CPU Scheduling

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

Types of CPU Scheduling

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 I/O request or invocation of wait for the termination of one of the child processes).
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**.
5. In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.
6. When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise, the scheduling scheme is **preemptive**.
7. Non-Preemptive Scheduling
8. Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
9. This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.
10. It is the only method that can be used on certain hardware platforms because It does not require the special hardware(for example a timer) needed for preemptive scheduling.
11. In non-preemptive scheduling, it does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then after that it can allocate the CPU to any other process.

12. Some Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non-preemptive) Scheduling and Priority (non- preemptive version) Scheduling, etc.

Some Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non-preemptive) Scheduling etc.

Process	Arrival time	CPU Burst Time (in millisecond
P0	2	8
P1	3	6
P2	0	9
P3	1	4

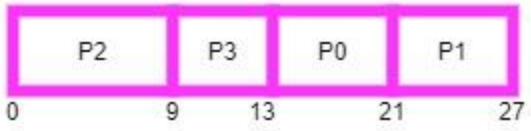


Figure: Non-Preemptive Scheduling

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Thus this type of scheduling is used mainly when a process switches either from running state to ready state or from waiting state to ready state. The resources (that is CPU cycles) are mainly allocated to the process for a limited amount of time and then are taken away, and after that, the process is again placed back in the ready queue in the case if that process still has a CPU burst time remaining. That process stays in the ready queue until it gets the next chance to execute.

Some Algorithms that are based on preemptive scheduling are Round Robin Scheduling (RR), Shortest Remaining Time First (SRTF).etc.

Process	Arrival time	CPU Burst Time (in millisecond
P0	2	3
P1	3	5
P2	0	6
P3	1	5

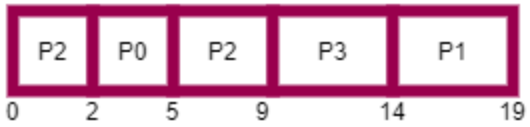


Figure: Preemptive Scheduling

CPU Scheduling: Scheduling Criteria

There are many different criteria to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization

To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit of time or rather says the total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Round Robin(RR) Scheduling

First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

- First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- This is used in Batch Systems.
- It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.
- A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

Calculating Average Waiting Time

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge it's performance.

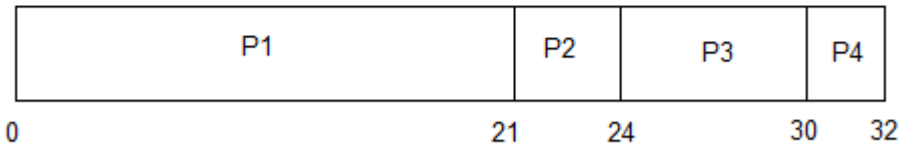
AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

Lower the Average Waiting Time, better the scheduling algorithm.

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given proccesses, first **P1** will be provided with the CPU resources,

- Hence, waiting time for **P1** will be 0
- **P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms
- Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms.

- For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

Problems with FCFS Scheduling

Below we have a few shortcomings or problems with the FCFS scheduling algorithm:

1. It is **Non Pre-emptive** algorithm, which means the **process priority** doesn't matter.

   If a process with very least priority is being executed, more like **daily routine backup** process, which takes more time, and all of a sudden some other high priority process arrives, like **interrupt to avoid system crash**, the high priority process will have to wait, and hence in this case, the system will crash, just because of improper process scheduling.

2. Not optimal Average Waiting Time.
3. Resources utilization in parallel is not possible, which leads to **Convoy Effect**, and hence poor resource(CPU, I/O etc) utilization.

What is Convoy Effect?

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

This essentially leads to poort utilization of resources and hence poor performance.

Shortest Job First(SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.

- This is used in Batch Systems.
- It is of two types:
    1. Non Pre-emptive
    2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non Pre-emptive Shortest Job First

Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :

Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

The average Turn around time = (32+5+11+2) / 4= 12.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

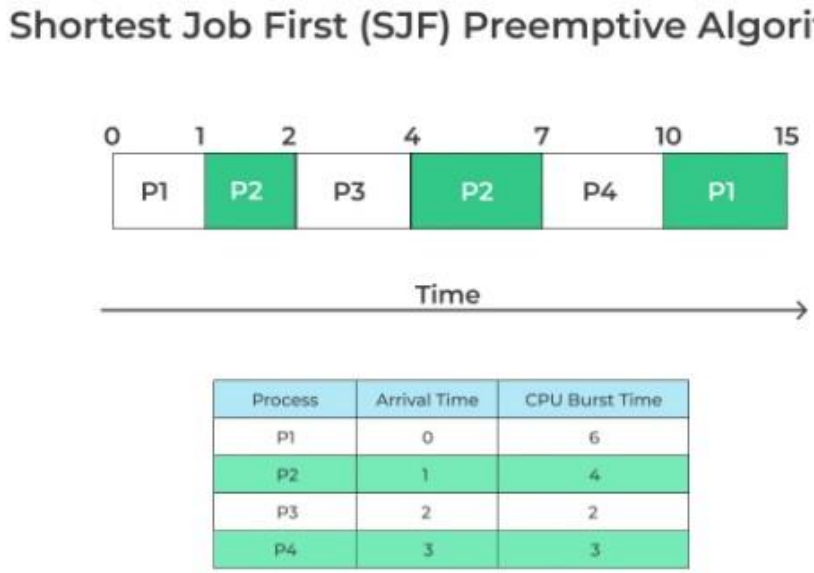
Problem with Non Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.



Process	Arrival Time	CPU Burst Time
P1	0	6
P2	1	4
P3	2	2
P4	3	3

At (t =0ms), P1 arrives. It's the only process so CPU starts executing it.

At (t = 1ms), P2 has arrived . At this time, P1 (remaining time) = 5 ms . P2 has 4ms , so as P2 is shorter, P1 is preempted and P2 process starts executing.

At (t = 2ms), P3 process has arrived. At this time, P1(remaining time) = 5ms, P2(remaining time) = 3 ms , P3 = 2ms. Since P3 is having least burst time, P3 is executed .

At (t = 3ms), P4 comes , At this time, P1 = 5ms, P2 = 3ms, P3 = 1ms, P4 = 3ms. Since P4 does not have short burst time, so P3 continues to execute.

At (t= 4ms),P3 is finished . Now, remaining tasks are P1 = 5ms, P2 = 3ms, P4 = 3ms. As ,P2 and P4 have same time, so the task which came first will be executed first. So, P2 gets executed first.

At (t = 7ms),P4 gets executed for 3ms.

At (t = 10ms), P1 gets executed till it finishes.

Turn around time= Completion time – Arrival Time

P1 = (15-0) = 15ms

P2 = (7-1) = 6ms

P3 = (4-2)=2ms

P4 = (10-3)= 7ms

The average Turn around time is (15 +6 + 2 + 7)/4 = 30/4 = 7.5

Waiting time = Turn around time – Burst Time

P1 waiting time = (15-6) = 9ms

P2 waiting time = (6-4) = 2ms

P3 waiting time = (2-2)=0ms

P4 waiting time = (7-3)= 4ms

The average waiting time is (9 +2 + 0 + 4)/4 = 15/4 = 3.75

Round Robin Scheduling

Round Robin(RR) scheduling algorithm is mainly designed for time-sharing systems. This algorithm is similar to FCFS scheduling, but in Round Robin(RR) scheduling, preemption is added which enables the system to switch between processes.

- A fixed time is allotted to each process, called a **quantum**, for execution.
- Once a process is executed for the given time period that process is preempted and another process executes for the given time period.
- Context switching is used to save states of preempted processes.
- This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU.
- It is important to note here that the length of time quantum is generally from 10 to 100 milliseconds in length.

Some important characteristics of the Round Robin(RR) Algorithm are as follows:

1. Round Robin Scheduling algorithm resides under the category of Preemptive Algorithms.
2. This algorithm is one of the oldest, easiest, and fairest algorithm.
3. This Algorithm is a real-time algorithm because it responds to the event within a specific time limit.
4. In this algorithm, the time slice should be the minimum that is assigned to a specific task that needs to be processed. Though it may vary for different operating systems.
5. This is a hybrid model and is clock-driven in nature.
6. This is a widely used scheduling method in the traditional operating system.

Important terms

1. **Completion Time**
   It is the time at which any process completes its execution.
2. **Turn Around Time**
   This mainly indicates the time Difference between completion time and arrival time. The Formula to calculate the same is: **Turn Around Time = Completion Time – Arrival Time**
3. **Waiting Time(W.T):**
   It Indicates the time Difference between turn around time and burst time.
   And is calculated as **Waiting Time = Turn Around Time – Burst Time**

Let us now cover an example for the same:

Process	P1	P2	P3	P4
Burst Time	10	2	5	6

Here, we have taken quantum as 3ms. So,each process is assigned 3ms before switching to next process.

P1 is executed for 3ms. The, P2 is executed. As, P2 is only 2ms long, So after 2 ms , next process occupies the CPU.

P3 then is executed for 3ms. So, P3 remaining = 2ms. Then , P4 is executed for 3ms. P4 remaining time = 3ms

P1 is again executed for 3ms. P1 remaining time = 4ms

P3 is executed for 2ms. Then, P4 is executed for 3ms.

Now remaining process is P1 only . It executed for 3 ms. After that, Since P2,P3 and P4 are already finished, P1 is executed again for remaining time (1ms).

Ready Queue: P1,P2,P3,P4,P1,P3,P4,P1,P1

Let’s calculate Average Turn around time for each process –

P1 = 23-0 = 23ms

P2 = 5-0=5ms

P3 = 16-0 = 16ms

P4 = 19-0 = 19ms

Total average turn around time = (23 +5 +16+19)/4 = 15.75ms

Let’s calculate Average Waiting time for each process –

P1 = 23-10 = 13ms

P2 =5-2= 3ms

P3 = 16-5 = 11ms

P4 = 19-6 = 13ms

Total average waiting time = (13 +3 +11+13)/4 = 10ms