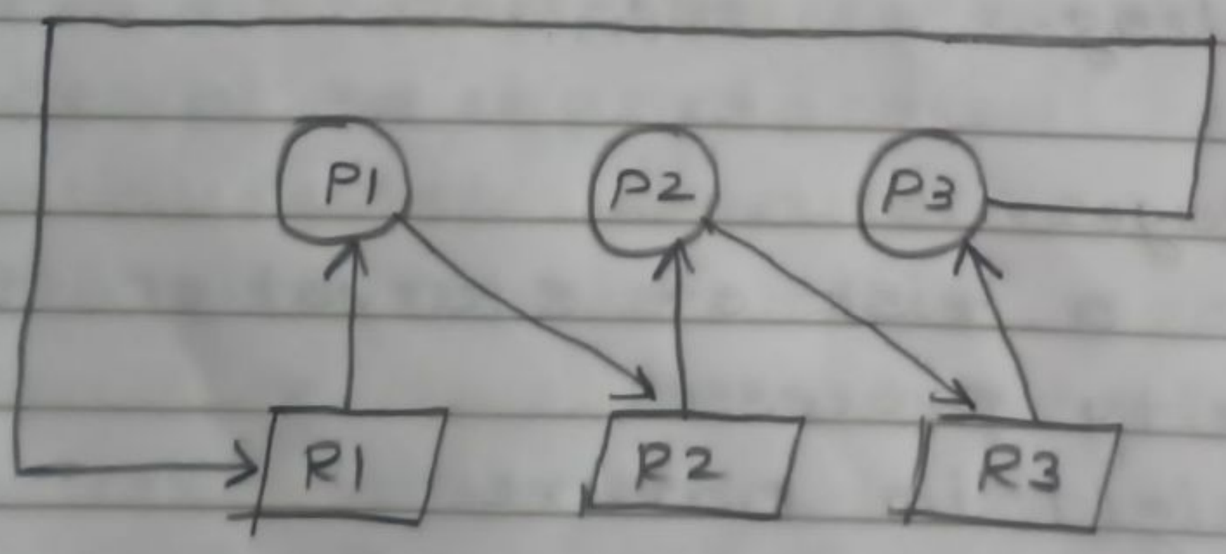
- Disadvantages :-
i) There is a risk of starvation for lower priority processes.
ii) It is rigid in nature.

* Deadlock :

- A deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process.
- In this situation, none of the process gets executed since the resource its needs, is held by some other process which is also waiting for some other resource to be released.
- Every process needs some resources to complete its execution.
- However, the resource is granted in a sequential order :

i) The process requests for some resource.
ii) OS grant the resource if it is available otherwise let the process waits.
iii) The process uses it and release on the completion. i.e.

i) Request
ii) Use
iii) Release.



Diagram

Necessary conditions leading to deadlocks:

i) Mutual Exclusion :
- At least one resource must be kept in non-sharable mode.
- If another process request it, it must be wait for it to be released.

ii) Hold and Wait :
- A process waits for some resources while holding another resource at the same time.

iii) No preemption :
- Resource cannot be preempted.
- i.e. A resource can be released by a process holding it, only after that process has completed its task.

iv) Circular wait :
- All the processes must be waiting for the resources in cyclic manner.
- The last process is waiting for the resource which is being held by first process.

* Deadlock Handling :

i) Deadlock Prevention :

- Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously.
- If it is possible to violate one of the four conditions at any time then the deadlock can ~~occur~~ never occur in the system.

i) Mutual Exclusion :

- In mutual exclusion, a resource can never be used by more than one process simultaneously.
- If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.
- but, we cannot force a resource to be used by more than one process at the same time because it may decrease the performance.
- Therefore, we cannot violate mutual exclusion for a process practically.

ii) Hold and Wait :

- Hold and wait condition occurs when a process holds a resource and waiting for some other resource to complete

its task.
- Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in cyclic manner.
- So, we need to find solution by which a process either doesn't hold any resource or doesn't wait.
- That means, a must be assigned all the necessary resources before the execution starts.
- A process must not wait for any resource once the execution has been started.

iii) No Preemption:
- Deadlock arises due to the fact that a process can't be stopped once it starts.
- However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.
- This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has been done till now can become inconsistent.

iv) Circular wait :

- To violate circular wait, we can assign a priority number to each of the resource.
- A process can't request for a lesser priority resource.
- This ensure that not a single process can request a resource which is being utilized by some other process. and no cycle will be formed.

Condition	Approach	Is Practically Possible ?
Mutual Exclusion	Spooling	NO
Hold and wait	Request for all the resources initially.	NO
No preemption	takes away all the resources	No
Circular wait	Assign priority to each resources and order resources numerically.	Yes

ii) Deadlock Avoidance:

- In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system.
- The state of the system will continuously be checked for safe and unsafe states.
- In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.
- The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need.
- The deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.
- Safe and unsafe states:
- The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.
- A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

- If the system cannot fulfill the request of all processes then the state of the system is called unsafe.