Table of Contents

[CPU SCHEDULING 2](#_Toc130476337)

[CPU-I/O CYCLE 2](#_Toc130476338)

[CPU Schedulers 2](#_Toc130476339)

[Preemptive vs Non-Preemptive 3](#_Toc130476340)

[Dispatcher 3](#_Toc130476341)

[Scheduling Criteria 3](#_Toc130476342)

[SCHEDULING ALGORITHMS 5](#_Toc130476343)

[1. First Come – First Serve Scheduling 5](#_Toc130476344)

[2. Shortest Job – First Scheduling 5](#_Toc130476345)

[3. PRIORITY SCHEDULING 5](#_Toc130476346)

[4. ROUND ROBIN (RR) SCHEDULING 6](#_Toc130476347)

[5. MULTILEVEL QUEUE SCHEDULING 7](#_Toc130476348)

[6. Multilevel Feedback Queue Scheduling 7](#_Toc130476349)

# CPU SCHEDULING

CPU scheduling is a process in an operating system where the CPU (central processing unit) is switched between different processes to improve the computer's efficiency. The idea is to avoid the CPU sitting idle and use it efficiently. There are different strategies called CPU scheduling algorithms, and each one works better in specific situations. The main goal is to keep the CPU busy and get the best performance out of it.

## CPU-I/O CYCLE

During the execution of a process, it spends time doing work on the CPU and waiting for I/O operations to complete. This is called the CPU-I/O burst cycle.

CPU burst, where it executes instructions on the CPU, and I/O burst, where it waits for input/output operations to complete. These cycles repeat until the process finishes. The efficiency of this process is maximized if the CPU and I/O cycles are used optimally.

## CPU Schedulers

A CPU scheduler selects a process from a group of ready-to-run processes in memory and allocates the CPU to it. Scheduling decisions happen when processes switch between states, such as running, waiting, and ready. The scheduling can be preemptive or non-preemptive depending on the state transition.

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

Scheduling under 1 and 4 is **non-preemptive**

All other scheduling is **preemptive – implications for data sharing between threads/processes**

## Preemptive vs Non-Preemptive

Preemptive and non-preemptive scheduling are two different ways of CPU scheduling. Preemptive scheduling takes the CPU away from a process before it completes, while non-preemptive scheduling allows the process to continue using the CPU until it finishes or goes into a waiting state. The choice between them depends on the situation and can be used together in different scenarios.

## Dispatcher

The dispatcher module is responsible for giving control of the CPU to the process selected by the scheduler. This involves switching context, switching to user mode, and jumping to the proper location in the user program to restart that program.

Dispatch latency is the time it takes for the dispatcher to stop one process and start another process running

## Scheduling Criteria

Scheduling criteria help to evaluate the different scheduling algorithms' efficiency. The five criteria are CPU utilization, throughput, turnaround time, waiting time, and response time.

1. CPU Utilization: This refers to the percentage of time that the CPU is being used to execute processes. Scheduling algorithms aim to keep the CPU busy as much as possible to increase overall system performance.

2. Throughput: This refers to the number of processes that are completed over a certain time period. High throughput means that the system is able to handle a large number of processes at once, improving overall system efficiency.

3. Turnaround Time: This measures the time it takes for a process to complete from the moment it is submitted to the moment it is finished. It includes the actual processing time as well as any time spent waiting in queues.

4. Waiting Time: This measures the amount of time that a process has been waiting in the ready queue, waiting for the CPU to become available for processing. Minimizing waiting time is an important goal of scheduling algorithms to ensure that processes are completed as quickly as possible.

5. Response Time: This measures the time it takes for the system to produce a response to a user's request, such as a keystroke or button click.

# SCHEDULING ALGORITHMS

## First Come – First Serve Scheduling

First Come First Served (FCFS) is a simple scheduling algorithm that allocates the CPU to the process that requests it first. It uses a FIFO queue but can result in long waiting times for processes. FCFS is non-preemptive and can delay small processes if they arrive after a big process. Overall, it is not an efficient algorithm.

## Shortest Job – First Scheduling

Shortest Job First (SJF) is a scheduling algorithm where the process with the smallest burst time is executed first. This algorithm minimizes the average waiting time for a given set of processes. It can be either non-preemptive or preemptive. In the non-preemptive version, once a process is scheduled, it runs to completion, while in the preemptive version, if a new process with a smaller burst time arrives, the currently running process is interrupted and the new process is scheduled to run. Preempted SJF can potentially suffer from starvation, where a long process never gets a chance to run if short processes keep arriving.

## PRIORITY SCHEDULING

Priority scheduling is a CPU scheduling algorithm where each process is assigned a priority and the CPU is allocated to the process with the highest priority. Priority scheduling can be either preemptive or non-preemptive.

Non-preemptive priority scheduling allows a lower priority process to continue executing until it completes, even if a higher priority process becomes available. Preemptive priority scheduling allows a higher priority process to interrupt the execution of a lower priority process.

Priority can be assigned based on various factors, such as the process type, the amount of CPU time required, the process deadline, or the process importance. However, priority scheduling can lead to starvation of lower-priority processes if higher-priority processes continuously request CPU time.

To prevent starvation, priority scheduling algorithms may incorporate aging, which gradually increases the priority of a process that has been waiting for a long time, ensuring that eventually, it will have a higher priority than other processes.

In priority scheduling, priorities can be set either internally or externally.

Internal priority is set by the operating system based on the process characteristics or the system state. For example, the operating system may assign a higher priority to an interactive process such as a GUI application to ensure that the user interface is responsive. Alternatively, the operating system may give higher priority to I/O-bound processes over CPU-bound processes to improve system performance.

External priority, on the other hand, is set by a user or an administrator. In this case, the priority of a process is set explicitly by the user or the administrator. For example, a user may assign a higher priority to a video rendering task to ensure that it completes as quickly as possible, while another user may assign a lower priority to a backup s to minimize its impact on system performance.

## ROUND ROBIN (RR) SCHEDULING

* Round Robin (RR) scheduling is a popular CPU scheduling algorithm that is commonly used in time-sharing systems.
* RR scheduling algorithm allocates a fixed time slice, called quantum or time slice, to each process in a cyclic way and ensures fair distribution of the CPU time.
* When a process arrives in the ready queue, it is added to the end of the queue, and the scheduler selects the first process in the queue to execute.
* The selected process is then given a quantum of time to execute. If the process completes execution before the quantum expires, it is moved to the completed queue. If the quantum expires before the process completes execution, the process is preempted and moved to the end of the ready queue.
* The value of the time quantum is usually between 10-100 milliseconds.
* RR scheduling is preferred over other scheduling algorithms because it provides reasonable response times and throughput while being fair to all processes.
* However, the performance of the RR algorithm can degrade if the time quantum is too large or too small. A small quantum size results in more context switches, while a large quantum size may result in poor response times.
* To optimize the performance of the RR algorithm, the value of the time quantum should be carefully chosen based on the characteristics of the system and the processes running on it.

## MULTILEVEL QUEUE SCHEDULING

Multilevel Queue Scheduling is a scheduling algorithm where the ready queue is partitioned into multiple separate queues, with each queue having its own scheduling algorithm.

In Multilevel Queue Scheduling, processes are categorized into different queues based on their attributes such as priority, memory requirements, and CPU burst time. Each queue has its own scheduling algorithm and priority level. Processes in higher priority queues are executed first, and if there are no processes in the highest priority queue, the scheduler moves to the next lower priority queue. This scheduling algorithm is useful for real-time systems and for systems with different types of processes.

## Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling is an extension of Multilevel Queue Scheduling where a process can move between queues based on its behavior and needs.

This algorithm uses feedback from the previous execution of the process to determine its next queue placement. A process that uses too much CPU time can be moved to a lower priority queue, while a process that suffers starvation can be moved to a higher priority queue. The objective is to achieve better turnaround time and response time for processes.

In summary, Multilevel Queue Scheduling and Multilevel Feedback Queue Scheduling are similar in that they divide processes into different queues with different priorities. However, Multilevel Feedback Queue Scheduling allows processes to move between queues based on recent behavior, while Multilevel Queue Scheduling does not.