CPU Scheduling

CPU scheduling

CPU scheduler is often called as **short- term scheduler**. Actually whenever CPU becomes idle, opearting system has to select one process, which is in the ready queue to be executed. This selection process is called **CPU scheduler**. The function of scheduler is to select a processes from the memory that are in ready queue (for the execution) and allocates them to CPU. The queue is not FIFO *i.e.* first in first out queue. It may be priority queue, tree or unordered linked list. Actually all the process in the queue waiting for execution is just waiting for a chance to run on CPU.

Preemptive Scheduling

CPU scheduling decision takes place under four circumstances. They are

- When a process switches from running state to wait state.
- When a process switches from running state to ready state.
- When a process switches from wait state to ready state
- When a process terminates.

A process which switches to new process before completing its own process is called **preempted process**. A preemptive scheduling is motivated to improve system performance. Preemption can be used in one of the two ways. A process may be preempted when it enters a passive state to free the process from processing another request. Secondly, it permits the scheduler to respond to the development within the system *e.g.* arrival of new requests, changes in process priorities.

Non-preemptive Scheduling

A process is said to be in the state of non-preemptive scheduling if the process always processes the scheduled request for its completion.

A scheduler schedules a request for a non preemptible server only when processing of the previously scheduled requests gets completed. Non preemptive scheduling algorithm is simple. Because scheduling is performed only when the previously scheduled request gets completed. The list of pending requests never contains a preempted request *i.e.* request are complete in their order in which they are scheduled.

Scheduling Criteria

CPU Utilization

The CPU must be kept busy all the time. Hence utilization of CPU ranges from 0% to 100%. But in real, it ranges from 40% (for light loaded system) to 90% (for heavily used system).

Thoroughput

Work is done, whenever CPU is busy in execution of process. Work can be measured as number of processes completed per unit of time is called thoroughput. For large processes, this rate vary from one process per hour but for short transaction thoroughput is 10 processes per second.

Turnaround time

We must know what will be the time required by the process to execute. So the time interval from submission of process to completion of process is called **turn around time**. Turn around time is the sum of period of waiting time, execution on CPU and performing input/output. It is also limited by speed of output device.

Waiting time

CPU scheduling algorithm does not affect the amount of time during which a process executes or does input/output; it only affects the amount of time that process spends in waiting in the ready queue. The waiting time is the sum of period spend waiting in ready queue.

Response time

Response time is the time measured from the submission of request until the first response is produced. It is only the amount of time it takes to start responding not the time it takes to output that response.

Scheduling Algorithms

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- PRIORITY
- RR
- MFQ (Multilevel Feedback Queue Scheduling)
- MULTILEVEL QUEUE

1) FCFS

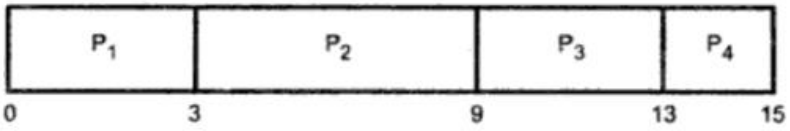
It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

EXAMPLE

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.

Process	Burst time
P_1	3
P_2	6
P_3	4
P_4	2

i) Gantt chart :



ii) Waiting time :

Process	Waiting time
P_1	0
P_2	3
P_3	9
P_4	13

iii) Average waiting time :

Sum of the all the process waiting times divided by number of processes.

$$\text{Average waiting time} = \frac{\text{Waiting times of all processes}}{\text{Number of processes}}$$

$$= \frac{0+3+9+13}{4}$$

$$= \frac{25}{4}$$

$$= 6.25$$

iv) Turnaround time :

It is computed by subtracting the time the process entered the system from the time it terminated. Process entered time is 0 for all processes.

Process	Turnaround time (Burst time + Waiting time)
P_1	3+0=3
P_2	6+3=9
P_3	4+9=13
P_4	2+13=15

v) Average turnaround time :

$$= \frac{3+9+13+15}{4} = 10$$

Question1

Use FCFS algorithm for the following and calculate average waiting time.

Process	*Burst Time*
P_1	23
P_2	4
P_3	4

Question2

Use FCFS to calculate average waiting time.

Process	*Burst Time*
P_1	15
P_2	5
P_3	3
P_4	2

Question3

Use FCFS for calculating average waiting time :

(i) Process	Burst time	(ii) Burst time
P_1	25	5
P_2	5	5
P_3	5	25

(iii) Burst time is 3, 13, 2

(iv) Burst time is 1, 1, 5

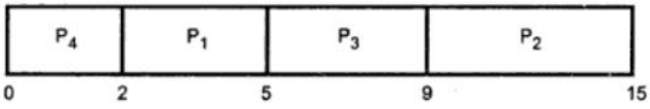
2) SJF (Shortest Job First Scheduling)

- This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is free, it is assigned to the process of the ready queue which has smallest next CPU burst. If two processes have the same length, FCFS scheduling is used to break the tie.
- SJF scheduling algorithm is used frequently in long term scheduling. SJF algorithm may be either **preemptive** or **nonpreemptive**.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst. Let us consider the set of process with burst time in milliseconds.

Process	Burst time
P_1	3
P_2	6
P_3	4
P_4	2

Arrival time of the process is 0 and processes arrive in the order of P_1, P_2, P_3 and P_4. The Gantt chart, waiting time and turnaround time is given below.

i) Gantt chart :



ii) Waiting time :

Process	Waiting time
P_1	2
P_2	9
P_3	5
P_4	0

iii) Average waiting time :

$$\text{Average waiting time} = \frac{2+9+5+0}{4}$$

$$= \frac{16}{4}$$

$$= 4$$

Question1

Use SJF scheduling algorithm for calculation of average waiting time

Process	*Burst Time*
P_1	6
P_2	8
P_3	7
P_4	3

Question2

Use SJF scheduling algorithm for calculation of average waiting time.

Process	*Burst Time*
P_1	5
P_2	5
P_3	1

3) PRIORITY SCHEDULING

In this algorithm, a priority is associated with each process and CPU is allocated to the process according to its priority *i.e.* CPU is allocated to the process having highest priority. Equal priority processes are scheduled in FCFS order.

Priority scheduling can be of two types:

1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stopped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

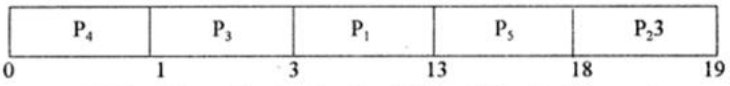
Example :***Use priority scheduling algorithm to calculate average waiting time.***

Process	*Burst time*	*Priority*
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P5	5	2

Solution : So P_4 has highest priority than P_3, P_1, P_5, P_2 so we can rearrange them as

Process	Burst Time	Priority
P_4	1	5
P_3	2	4
P_1	10	3
P_5	5	2
P_2	1	1

Gantt Chart



Average Waiting Time = (0 + 1 + 3 + 13 + 18)/5 = 35/5 = 7 millisecond

Question1

Following jobs are given with their burst time priority . Find the average waiting time by using priority scheduling algorithm.

Process	*Burst Time*	*Priority*
P_1	5	3
P_2	2	1
P_3	1	2

Question2

Consider the below table fo processes with their respective CPU burst times and the priorities.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

Question3

Using priority scheduling algorithm. Calculate the average waiting time.

Process	Burst time	Priority
A	8	3
B	3	1
C	2	4
D	5	2

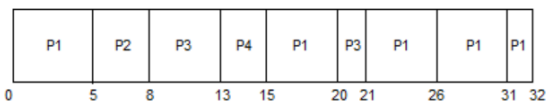
4) <u>Round Robin Scheduling</u>

- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as **time quantum** or **time slice**.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.

Example : Calculate completion time of each process when time quantam is 5 ms .

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



Example

Consider the set of 5 processes whose arrival time and burst time are given below-

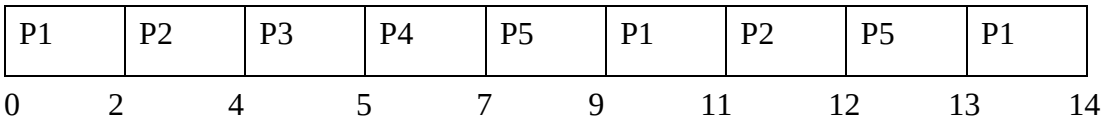
Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

Solution

- Turn Around time = Completion time – Arrival time
- Waiting time = Turn Around time – Burst time

Gantt Chart



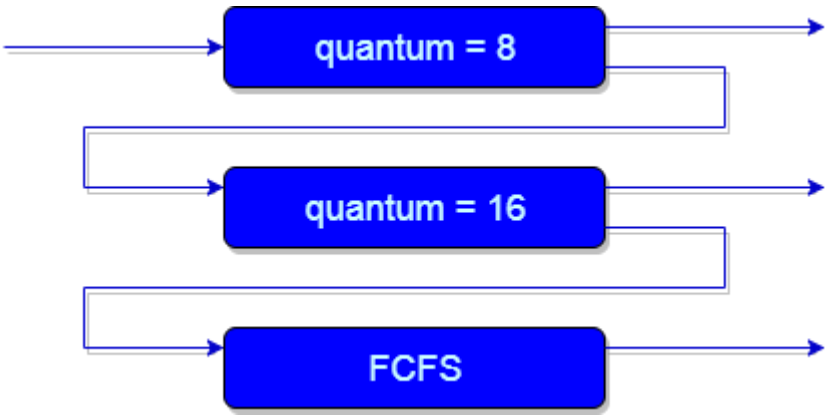
Process Id	Arrival time	Burst time	Completion time	Turnaround time	Waiting Time
P1	0	5	14	14	9
P2	1	3	12	11	8
P3	2	1	5	3	2
P4	3	2	7	4	2
P5	4	3	13	9	6

Average Waiting Time = 9+8+2+2+6 / 5 = 5.4

5) Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of **aging prevents starvation.**



In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

6) Multilevel Queue Scheduling

Another class of scheduling algorithm has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground (or interactive) processes and background (or batch) processes. These two types of processes have different response-

time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

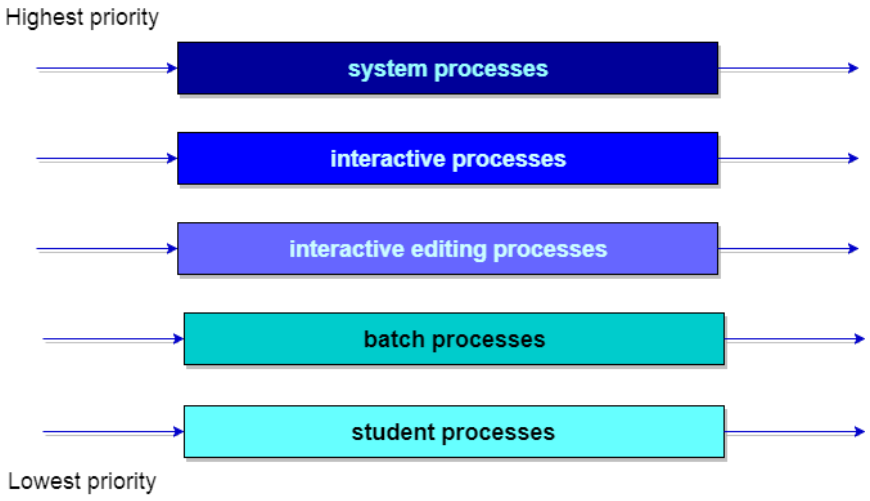
For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be pre-empted.



2.18.7 Comparison between FCFS and RR Method

Sr. No.	FCFS	Round Robin
1.	FCFS decision made is nonpreemptive.	RR decision made is preemptive.
2.	It has minimum overhead.	It has low overhead.
3.	Response time may be high.	Provides good response time for short processes.
4.	It is troublesome for time sharing system.	It is mainly designed for time sharing system.
5.	The workload is simply processed in the order of arrival.	It is similar like FCFS but uses time quantum.
6.	No starvation in FCFS.	No starvation in RR.

2.22 Comparison of Scheduling Algorithms

Algorithm	Police type	Used in	Advantages	Disadvantages
FCFS	Nonpreemptive	Batch	1. Easy to implement. 2. Minimum overhead.	1. Unpredictable turn around time. 2. Average waiting is more.
RR	Preemptive	Interactive	1. Provides fair CPU allocation. 2. Provides reasonable response times to interactive users.	1. Requires selection of good time slice.
Priority	Nonpreemptive	Batch	1. Ensures fast completion of important jobs	1. Indefinite postponement of some jobs. 2. Faces starvation problem.
SJF	Nonpreemptive	Batch	1. Minimizes average waiting time. 2. SJF algorithm is optimal.	1. Indefinite postponement of some jobs. 2. Cannot be implemented at the level of short term scheduling. 3. Difficulty is knowing the length of the next CPU request.
Multilevel queues	Preemptive or nonpreemptive	Batcn/Interactive	1. Flexible. 2. Gives fair treatment to CPU bound jobs.	1. Overhead incurred by monitoring of queues.

Thread Scheduling

1. **Load Sharing**: Processes are not assigned to a particular processor. A global queue of threads is maintained. Each processor, when idle, selects a thread from this queue.
2. **Gang Scheduling**: A set of related threads is scheduled to run on a set of processors at the same time, on a 1-to-1 basis. Closely related threads / processes may be scheduled this way to reduce synchronization blocking, and minimize process switching. Group scheduling predated this strategy.
3. **Dedicated processor assignment**: Provides implicit scheduling defined by assignment of threads to processors. For the duration of program execution, each program is allocated a set of processors equal in number to the number of threads in the program. Processors are chosen from the available pool.
4. **Dynamic scheduling**: The number of thread in a program can be altered during the course of execution.