#### Process synchronization

###### Race conditions, Critical Sections and Semaphores are an key part of Operating systems. Details about these are given as follows:

**Race Condition**

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

**Critical Section**

* The critical section in a code segment where the shared variables can be accessed.
* Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.
* The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.
* The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.
* In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.
* Atomic action is required in a critical section i.e. only one process can execute in its critical section at a time.
* All the other processes have to wait to execute in their critical sections.

The critical section is given as follows:

do{

Entry Section Critical Section Exit Section Remainder Section

} while (TRUE);

In the above diagram, the entry sections handles the entry into the critical section. It acquires the resources needed for execution by the process. The exit section handles the exit from the critical section. It releases the resources and also informs the other processes that critical section is free.

The critical section problem needs a solution to synchronise the different processes. The solution to the critical section problem must satisfy the following conditions:

* **Mutual Exclusion**

Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

* **Progress**

Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

* **Bounded Waiting**

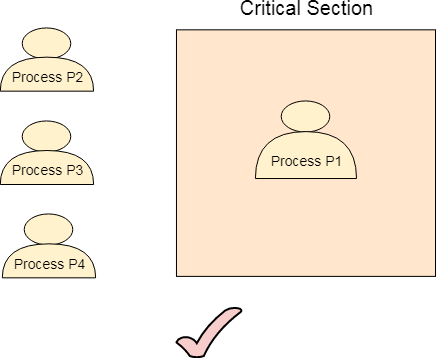
Bounded waiting means that each process must have a limited waiting time. Itt should not wait endlessly to access the critical section.

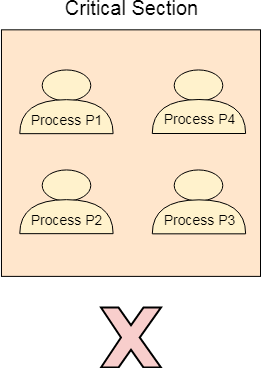
Requirements of Synchronization mechanisms

**PRIMARY**

1. **Mutual Exclusion**

Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.





1. **Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

**SECONDARY**

1. **Bounded Waiting**

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

1. **Architectural Neutrality**

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Various Times related to the Process

1. Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

1. Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

##### Completion Time

The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

##### Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

##### Waiting Time

The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

##### Response Time

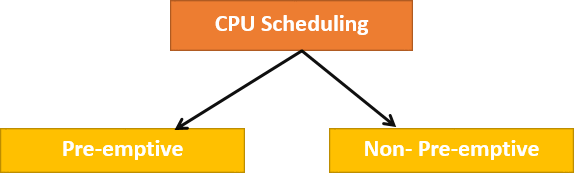
The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

## What is CPU Scheduling?

**CPU Scheduling** is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

**Types of CPU Scheduling**

Here are two kinds of Scheduling methods:



**Preemptive Scheduling**

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

**Non-Preemptive Scheduling**

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

**When scheduling is Preemptive or Non-Preemptive?**

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

* 1. A process switches from the running to the waiting state.
  2. Specific process switches from the running state to the ready state.
  3. Specific process switches from the waiting state to the ready state.
  4. Process finished its execution and terminated.

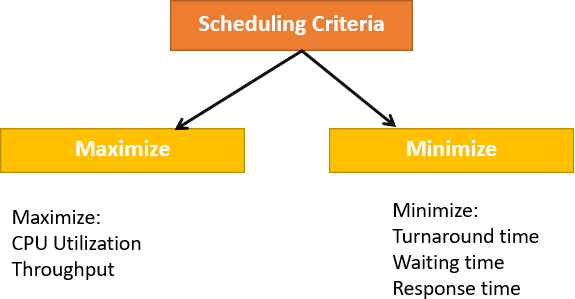
**Only conditions 1 and 4 apply, the scheduling is called non- preemptive. All other scheduling are preemptive.**

**Important CPU scheduling Terminologies**

* **Burst Time/Execution Time:** It is a time required by the process to complete execution. It is also called running time.
* **Arrival Time:** when a process enters in a ready state
* **Finish Time:** when process complete and exit from a system
* **Multiprogramming:** A number of programs which can be present in memory at the same time.
* **Jobs:** It is a type of program without any kind of user interaction.
* **User:** It is a kind of program having user interaction.
* **Process:** It is the reference that is used for both job and user.
* **CPU/IO burst cycle:** Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

**CPU Scheduling Criteria**

A CPU scheduling algorithm tries to maximize and minimize the following:



**Maximize:**

**CPU utilization:** CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

**Throughput:** The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

**Minimize:**

**Waiting time:** Waiting time is an amount that specific process needs to wait in the ready queue.

**Response time:** It is an amount to time in which the request was submitted until the first response is produced.

**Turnaround Time:** Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

**Key Differences Between Preemptive and Non-Preemptive Scheduling:**

1. In preemptive scheduling the CPU is allocated to the processes for the limited time whereas in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.
2. The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non- preemptive scheduling is not interrupted in the middle of execution and wait till its execution.
3. In Preemptive Scheduling, there is the overhead of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. Whereas in case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.
4. In preemptive scheduling, if a high priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process having larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling attain flexible by allowing the critical processes to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
6. The Preemptive Scheduling has to maintain the integrity of shared data that’s why it is cost associative as it which is not the case with Non-preemptive Scheduling.

**Comparison Chart:**

|  |  |  |
| --- | --- | --- |
| **PARAMENTER** | **PREEMPTIVE SCHEDULING** | **NON-PREEMPTIVE SCHEDULING** |
| Basic | In this resources(CPU Cycle)  are allocated to a process for a limited time. | Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting  state. |

|  |  |  |
| --- | --- | --- |
| **PARAMENTER** | **PREEMPTIVE SCHEDULING** | **NON-PREEMPTIVE SCHEDULING** |
| Interrupt | Process can be interrupted in between. | Process can not be interrupted untill it terminates itself or its time is up. |
| Starvation | If a process having high priority frequently arrives in the ready queue, low priority process may starve. | If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve. |
| Overhead | It has overheads of scheduling the processes. | It does not have overheads. |
| Flexibility | flexible | rigid |
| Cost | cost associated | no cost associated |

**TYPES OF SCHEDULING**

###### A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

* + First-Come, First-Served (FCFS) Scheduling
  + Shortest-Job-Next (SJN) Scheduling
  + Priority Scheduling
  + Shortest Remaining Time
  + Round Robin(RR) Scheduling
  + Multiple-Level Queues 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.

FCFS Scheduling

**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. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

* Simple
* Easy
* First come, First serv

### Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. 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

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 2 | 4 | 12 | 8 | 4 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 4 | 12 | 33 | 29 | 17 |

Avg Waiting Time=31/5

os FCFS Scheduling

**(Gantt chart)**

# Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, 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 of SJF

* 1. Maximum throughput
  2. Minimum average waiting and turnaround time

### Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

##### Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 7 | 8 | 7 | 0 |
| 2 | 3 | 3 | 13 | 10 | 7 |
| 3 | 6 | 2 | 10 | 4 | 2 |
| 4 | 7 | 10 | 31 | 24 | 14 |
| 5 | 9 | 8 | 21 | 12 | 4 |

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives).

According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

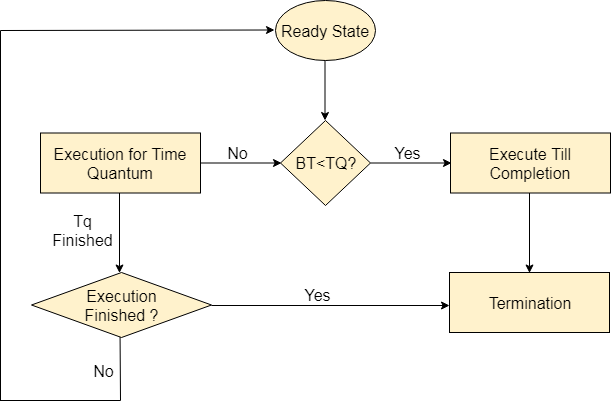
So that's how the procedure will go on in **shortest job first (SJF)** scheduling algorithm.

os SJF scheduling algorithm

Avg Waiting Time = 27/5

# Round Robin Scheduling Algorithm

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called 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

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

### Disadvantages

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

# RR Scheduling Example

In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

|  |  |  |
| --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** |
| 1 | 0 | 5 |
| 2 | 1 | 6 |

|  |  |  |
| --- | --- | --- |
| 3 | 2 | 3 |
| 4 | 3 | 1 |
| 5 | 4 | 5 |
| 6 | 6 | 4 |

According to the algorithm, we have to maintain the ready queue and the Gantt chart. The structure of both the data structures will be changed after every scheduling.

##### Ready Queue:

Initially, at time 0, process P1 arrives which will be scheduled for the time slice 4 units. Hence in the ready queue, there will be only one process P1 at starting with CPU burst time 5 units.

|  |
| --- |
| P1 |
| 5 |

##### GANTT chart

The P1 will be executed for 4 units first.

os RR Scheduling Example GANTT chart

##### Ready Queue

Meanwhile the execution of P1, four more processes P2, P3, P4 and P5 arrives in the ready queue. P1 has not completed yet, it needs another 1 unit of time hence it will also be added back to the ready queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P3 | P4 | P5 | P1 |
| 6 | 3 | 1 | 5 | 1 |

##### GANTT chart

After P1, P2 will be executed for 4 units of time which is shown in the Gantt chart.

os RR Scheduling Example GANTT chart 1

##### Ready Queue

During the execution of P2, one more process P6 is arrived in the ready queue. Since P2 has not completed yet hence, P2 will also be added back to the ready queue with the remaining burst time 2 units.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P3 | P4 | P5 | P1 | P6 | P2 |
| 3 | 1 | 5 | 1 | 4 | 2 |

##### GANTT chart

After P1 and P2, P3 will get executed for 3 units of time since its CPU burst time is only 3 seconds.

os RR Scheduling Example GANTT chart 2

##### Ready Queue

Since P3 has been completed, hence it will be terminated and not be added to the ready queue. The next process will be executed is P4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P4 | P5 | P1 | P6 | P2 |
| 1 | 5 | 1 | 4 | 2 |

##### GANTT chart

After, P1, P2 and P3, P4 will get executed. Its burst time is only 1 unit which is lesser then the time quantum hence it will be completed.

os RR Scheduling Example GANTT chart 3

##### Ready Queue

The next process in the ready queue is P5 with 5 units of burst time. Since P4 is completed hence it will not be added back to the queue.

|  |  |  |  |
| --- | --- | --- | --- |
| P5 | P1 | P6 | P2 |
| 5 | 1 | 4 | 2 |

##### GANTT chart

P5 will be executed for the whole time slice because it requires 5 units of burst time which is higher than the time slice.

os RR Scheduling Example GANTT chart 4

##### Ready Queue

P5 has not been completed yet; it will be added back to the queue with the remaining burst time of 1 unit.

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P6 | P2 | P5 |
| 1 | 4 | 2 | 1 |

##### GANTT Chart

The process P1 will be given the next turn to complete its execution. Since it only requires 1 unit of burst time hence it will be completed.

os RR Scheduling Example GANTT chart 5

##### Ready Queue

P1 is completed and will not be added back to the ready queue. The next process P6 requires only 4 units of burst time and it will be executed next.

|  |  |  |
| --- | --- | --- |
| P6 | P2 | P5 |
| 4 | 2 | 1 |

##### GANTT chart

P6 will be executed for 4 units of time till completion.

os RR Scheduling Example GANTT chart 6

##### Ready Queue

Since P6 is completed, hence it will not be added again to the queue. There are only two processes present in the ready queue. The Next process P2 requires only 2 units of time.

|  |  |
| --- | --- |
| P2 | P5 |
| 2 | 1 |

##### GANTT Chart

P2 will get executed again, since it only requires only 2 units of time hence this will be completed.

os RR Scheduling Example GANTT chart 7

##### Ready Queue

Now, the only available process in the queue is P5 which requires 1 unit of burst time. Since the time slice is of 4 units hence it will be completed in the next burst.

|  |
| --- |
| P5 |
| 1 |

##### GANTT chart

P5 will get executed till completion.

os RR Scheduling Example GANTT chart 8

The completion time, Turnaround time and waiting time will be calculated as shown in the table below. As, we know,

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 0 | 5 | 17 | 17 | 12 |
| 2 | 1 | 6 | 23 | 22 | 16 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 3 | 11 | 9 | 6 |
| 4 | 3 | 1 | 12 | 9 | 8 |
| 5 | 4 | 5 | 24 | 20 | 15 |
| 6 | 6 | 4 | 21 | 15 | 11 |

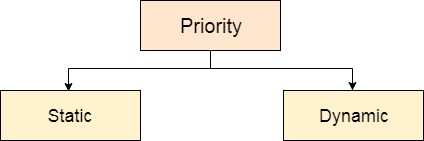
Avg Waiting Time = (12+16+6+8+15+11)/6 = 76/6 units

# 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 of priority scheduling algorithm exists. One is **Preemptive** priority scheduling while the other is **Non Preemptive** Priority scheduling.



The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called **static priority**, while if it keeps changing itself at the regular intervals, it is called **dynamic priority**.



Non Preemptive Priority Scheduling

In the 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. The people

might get confused with the priority numbers, hence in the GATE, there clearly mention which one is the highest priority and which one is the lowest one.

Example

In the Example, there are 7 processes P1, P2, P3, P4, P5, P6 and P7. Their priorities, Arrival Time and burst time are given in the table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process ID** | **Priority** | **Arrival Time** | **Burst Time** |
| 1 | 2 | 0 | 3 |
| 2 | 6 | 2 | 5 |
| 3 | 3 | 1 | 4 |
| 4 | 5 | 4 | 2 |
| 5 | 7 | 6 | 9 |
| 6 | 4 | 5 | 4 |
| 7 | 10 | 7 | 10 |

We can prepare the Gantt chart according to the Non Preemptive priority scheduling.

The Process P1 arrives at time 0 with the burst time of 3 units and the priority number 2. Since No other process has arrived till now hence the OS will schedule it immediately.

Meanwhile the execution of P1, two more Processes P2 and P3 are arrived. Since the priority of P3 is 3 hence the CPU will execute P3 over P2.

Meanwhile the execution of P3, All the processes get available in the ready queue. The Process with the lowest priority number will be given the priority. Since P6 has priority number assigned as 4 hence it will be executed just after P3.

After P6, P4 has the least priority number among the available processes; it will get executed for the whole burst time.

Since all the jobs are available in the ready queue hence All the Jobs will get executed according to their priorities. If two jobs have similar priority number assigned to them, the one with the least arrival time will be executed.

os Non Preemptive Priority Scheduling

From the GANTT Chart prepared, we can determine the completion time of every process. The turnaround time, waiting time and response time will be determined.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Process Id** | **Priority** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** | **Response Time** |
| 1 | 2 | 0 | 3 | 3 | 3 | 0 | 0 |
| 2 | 6 | 2 | 5 | 18 | 16 | 11 | 13 |
| 3 | 3 | 1 | 4 | 7 | 6 | 2 | 3 |
| 4 | 5 | 4 | 2 | 13 | 9 | 7 | 11 |
| 5 | 7 | 6 | 9 | 27 | 21 | 12 | 18 |
| 6 | 4 | 5 | 4 | 11 | 6 | 2 | 7 |
| 7 | 10 | 7 | 10 | 37 | 30 | 18 | 27 |

Avg Waiting Time = (0+11+2+7+12+2+18)/7 = 52/7 units

Preemptive Priority Scheduling

In Preemptive 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.

The difference between preemptive priority scheduling and non preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.

Example

There are 7 processes P1, P2, P3, P4, P5, P6 and P7 given. Their respective priorities, Arrival Times and Burst times are given in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Id** | **Priority** | **Arrival Time** | **Burst Time** |
| 1 | 2(L) | 0 | 1 |
| 2 | 6 | 1 | 7 |
| 3 | 3 | 2 | 3 |
| 4 | 5 | 3 | 6 |
| 5 | 4 | 4 | 5 |
| 6 | 10(H) | 5 | 15 |
| 7 | 9 | 15 | 8 |

GANTT chart Preparation

At time 0, P1 arrives with the burst time of 1 units and priority 2. Since no other process is available hence this will be scheduled till next job arrives or its completion (whichever is lesser).

os Preemptive Priority Scheduling GANTT chart Preparation

At time 1, P2 arrives. P1 has completed its execution and no other process is available at this time hence the Operating system has to schedule it regardless of the priority assigned to it.

os Preemptive Priority Scheduling GANTT chart Preparation 1

The Next process P3 arrives at time unit 2, the priority of P3 is higher to P2. Hence the execution of P2 will be stopped and P3 will be scheduled on the CPU.

os Preemptive Priority Scheduling GANTT chart Preparation 2

During the execution of P3, three more processes P4, P5 and P6 becomes available. Since, all these three have the priority lower to the process in execution so PS can't preempt the process. P3 will complete its execution and then P5 will be scheduled with the priority highest among the available processes.

os Preemptive Priority Scheduling GANTT chart Preparation 3

Meanwhile the execution of P5, all the processes got available in the ready queue. At this point, the algorithm will start behaving as Non Preemptive Priority Scheduling. Hence now, once all the processes get available in the ready queue, the OS just took the process with the highest priority and execute that process till completion. In this case, P4 will be scheduled and will be executed till the completion.

os Preemptive Priority Scheduling GANTT chart Preparation 4

Since P4 is completed, the other process with the highest priority available in the ready queue is P2. Hence P2 will be scheduled next.

os Preemptive Priority Scheduling GANTT chart Preparation 5

P2 is given the CPU till the completion. Since its remaining burst time is 6 units hence P7 will be scheduled after this.

os Preemptive Priority Scheduling GANTT chart Preparation 6

The only remaining process is P6 with the least priority, the Operating System has no choice unless of executing it. This will be executed at the last.

os Preemptive Priority Scheduling GANTT chart Preparation 7

The Completion Time of each process is determined with the help of GANTT chart. The turnaround time and the waiting time can be calculated by the following formula.

1. Turnaround Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process Id** | **Priority** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn around Time** | **Waiting Time** |
| 1 | 2 | 0 | 1 | 1 | 1 | 0 |
| 2 | 6 | 1 | 7 | 22 | 21 | 14 |
| 3 | 3 | 2 | 3 | 5 | 3 | 0 |
| 4 | 5 | 3 | 6 | 16 | 13 | 7 |

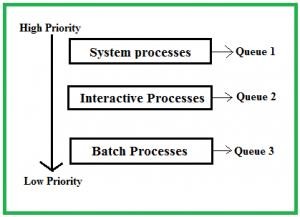
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 4 | 4 | 5 | 10 | 6 | 1 |
| 6 | 10 | 5 | 15 | 45 | 40 | 25 |
| 7 | 9 | 6 | 8 | 30 | 24 | 16 |

Avg Waiting Time = (0+14+0+7+1+25+16)/7 = 63/7 = 9 units

# Multilevel Queue (MLQ) CPU Scheduling

Prerequisite : [CPU Scheduling](https://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/) It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a **foreground (interactive)** process and **background (batch)** processes.These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.Now, let us see how it works.

**Ready Queue** is divided into separate queues for each class of processes. For example, let us take three different types of process System processes, Interactive processes and Batch Processes. All three process have there own queue. Now,look at the below figure.

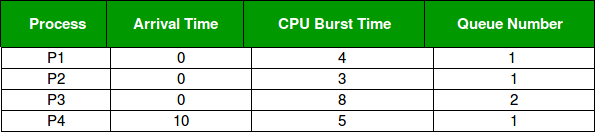


All three different type of processes have there own queue. Each queue have its own Scheduling algorithm. For example, queue 1 and queue 2 uses **Round Robin** while queue 3 can use **FCFS** to schedule there processes.

**Scheduling among the queues :** What will happen if all the queues have some processes? Which process should get the cpu? To determine this Scheduling among the queues is necessary. There are two ways to do so –

* 1. **Fixed priority preemptive scheduling method –** Each queue has absolute priority over lower priority queue. Let us consider following priority order **queue 1 > queue 2 > queue 3**.According to this algorithm no process in the batch queue(queue 3) can run unless queue 1 and 2 are empty. If any batch process (queue 3) is running and any system (queue 1) or Interactive process(queue 2) entered the ready queue the batch process is preempted.
  2. **Time slicing** – In this method each queue gets certain portion of CPU time and can use it to schedule its own processes.For instance, queue 1 takes 50 percent of CPU time queue 2 takes 30 percent and queue 3 gets 20 percent of CPU time.

**Example Problem :** Consider below table of four processes under Multilevel queue scheduling.Queue number denotes the queue of the process.



Priority of queue 1 is greater than queue 2. queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS.

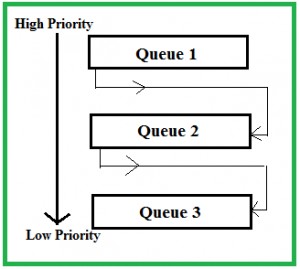
Below is the **gantt chart** of the problem :

At starting both queues have process so process in queue 1 (P1, P2) runs first (because of higher priority) in the round robin fashion and completes after 7 units then process in queue 2 (P3) starts running (as there is no process in queue 1) but while it is running P4 comes in queue 1 and interrupts P3 and start running for 5 second and after its completion P3 takes the CPU and completes its execution.

Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling

**Prerequisite –** [CPU Scheduling](https://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/), [Multilevel Queue Scheduling](https://www.geeksforgeeks.org/operating-system-multilevel-queue-scheduling/)

This Scheduling is like Multilevel Queue(MLQ) Scheduling but in this process can move between the queues. **Multilevel Feedback Queue Scheduling (MLFQ)** keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.Now, look at the diagram and explanation below to understand it properly.



Now let us suppose that queue 1 and 2 follow round robin with time quantum 4 and 8 respectively and queue 3 follow FCFS.One implementation of MFQS is given below –

1. When a process starts executing then it first enters queue 1.
2. In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.
3. If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit.In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in FCFS manner.
6. A process in lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

Well, above implementation may differ for example the last queue can also follow Round-robin Scheduling.

**Problems in the above implementation –** A process in the lower priority queue can suffer from starvation due to some short processes taking all the CPU time. **Solution –** A simple solution can be to boost the priority of all the process after regular intervals and place them all in the highest priority queue.

**What is the need of such complex Scheduling?**

* Firstly, it is more flexible than the multilevel queue scheduling.
* To optimize turnaround time algorithms like SJF is needed which require the running time of processes to schedule them. But the running time of the process is not known in advance. MFQS runs a process for a time quantum and then it can change its priority(if it is a long process). Thus it learns from past behavior of the process and then predicts its future behavior.This way it tries to run shorter process first thus optimizing turnaround time.
* MFQS also reduces the response time.

**Example –**

Consider a system which has a CPU bound process, which requires the burst time of 40 seconds.The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum ‘2’ seconds and in each level it is incremented by ‘5’ seconds.Then how many times the process will be interrupted and on which queue the process will terminate the execution?

**Solution –**

Process P needs 40 Seconds for total execution.

At Queue 1 it is executed for 2 seconds and then interrupted and shifted to queue 2. At Queue 2 it is executed for 7 seconds and then interrupted and shifted to queue 3. At Queue 3 it is executed for 12 seconds and then interrupted and shifted to queue 4. At Queue 4 it is executed for 17 seconds and then interrupted and shifted to queue 5. At Queue 5 it executes for 2 seconds and then it completes.

Hence the process is interrupted 4 times and completes on queue 5.