OS (CPU Scheduling)

★★ **Difference among short-term. Medium-term, long-term Scheduling [Scheduling levels]**

Sl	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a Job Scheduler	It is a CPU Scheduler	It is a Process swapping Scheduler
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler
3	It controls the degree of multiprogramming	It provides lesser control over the degree of multiprogramming	It reduces the degree of multiprogramming
4	It is almost absent/minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing system
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued

★★ **Difference between Preemptive and Non-Preemptive Scheduling [Scheduling levels]**

Basis for Comparison	**Preemptive Scheduling**	**Non Preemptive Scheduling**
Basic	A running process may be replaced by a higher-priority process at any time.	Once CPU allocated to a process, the process holds CPU until it (process) completes its burst/service time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

★★Preemptive and Non-Preemptive Scheduling Determination

CPU scheduling **decisions** may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, I/O request)

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)

3. When a process switches from the waiting state to the ready state (for example, completion of I/O)

4. When a process terminates

When scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **nonpreemptive;** otherwise, the scheduling scheme is **preemptive.**

★★Scheduling Criteria

The criteria include the following:

- **CPU utilization:** We want to keep the CPU as busy as possible. CPU utilization may range from 0 to 100 percent.
- **Throughput:** the number of jobs which are completed per time unit.
- **Turnaround Time:** The time between submission & completion.
- **Response Time:** the amount of time it takes to start responding.
- **Waiting time:** the amount of time a job spends waiting in the ready queue.

★★Scheduling Algorithm

There are seven popular process scheduling algorithms which we are going to discuss in this chapter:

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) or Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time First (SRTF) [or, preemptive algorithm]
- Round Robin(RR) Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

These algorithms are either **non-preemptive** or **preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

☆☆**First Come, First Served (FCFS)**

- Jobs are executed on first come, first served basis.
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance, as average wait time is high.

Example

Considering the following example,

Job	Arrival Time	Service time
1	1	8
2	2	2
3	3	1
4	4	2
5	5	5

If the jobs arrive in the above order and are served in FCFS order, we get the result shown in the following **Gantt chart**.

Job 1	Job 2	Job 3	Job 4	Job 5	
1	9	11	12	14	19

From the Chart we can get the following Table: **[we can use value directly from the Gantt chart; so that case following table is not needed.]**

Job	Time Arrive	Time Finish	Time Waiting	Turnaround Time
1	1	9	0	8
2	2	11	7	9
3	3	12	8	9
4	4	14	8	10
5	5	19	9	14

Turnaround time = Time _ Finish - time _arrive.

Waiting time = Starting execution of the respective job- arriving time of the respective job
= **Turnaround time – Service time**

So, the average turnaround time of process = $\frac{(8+9+9+10+14)}{5}$

Average wait time = $\frac{(0+7+8+8+9)}{5}$ = 32/5 = 6.4

☆☆Shortest Job First (SJF)

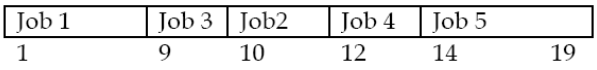
- This is also known as **Shortest-Job-Next (SJN)**.
- This is a non-preemptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where the required CPU time is not known.
- The processer should know in advance how much time a process will take.

Example 1:

Considering the following example,

Job	Arrival Time	Service time
1	1	8
2	2	2
3	3	1
4	4	2
5	5	5

If the jobs arrive in the above order and are served in SJF order, we get the result shown in the following Gantt chart. [As it is a non-preemptive scheduling algorithm, so Job1 is continues its execution and other jobs are sorted according to service time in ascending order.]



From the Chart we can get the following Table:

Job	Time Arrive	Time Finish	Time Waiting	Turnaround Time
1	1	9	0	8 (9-1)
2	2	12	8 (10-2)	10 (12-2)
3	3	10	6 (7-1)	7 (10-3)
4	4	14	8 (10-2)	10 (14-4)
5	5	19	9 (14-5)	14 (19-5)

i) Average turnaround time = (8+10+7+10+14)/5 = 9.8

ii) Average wait time = (0+8+6+8+9) /5 = 6.2

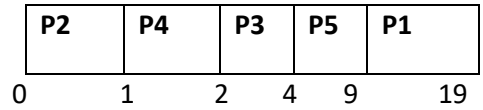
Example 2[When Priority is given]:

Process	Burst/ Service time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0

SJF is simply a priority algorithm where the lower the burst time, higher the priority. So, here priority column is ignored **because it does not follow SJF mechanism**. [We use low numbers to represent high priority.]

Using SJF scheduling, we would schedule these jobs according to the following Gantt chart,



From the chart, we get following table

Process	Arrival time	Finish time	Waiting time	Turnaround time
P1	0	19	9(19-10)	19
P2	0	1	0(1-1)	1
P3	0	4	2(4-2)	4
P4	0	2	1(2-1)	2
P5	0	9	4(9-5)	9

Average turnaround time= $\frac{19+1+4+2+9}{5} = \frac{35}{5}$ = 7 ms.

Average waiting time= $\frac{9+0+2+1+4}{5} = \frac{16}{5}$ = 3.2 ms.

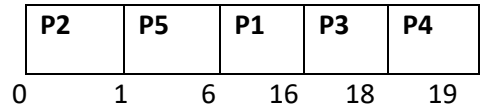
☆☆**Priority based scheduling**

- Non preemptive algorithm
- Each process is assigned a priority
- Processes with same priority are executed or first come first served basis
- Priority can be decided based on memory requirements, time requirements etc.

Process	Burst/Service time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0.

Using priority scheduling we would schedule there processes according to the following Gantt chart:



From the chart, we get following table

Process	Arrival time	Finish time	Waiting time	Turnaround time
P1	0	16	6(16-10)	16
P2	0	1	0(1-1)	1
P3	0	18	16(18-2)	18
P4	0	19	18(19-1)	19
P5	0	6	1(6-5)	6

Average turnaround time= $\frac{16+1+18+19+6}{5} = \frac{62}{5}$ = 12.4 ms.

Average waiting time= $\frac{6+0+16+18+1}{5} = \frac{41}{5}$ = 8.2 ms.

☆☆**Shortest Remaining Time First (SRTF) [or, preemptive algorithm]**

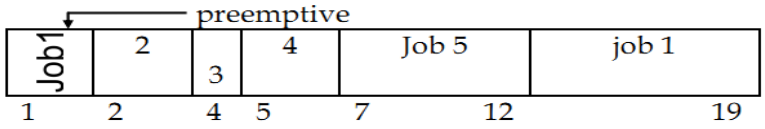
Shortest-Job-First may be either preemptive or non-preemptive. A preemptive shortest-Job-First algorithm will preempt the currently executing job, while a non-preemptive Shortest-Job-First algorithm will allow the currently running job to finish its CPU burst. Preemptive Shortest-Job-First is sometimes called Shortest-Remaining-Time-First (SRTF).

Example

As an example, consider the following 5 jobs.

Process	Arrival time	Burst/Service time
P1	1	8
P2	2	2
P3	3	1
P4	4	2
P5	5	5

We get the result shown in the following Gantt chart (Job 1 is preempted after 1 second.)



From the Chart we can get the following Table:

Process/Job	Arrival time	Finish time	Waiting time	Turnaround time
P1	1	19	10 (18-8)	18 (19-1)
P2	2	4	0 (2-2)	2 (4-2)
P3	3	5	1 (2-1)	2 (5-3)
P4	4	7	1 (3-2)	3 (7-4)
P5	5	12	2 (7-5)	7 (12-5)

i) Average turnaround time = (18+2+2+3+7)/5 = 6.4

ii) Average wait time = (10+0+1+1+2) /5 = 2.8

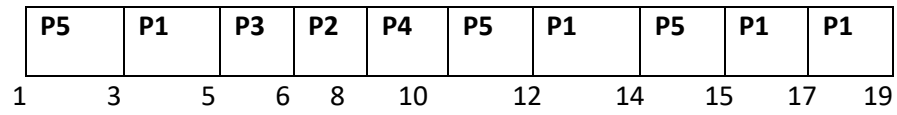
☆☆**Round Robin scheduling**

- Round Robin is a preemptive process scheduling algorithm
- Each process is provided a fix time to execute, it is called a quantum
- Once a process is executed for a given time period, it preempted and other process executes for a given time period.

Example:

Process	Arrival time	Service time	Priority
P1	1	8	2
P2	2	2	4
P3	3	1	3
P4	4	2	4
P5	5	5	1

Using **a quantum of 2**, Gantt chart is given below,



We get the following **table,**

Process	Arrival time	Finish time	Waiting time	Turnaround time
P1	1	19	10	18
P2	2	8	4	6
P3	3	6	2	3
P4	4	10	4	6
P5	5	15	5	10

Average turnaround time= $\frac{18+6+3+6+10}{5} = \frac{43}{5}$ = 8.6 ms

Average waiting time= $\frac{10+4+2+4+5}{5} = \frac{25}{5}$ = 5 ms

☆☆**Multilevel Queue Scheduling Algorithm [Or, Multiple-Level Queues Scheduling]**

Another class of scheduling algorithms 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 is not an independent scheduling algorithm. It makes use of other existing algorithms to group and schedule processes/jobs with common characteristics.

- 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.

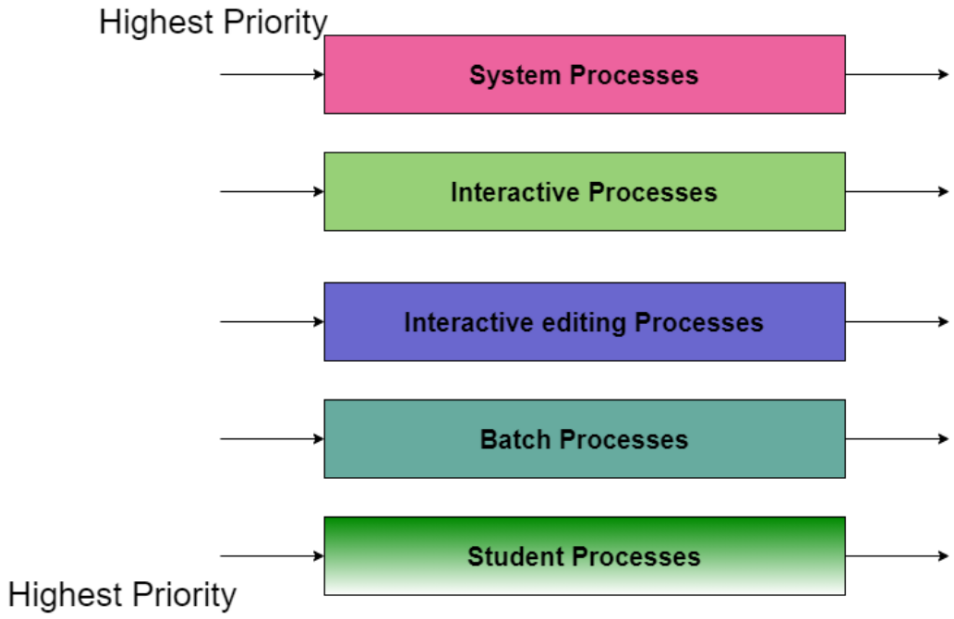
For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by the Round Robin algorithm, **while** the background queue is scheduled by an FCFS algorithm.

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 preempted.



In this case, if there are no processes on the higher priority queue only then the processes on the low priority queues will run. For **Example:** Once processes on the system queue, the Interactive queue, and Interactive editing queue become empty, only then the processes on the batch queue will run.

The description of the processes in the above diagram is as follows:

- **System Process**: The Operating system itself has its own process to run and is termed as System Process.
- **Interactive Process**: The Interactive Process is a process in which there should be the same kind of interaction (basically an online game).
- **Batch Processes**: Batch processing is basically a technique in the Operating system that collects the programs and data together in the form of the **batch** before the **processing** starts.
- **Student Process**: The system process always gets the highest priority while the student processes always get the lowest priority.

In an operating system, there are many processes, in order to obtain the result we cannot put all processes in a queue; thus this process is solved by Multilevel queue scheduling.

★★What advantage is there in having different time-quantum sizes on different levels of a multilevel queuing system?

Answer: Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, making more efficient use of the computer.

☆☆Multilevel Feedback Queue Scheduling

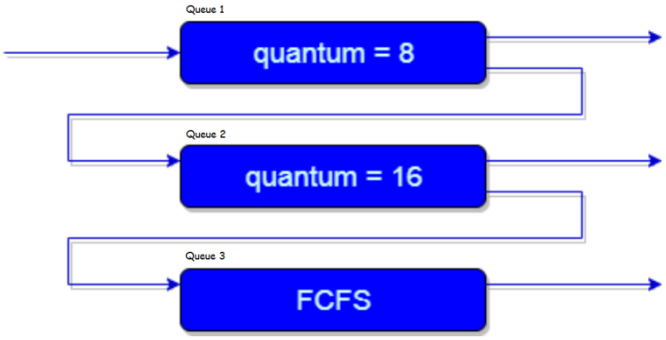
In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst(service time) 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.

An example of a multilevel feedback queue can be seen in the following figure.



Here is the **explanation:**

First of all, Suppose that queues 1 and 2 follow round robin with time quantum 8 and 16 respectively and queue 3 follows FCFS. One of the implementations of Multilevel Feedback Queue Scheduling is as follows:

1. If any process starts executing then firstly it enters **queue 1**.
2. In queue 1, the process executes for 8 unit and if it completes in these 8 units or it gives CPU for I/O operation in these 8 units unit then the priority of this process does not change, and if for some reasons it again comes in the ready queue than it again starts its execution in the Queue 1.
3. If a process that is in queue 1 does not complete in 8 units then its priority gets reduced and it gets shifted to **queue 2**.
4. Above points 2 and 3 are also true for processes in queue 2 but the time quantum is 16 units. Generally, if any process does not complete in a given time quantum then it gets shifted to the lower priority queue **i.e. queue 3** in this case..
5. After that in the last queue, all processes are scheduled in an FCFS manner.
6. It is important to note that a process that is in a lower priority queue can only execute only when the higher priority queues are empty.
7. Any running process in the lower priority queue can be interrupted by a process arriving in the higher priority queue.

In the above Implementation, there is a problem and that is; Any process that is in the lower priority queue has to suffer starvation due to some short processes that are taking all the CPU time.

And the solution to this problem is : There is a solution that is to boost the priority of all the process after regular intervals then place all the processes in the highest priority queue.

★★Algorithm Evaluation [How do we select a CPU-scheduling algorithm for a particular system?]

As we learn, there are many scheduling algorithms, each with its own parameters. As a result, selecting an algorithm can be difficult.

The first problem is defining the criteria to be used in selecting an algorithm. Criteria are often defined in terms of **CPU utilization**, response **time**, **or throughput**. To select an algorithm, we must first define the relative importance of these measures. Our criteria may include several measures, such as:

- Maximize CPU utilization under the constraint that the maximum response time is 1 second.
- Maximize throughput such that turnaround time is (on average) linearly proportional to total execution time.

Once the selection criteria have been defined, we want to evaluate the various algorithms under consideration. **There are several methods to evaluated** –

- Deterministic Modeling
- Queueing Models (queuing-network analysis)
- Simulation
- Implementation with real system

We here will discuss about **Deterministic Modeling.**

☆☆Deterministic Modeling

One major class of evaluation methods is called **analytic evaluation**. It uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload.

One type of analytic evaluation is deterministic modeling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload.

For example, assume that we have the workload shown in following figure. All five processes arrive at time 0, in the order given, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Consider the FCFS, SJF, and RR (quantum = 10 milliseconds) scheduling algorithms for this set of processes. **Which algorithm would give the minimum average waiting time?**

<Do the Math part briefly as shown in Scheduling Algorithm section.>

Reference: Page 173-174 [Textbook: Operating System Concepts 6e by Abraham Silberschatz]

In this case, the SJF policy results in less than one-half the average waiting time obtained with FCFS scheduling; the RR algorithm gives us an intermediate value.

Deterministic modeling is simple and fast. It gives exact numbers, allowing the algorithms to be compared. However, it requires exact numbers for input, and its answers apply to only those cases.