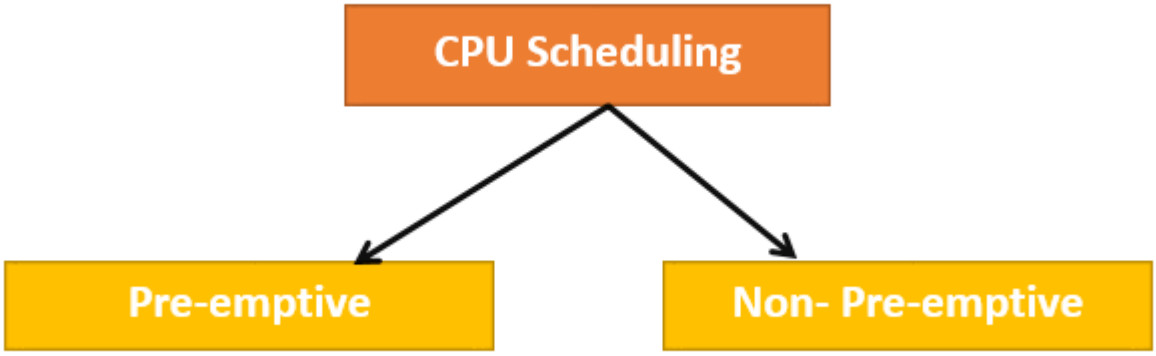
# What is CPU Scheduling?

**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.




## 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.**

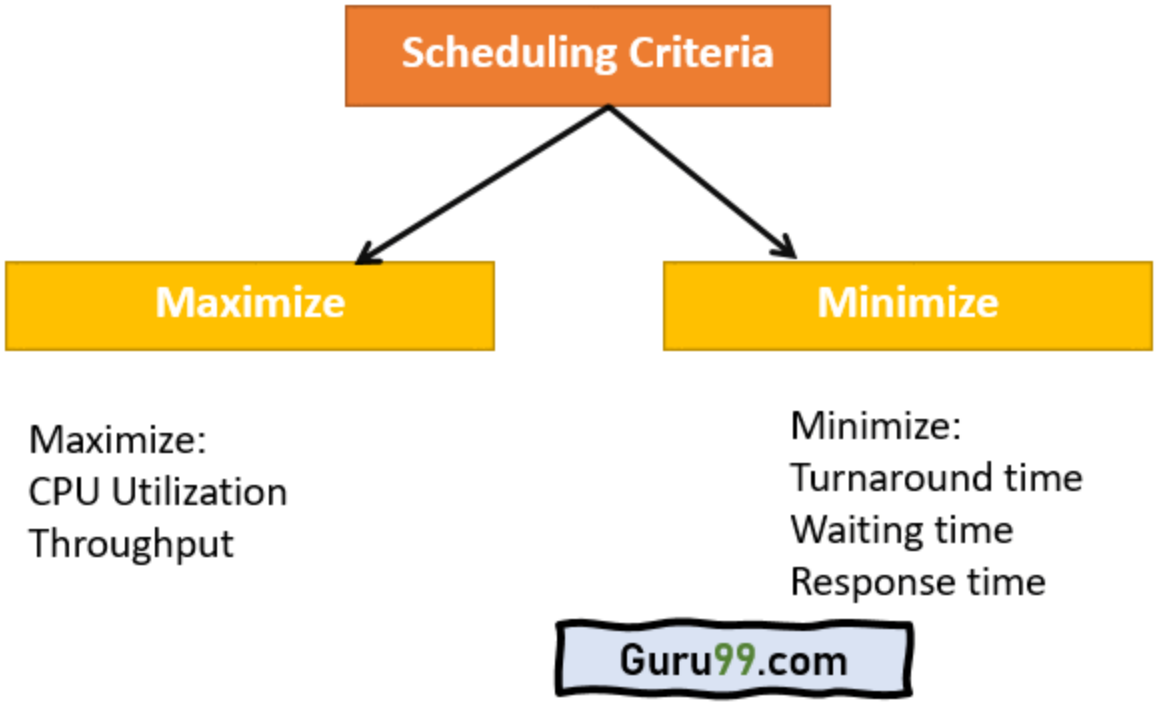
**All other scheduling are preemptive.**

# 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.

### Interval Timer

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

### What is Dispatcher?

It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

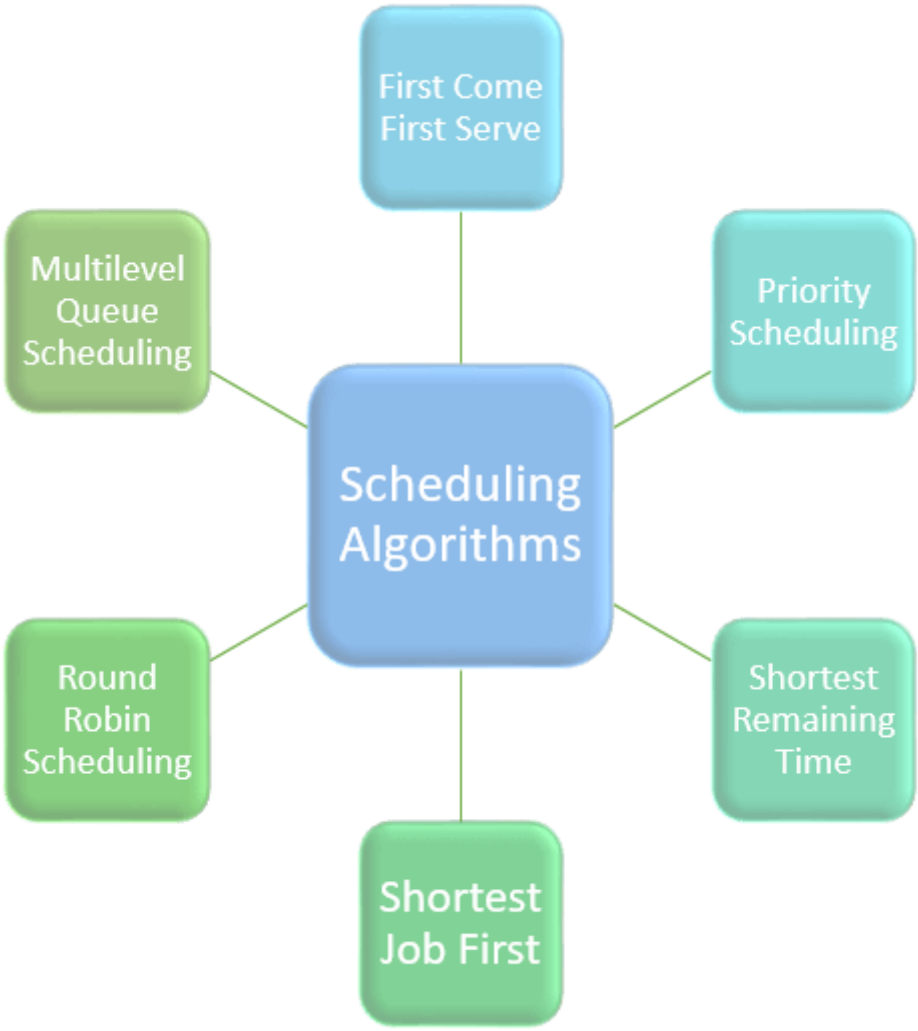
Functions performed by Dispatcher:

- Context Switching
- Switching to user mode
- Moving to the correct location in the newly loaded program.

## Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling




## Problems in concurrent processes

- **Sharing global resources –**
  Sharing of global resources safely is difficult. If two processes both make use of a global variable and both perform read and write on that variable, then the order in which various read and write are executed is critical.
- **Optimal allocation of resources –**
  It is difficult for the operating system to manage the allocation of resources optimally.
- **Locating programming errors –**
  It is very difficult to locate a programming error because reports are usually not reproducible.

- **Locking the channel –**
  It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

## The Critical Section Problem

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.

## 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.

2. **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.

2. **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.

## Mutual Exclusion in Synchronization

During concurrent execution of processes, processes need to enter the critical section (or the section of the program shared across processes) at times for execution. It might so happen that because of the execution of multiple processes at once, the values stored in the critical section become inconsistent. In other words, the values depend on the sequence of execution of instructions – also known as a race condition. The primary task of process synchronization is to get rid of race conditions while executing the critical section.
This is primarily achieved through mutual exclusion.

**Mutual exclusion** is a property of process synchronization which states that "no two processes can exist in the critical section at any given point of time". The term was first coined by Djikstra. Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.

## Process Synchronization

**Process Synchronization** is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

This can lead to the inconsistency of shared data. So the change made by one process not necessarily reflected when other processes accessed the same shared data. To avoid this type of inconsistency of data, the processes need to be synchronized with each other.

## Producer-Consumer problem

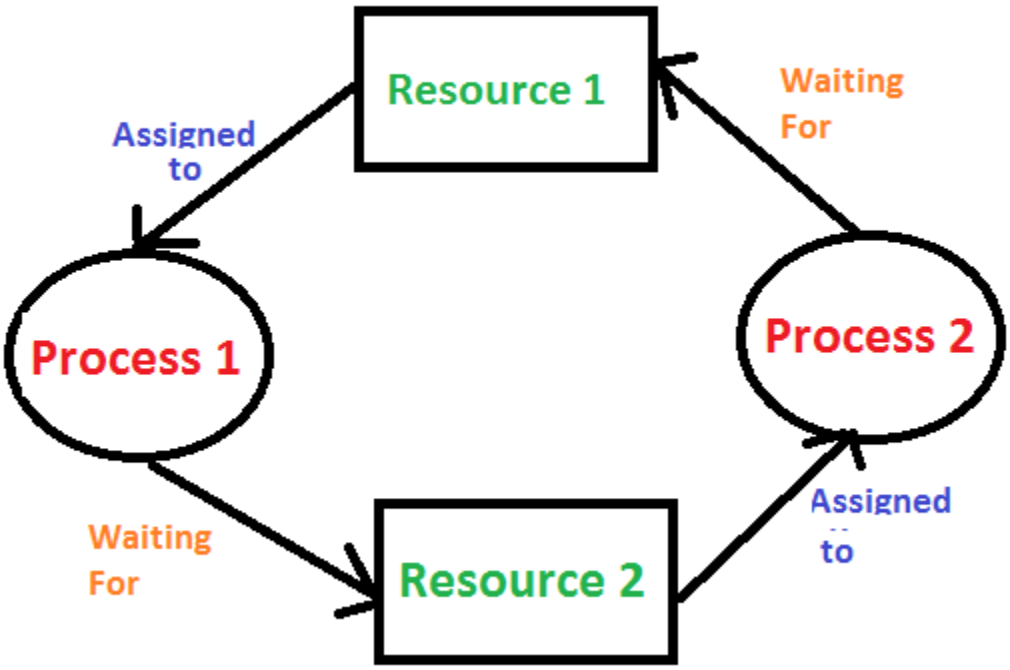
The Producer-Consumer problem is a classical multi-process synchronization problem, that is we are trying to achieve synchronization between more than one process.

There is one Producer in the producer-consumer problem, Producer is producing some items, whereas there is one Consumer that is consuming the items produced by the Producer. The same memory buffer is shared by both producers and consumers which is of fixed-size.

The task of the Producer is to produce the item, put it into the memory buffer, and again start producing items. Whereas the task of the Consumer is to consume the item from the memory buffer.

## DEADLOCK

***Deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.




**Deadlock can arise if** the **following four conditions hold simultaneously (Necessary Conditions)**
***Mutual Exclusion:*** One or more than one resource are non-shareable (Only one process can use at a time)
***Hold and Wait:*** A process is holding at least one resource and waiting for resources.
***No Preemption:*** A resource cannot be taken from a process unless the process releases the resource.
***Circular Wait:*** A set of processes are waiting for each other in circular form.

**Methods for handling deadlock**
There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into a deadlock state.
One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.
Avoidance is kind of futuristic in nature. By using strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources which process will need are known to us prior to execution of the process. We use Banker’s algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock.

2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem altogether: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.