**UNIT – 3**

**DEADLOCK**

***What is Deadlock?***

A **process in operating system uses resources** in the following way :-

1. The process **requests for some resource**.
2. **OS grant the resource** if it is available **otherwise** let the **process waits**.
3. The process **uses it and release it on completion**.

*A****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**.

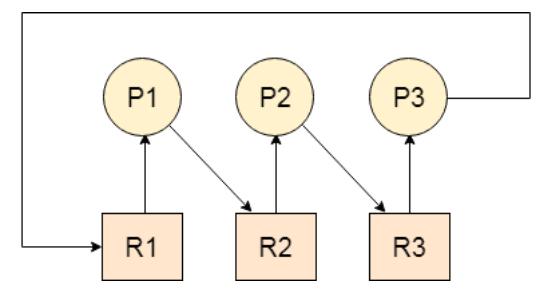
In this situation, **none of the process gets executed** since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Deadlock is an **infinite Process.** There is **only detection, resolution, prevention techniques.** But, there is **no Deadlock stopping techniques**.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R2 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they all are waiting. The computer becomes unresponsive since all the processes got blocked. This situation is called **Deadlock**.



***Deadlock Characterization or Necessary conditions for the occurrence of a Deadlock***

**Deadlock can arise if the** **following four conditions hold simultaneously (Necessary Conditions)**

### 1.Mutual Exclusion

This condition requires that at least one resource be held in a non-shareable mode, which means that **only one process can use the resource at any given time**. Both Resource 1 and Resource 2 are non-shareable in our scenario, and **only one process can have exclusive access to each resource at any given time**.As an example:

* Process A acquires Resource 1.
* Process B acquires Resource 2.

### 2.Hold and Wait

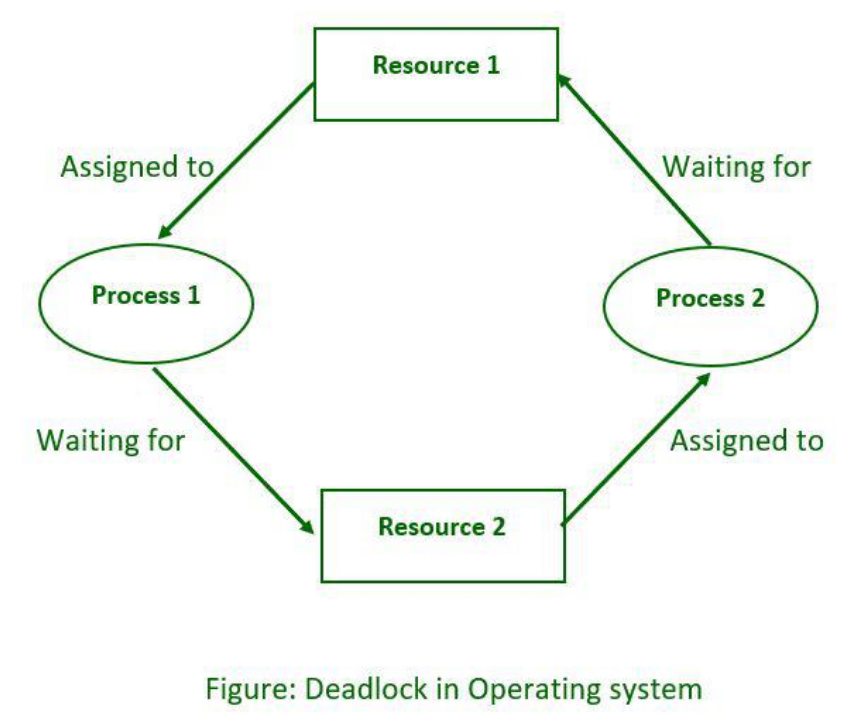
The hold and wait condition specifies that **a process must be holding at least one resource while waiting for other processes to release resources that are currently held by other processes**. In our example,

* Process A has Resource 1 and is awaiting Resource 2.
* Process B currently has Resource 2 and is awaiting Resource 1.
* Both processes hold one resource while waiting for the other, satisfying the hold and wait condition.

### 3.No Preemption

**Preemption** is the **act of taking a resource from a process before it has finished its task**. According to the **no preemption condition**, **resources cannot be taken forcibly from a process, a process can only release resources voluntarily after completing its task**. In our scenario, neither Process A nor Process B can be forced to release the resources in their possession.

### 4.Circular Wait

**Each process waiting for a resource held by the next process in a circular chain**. In our scenario, Process A is waiting for Resource 2, which is being held by Process B. Process B is awaiting Resource 1 from Process A. **This circular chain of dependencies causes a deadlock** because neither process can proceed, resulting in a system shutdown.

***Methods of handling deadlocks***

There are four approaches to dealing with deadlocks.

**1.** Deadlock Prevention

**2.** Deadlock avoidance (Banker's Algorithm)

**3.** Deadlock detection & recovery

**4.** Deadlock Ignorance (Ostrich Method)

**1.Deadlock Prevention:** The strategy of deadlock prevention is to **design the system in such a way that the possibility of deadlock is excluded**. Deadlock happens only **when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously**. If it is possible to **violate one of the four conditions at any time then the deadlock can never occur in the system**.

* **Eliminate Mutual Exclusion:**It is **not possible to dis-satisfy the**[**mutual exclusion**](https://www.geeksforgeeks.org/mutual-exclusion-in-synchronization/) **because** **some resources**, such as the tape drive and printer, are **inherently non-shareable**.
* **Eliminate Hold and wait:** **Allocate all required resources to the process before the start of its execution**, this way hold and wait condition is eliminated but it will lead to **low device utilization**. For example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to **starvation**.
* **Eliminate No Preemption:** Preempt resources from the process when resources are required by other high-priority processes.
* **Eliminate Circular Wait:** Each **resource** will be **assigned a numerical number**. A process can request the resources to increase/decrease the order of numbering. For example, if the process P1 is allocated resource R5, now next time if P1 asks for resource R3, R4 **(**lesser than R5**)** then such a request will not be granted, only a request for resources more than R5 will be granted.

**2.Deadlock Avoidance:** The deadlock avoidance Algorithm **works** **by proactively looking for potential deadlock situations before they occur**. It does this by **tracking the resource usage of each process and identifying conflicts that could potentially lead to a deadlock**. In deadlock avoidance, the operating system **checks** whether the system is in **safe state** or in **unsafe state** at **every step** which the operating system performs. The **process** **continues until the system is in safe state**. Once the **system** **moves to unsafe state** and if a **potential deadlock is identified**, the **algorithm** will **take steps to resolve the conflict**, such as **rolling back one of the processes or pre-emptively allocating resources to other processes**. The Deadlock Avoidance Algorithm is designed to **minimize the chances of a deadlock occurring**, although it **cannot guarantee that a deadlock will never occur**. It **requires knowledge of future process requests**. **Two techniques to avoid deadlock** :

1. Process initiation denial
2. Resource allocation denial

**Advantages of deadlock avoidance techniques:**

* Not necessary to pre-empt and rollback processes
* Less restrictive than deadlock prevention

**Disadvantages:**

* Future resource requirements must be known in advance
* Processes can be blocked for long periods
* Exists a fixed number of resources for allocation

**Banker’s Algorithm**

The Banker’s Algorithm is **based on the concept of resource allocation graphs**. A resource allocation graph is **a directed graph where each node represents a process, and each edge represents a resource**. The **state of the system** is **represented by the current allocation of resources between processes**. For example, if the system has three processes, each of which is using two resources, the resource allocation graph would look like this:

Processes A, B, and C would be the nodes, and the resources they are using would be the edges connecting them. The Banker’s Algorithm works by analyzing the state of the system and determining if it is in a safe state or at risk of entering a deadlock.

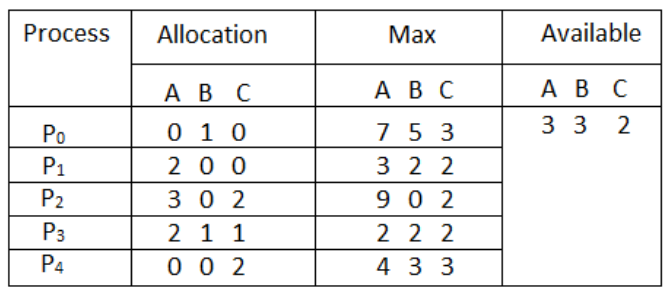
To determine if a system is in a safe state, the Banker’s Algorithm uses two matrices: the available matrix and the need matrix. The available matrix contains the amount of each resource currently available. The need matrix contains the amount of each resource required by each process.

The Banker’s Algorithm then checks to see if a process can be completed without overloading the system. It does this by subtracting the amount of each resource used by the process from the available matrix and adding it to the need matrix. If the result is in a safe state, the process is allowed to proceed, otherwise, it is blocked until more resources become available.

The Banker’s Algorithm is an effective way to prevent deadlocks in multiprogramming systems. It is used in many operating systems, including Windows and Linux. In addition, it is used in many other types of systems, such as manufacturing systems and banking systems.

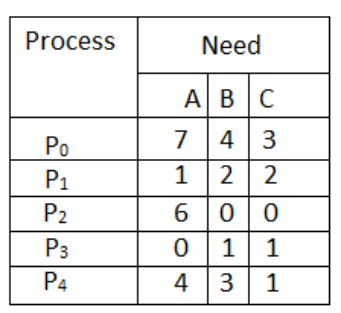
The Banker’s Algorithm is a powerful tool for resource allocation problems, but it is not foolproof. It can be fooled by processes that consume more resources than they need, or by processes that produce more resources than they need. Also, it can be fooled by processes that consume resources in an unpredictable manner. To prevent these types of problems, it is important to carefully monitor the system to ensure that it is in a safe state.

**Considering a system with five processes P0 to P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:**

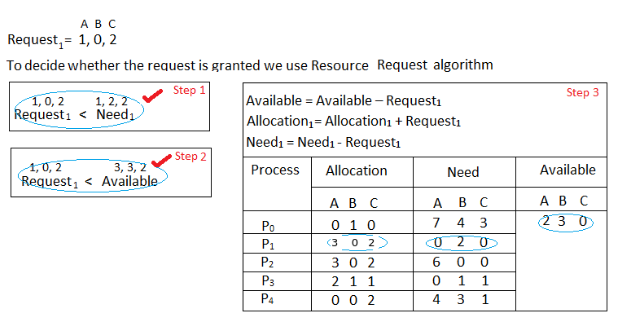


### ****Q.1 What will be the content of the Need matrix?****

### Need [i, j] = Max [i, j] – Allocation [i, j] So, the content of Need Matrix is:



### ****Q.2 What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?****



**3.Deadlock Detection & Recovery:** This approach **let the processes fall in deadlock** and then **periodically check whether deadlock occur in the system or not**. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock. Deadlock detection is used by employing an algorithm that tracks the circular waiting and kills one or more processes so that the deadlock is removed. The system state is examined periodically to determine if a set of processes is deadlocked. A deadlock is resolved by aborting and restarting a process, relinquishing all the resources that the process held.

* This technique does not limit resource access or restrict process action.
* Requested resources are granted to processes whenever possible.
* It never delays the process initiation and facilitates online handling.
* The disadvantage is the inherent pre-emption losses.

**4.** **Deadlock Ignorance:** In the Deadlock ignorance method the OS acts like the deadlock never occurs and completely ignores it even if the deadlock occurs. This method only applies if the deadlock occurs very rarely. The algorithm is very simple. It says ” if the deadlock occurs, simply reboot the system and act like the deadlock never occurred.” That’s why the algorithm is called the **Ostrich Algorithm**. Windows and Linux are mainly using this approach.

**Advantages:**

* Ostrich Algorithm is relatively easy to implement and is effective in most cases.
* It helps in avoiding the deadlock situation by ignoring the presence of deadlocks.

**Disadvantages:**

* Ostrich Algorithm does not provide any information about the deadlock situation.
* It can lead to reduced performance of the system as the system may be blocked for a long time.
* It can lead to a resource leak, as resources are not released when the system is blocked due to deadlock.

## 

**Difference between Deadlock and Starvation**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Deadlock** | **Starvation** |
| 1. | **All processes keep waiting for each other to complete** and **none get executed** | **High priority processes keep executing** and **low priority processes are blocked** |
| 2. | **Resources** are **blocked by the processes** | **Resources** are **continuously utilized by high priority processes** |
| 3. | Necessary conditions **Mutual Exclusion, Hold and Wait, No preemption, Circular Wait** | **Priorities** are **assigned to the processes** |
| 4. | Also known as **Circular wait** | Also known as **lived lock** |
| 5. | It can be **prevented by avoiding the necessary conditions for deadlock** | It can be **prevented by Aging** |

**Difference between Deadlock Prevention and Deadlock Aviodance**

|  |  |  |
| --- | --- | --- |
| **Features** | **Deadlock Prevention** | **Deadlock Avoidance** |
| **Definition** | It **assures** that **at least one of the four deadlock conditions never occurs**. | It **prevents the system from coming to** an **unsafe state**. |
| **Procedure** | It **prevents deadlock by limiting the resource request process** and **resource handling**. | It **automatically evaluates requests and determines whether they are safe for the system**. |
| **Resource Request** | **All** the **resources** in deadlock prevention are **requested together**. | **Resource requests** in deadlock avoidance are **executed according to the safe path**. |
| **Resource Allocation Strategy** | It is **conservative** in deadlock prevention. | It is **not conservative** in deadlock avoidance. |
| **Information** | It **doesn't need information** about **existing resources, resource requests, and available resources**. | It **needs information** about **existing resources, resource requests, and available resources**. |
| **Pre-emption** | In deadlock prevention, it **occurs more frequently**. | There is **no pre-emption** in deadlock avoidance. |
| **Future Resource Requests** | The knowledge of future process resource requirements is **not required** for deadlock prevention. | The knowledge of future process resource requirements is **necessary** for deadlock avoidance. |
| **Algorithms** | Some algorithms are used in deadlock prevention, including **Non-blocking synchronization and serializing tokens**. | The most widely used algorithm in deadlock avoidance is **Banker's algorithm**. |

**PROCESS SYNCHRONIZATION**

# ***Introduction of Process Synchronization***

Process Synchronization is the **coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner**. It **aims** to **resolve the problem of race conditions and other synchronization issues in a concurrent system**.

The **main objective** of process synchronization is to **ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access**. To achieve this, various synchronization techniques like **semaphores**, **monitors**, and **critical sections** are used.

In a **multi-process system**, synchronization is necessary to **ensure data consistency and integrity, and to avoid the risk of deadlocks and other synchronization problems**.

On the basis of synchronization, processes are categorized in following two types:

* **Independent Process**: The execution of one process **does not affect** the execution of other processes.
* **Cooperative Process**: A process that can affect or be **affected** by other processes executing in the system.

## ***Race Condition***

When **more than one process is executing the same code** or **accessing the same memory or any shared variable** in that condition there is **a possibility that the output or the value of the shared variable** is **wrong** so for that all the processes doing the race to say that my output is correct this condition known as a **race condition.** A race condition is **a situation that may occur inside a critical section**. Race conditions in critical sections can be **avoided** if the critical section is treated as an atomic instruction.

## ***Advantages of Process Synchronization***

* Ensures data consistency and integrity
* Avoids race conditions
* Prevents inconsistent data due to concurrent access
* Supports efficient and effective use of shared resources

## ***Disadvantages of Process Synchronization***

* Adds overhead to the system
* This can lead to performance degradation
* Increases the complexity of the system
* Can cause deadlocks if not implemented properly.

# ***Hardware Synchronization Algorithms : Unlock and Lock, Test and Set, Swap***

Process Synchronization problems occur when two processes running concurrently share the same data or same variable. The value of that variable may not be updated correctly before its being used by a second process. Such a condition is known as **Race Around Condition.**

There are three algorithms in the hardware approach of solving Process Synchronization problem:

1. Test and Set
2. Swap
3. Unlock and Lock

## **1.Test and Set:** This algorithm uses a shared variable called "lock" which is initially set to false. When a process invokes the Test And Set function, it returns the current value of "lock" and sets it to true. The first process to access the critical section can do so immediately, as the Test And Set function returns false initially. Subsequent processes are blocked by the while loop until the lock is released. While mutual exclusion and progress are ensured, bounded waiting is not guaranteed as any new process that finds the lock released can enter immediately.

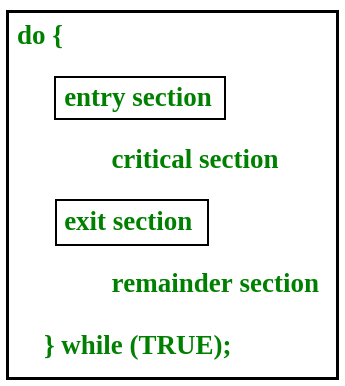
## **2.Swap:** Similar to Test and Set, the Swap algorithm also utilizes a shared variable to control access to the critical section. However, it employs a key variable which is swapped with the lock variable. The first process executes smoothly, but subsequent processes are blocked until the lock is released by the first process. While this ensures mutual exclusion and progress, bounded waiting is not guaranteed due to the lack of a queue system.

## **3.Unlock and Lock:** This algorithm enhances Test and Set by introducing a waiting queue for processes attempting to access the critical section. Each process is assigned a waiting flag. When a process exits the critical section, it checks the waiting queue. If there are no waiting processes, it releases the lock. If there are waiting processes, it selects the next process in line and grants it access to the critical section. This ensures bounded waiting, as processes are served in a fair and orderly manner.

## In summary, while all three algorithms ensure mutual exclusion and progress, only the Unlock and Lock algorithm guarantees bounded waiting by implementing a waiting queue system.

***Critical Section Problem***

A critical section is a **code segment** that can be **accessed** by **only one process at a time**. **The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables**. So the critical section problem means designing a way for cooperative processes to access shared resources without creating data inconsistencies.



In the entry section, the process requests for entry in the **Critical Section.**

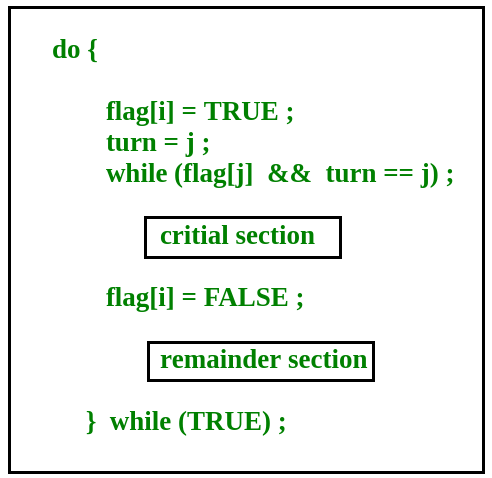
**Any** **solution** to the **critical section problem** must satisfy **three** requirements:

* **Mutual Exclusion**: If a **process is executing in its critical section**, then no other process is allowed to execute in the critical section.
* **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter the critical section next, and the selection cannot be postponed indefinitely.In simple words, 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.
* **Bounded Waiting**: A **bound** **must exist** on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

***Peterson’s Solution***

Peterson’s Solution is a classical software-based solution to the critical section problem. In Peterson’s solution, we have **two shared variables**:

1. **boolean flag[i]:** Initialized to FALSE, initially no one is interested in entering the critical section
2. **int turn:** The process whose turn is to enter the critical section.



**Peterson’s Solution preserves all three conditions:**

* Mutual Exclusion is assured as only one process can access the critical section at any time.
* Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
* Bounded Waiting is preserved as every process gets a fair chance.

## ***Disadvantages of Peterson’s Solution***

* It involves busy waiting. (In the Peterson’s solution, the code statement- “while(flag[j] && turn == j);” is responsible for this. Busy waiting is not favored because it wastes CPU cycles that could be used to perform other tasks.)
* It is limited to 2 processes.
* Peterson’s solution cannot be used in modern CPU architectures.

# ***Semaphores***

Semaphores are **integer variables** that are **used to solve the critical section problem by using two atomic operations**, **wait** and **signal** that are used for process synchronization. The Semaphore **cannot be negative**. The **least value** for a Semaphore is **zero (0).** The **Maximum value** of a Semaphore **can be anything**. The Semaphores usually have two operations:-

* Wait( )
* Signal( )

**Wait**

The wait operation **decrements** the value of its argument S, **if it is positive**. If S is negative or zero, then no operation is performed.

|  |
| --- |
| wait(S)  {     while (S<=0);     S--;  } |

**Signal**

The signal operation **increments** the value of its argument S.

|  |
| --- |
| signal(S)  {     S++;  } |

***Types of Semaphores***

There are two main types of semaphores :-

**1.Counting Semaphores**

These are **integer value semaphores** and **have an unrestricted value domain**. These semaphores are **used to coordinate the resource access**, where the **semaphore count is the number of available resources**. If the **resources are added**, **semaphore count automatically incremented** and if the **resources are removed**, the **count** is **decremented**.

**2.Binary Semaphores**

The binary semaphores are **like counting semaphores but their value is restricted to 0 and 1**. The **wait** operation **only works when the semaphore is 1** and the **signal** **operation succeeds when semaphore is 0**. It is sometimes **easier** to implement binary semaphores than counting semaphores.

***Advantages of Semaphores:***

1. A simple and effective mechanism for process synchronization.
2. Supports coordination between multiple processes.
3. Provides a flexible and robust way to manage shared resources.
4. It can be used to implement critical sections in a program.
5. It can be used to avoid race conditions.

***Disadvantages of Semaphores:***

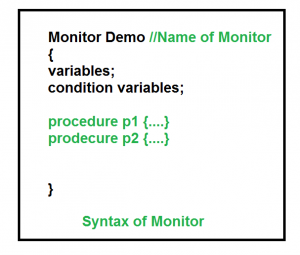
1. It can lead to performance degradation due to overhead associated with wait and signal operations.
2. Can result in deadlock if used incorrectly.
3. It can cause performance issues in a program if not used properly.
4. It can be difficult to debug and maintain.
5. It can be prone to race conditions and other synchronization problems if not used correctly.
6. It can be vulnerable to certain types of attacks, such as denial of service attacks.

***Monitors***

Monitors are a synchronization tool used in process synchronization to manage access to shared resources and coordinate the actions of numerous threads or processes. When opposed to low-level primitives like locks or semaphores, they offer a higher-level abstraction for managing concurrency.

Monitors are used to prevent concurrent access to shared resources by numerous threads or processes. When several entities attempt to alter the same resource at once, they can avoid conflicts and data inconsistencies. Monitors give synchronization a formal method, making the design and implementation of concurrent systems simpler.

**Syntax:**

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2015/06/monitors.png)

**Condition Variables:** Two different operations are performed on the condition variables of the monitor.

Wait.

signal.

***Advantages***

1. Mutual exclusion is automatic in monitors.
2. Monitors are less difficult to implement than semaphores as they are higher-level constructs.
3. Monitors may overcome the timing errors that occur when semaphores are used.
4. Monitors are a collection of procedures and condition variables that are combined in a special type of module.

***Disadvantages***

1. Monitors must be implemented into the programming language.
2. The compiler should generate code for them.
3. It gives the compiler the additional burden of knowing what operating system features is available for controlling access to crucial sections in concurrent processes.

***Difference between Semaphore and Monitor***

|  |  |  |
| --- | --- | --- |
| **Features** | **Semaphore** | **Monitor** |
| **Definition** | A semaphore is an integer variable that allows many processes in a parallel system to manage access to a common resource like a multitasking OS. | It is a synchronization process that enables threads to have mutual exclusion and the wait() for a given condition to become true. |
| **Syntax** | // Wait Operation wait(Semaphore S) { while (S<=0); S--; } // Signal Operation signal(Semaphore S) { S++; } | monitor { variables;  condition variables; Procedure P1() { ... } Procedure P2() { ... } **.**  **.**  **.** Procedure Pn() { ... } } |
| **Basic** | Integer variable | Abstract data type |
| **Access** | When a process uses shared resources, it calls the wait() method on S, and when it releases them, it uses the signal() method on S. | When a process uses shared resources in the monitor, it has to access them via procedures. |
| **Action** | The semaphore's value shows the number of shared resources available in the system. | The Monitor type includes shared variables as well as a set of procedures that operate on them. |
| **Condition Variable** | No condition variables. | It has condition variables. |

**CPU Scheduling**

CPU Scheduling is a process that allows one process to use the CPU while another process is delayed due to unavailability of any resources such as I / O etc, thus making full use of the CPU. In short, CPU scheduling decides the order and priority of the processes to run and allocates the CPU time based on various parameters such as CPU usage, throughput, turnaround, waiting time, and response time. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

## **Criteria of CPU Scheduling**

CPU Scheduling has several criteria. Some of them are mentioned below.

### ****1. CPU utilization****

The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a [real-time system](https://www.geeksforgeeks.org/real-time-systems/), it varies from 40 to 90 percent depending on the load upon the system.

### ****2. Throughput****

A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.

### ****3. Turnaround Time****

For a particular process, an important criterion is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.

*Turn Around Time = Completion Time – Arrival Time.*

### ****4. Waiting Time****

A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

*Waiting Time = Turnaround Time – Burst Time.*

### ****5. Response Time****

In an interactive system, turn-around time is not the best criterion. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criterion is the time taken from submission of the process of the request until the first response is produced. This measure is called response time.

*Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time***6. Completion Time**

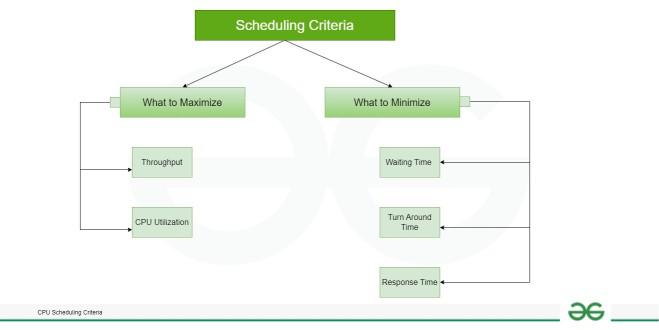
The completion time is the time when the process stops executing,  which means that the process has completed its burst time and is completely executed.

### ****7. Priority****

If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.

### ****8. Predictability****

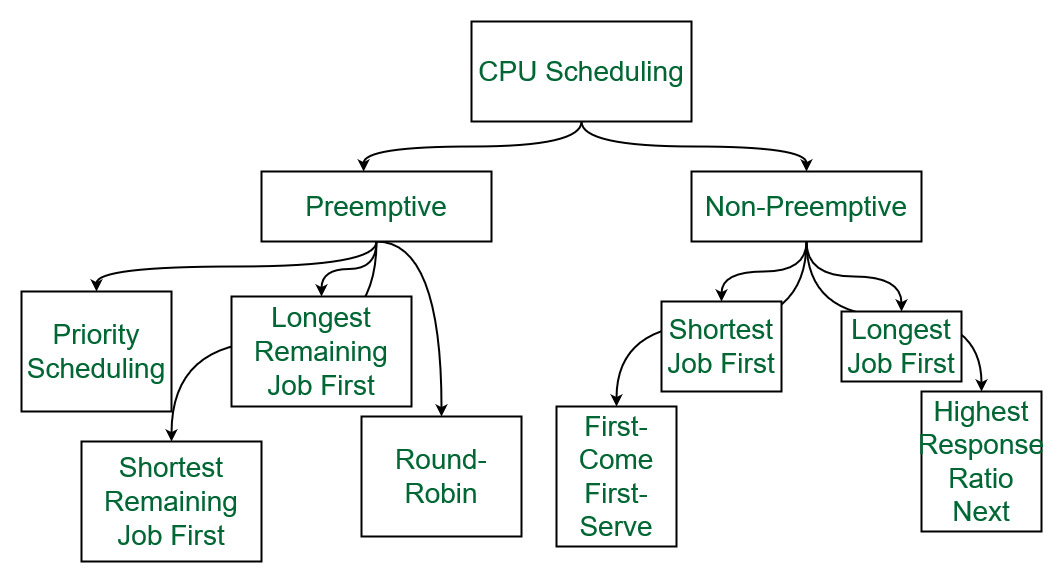
A given process always should run in about the same amount of time under a similar system load.



## **What are the different types of CPU Scheduling Algorithms?**

There are mainly two types of scheduling methods:

* **Preemptive Scheduling :** Preemptive scheduling is a method used by operating systems to manage the execution of multiple processes where the operating system can interrupt a currently executing process to allocate the CPU to another process, typically based on priorities, time slices, or events, without requiring the cooperation of the running process.
* **Non-Preemptive Scheduling :** Non-preemptive scheduling, also known as cooperative scheduling, is a method used by operating systems to manage the execution of multiple processes where a currently running process continues to hold the CPU until it voluntarily relinquishes control, typically by terminating or entering a waiting state, without being interrupted by the operating system.



*Different types of CPU Scheduling Algorithms*

### ****1. First Come First Serve:****

**FCFS**considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using [FIFO queue](https://www.geeksforgeeks.org/queue-data-structure/).

**Characteristics of FCFS:**

* FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
* Tasks are always executed on a First-come, First-serve concept.
* FCFS is easy to implement and use.
* This algorithm is not much efficient in performance, and the wait time is quite high.

**Advantages of FCFS:**

* Easy to implement
* First come, first serve method

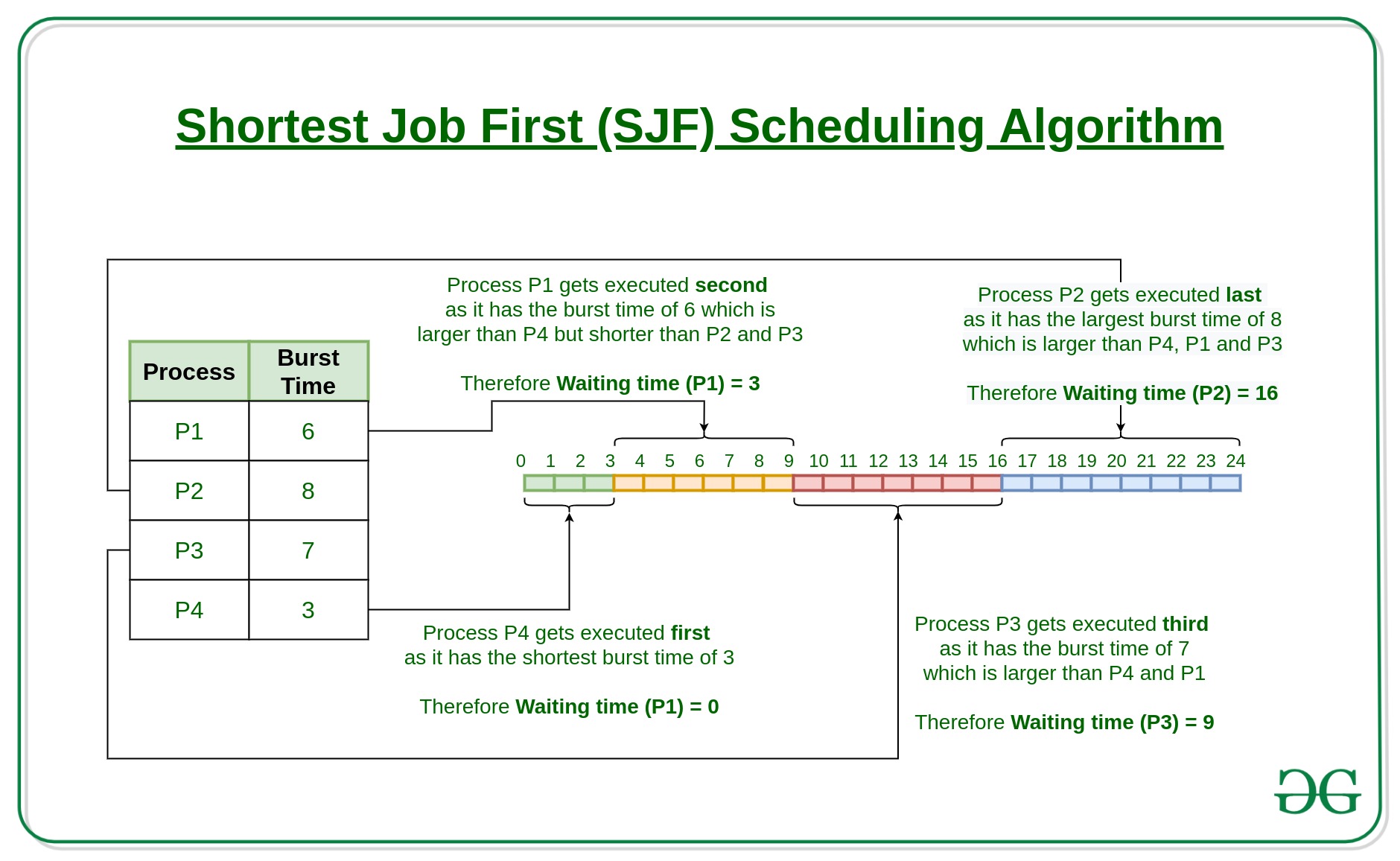
**Disadvantages of FCFS:**

* FCFS suffers from **Convoy effect**.
* The average waiting time is much higher than the other algorithms.
* FCFS is very simple and easy to implement and hence not much efficient.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on [First come, First serve Scheduling.](https://www.geeksforgeeks.org/first-come-first-serve-cpu-scheduling-non-preemptive/)

### 2. Shortest Job First(SJF):

**Shortest job first (SJF)** is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed. The full form of SJF is Shortest Job First.



**Characteristics of SJF:**

* Shortest Job first has the advantage of having a minimum average waiting time among all [operating system scheduling algorithms.](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)
* It is associated with each task as a unit of time to complete.
* It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.

**Advantages of Shortest Job first:**

* As SJF reduces the average waiting time thus, it is better than the first come first serve scheduling algorithm.
* SJF is generally used for long term scheduling

**Disadvantages of SJF:**

* One of the demerit SJF has is starvation.
* Many times it becomes complicated to predict the length of the upcoming CPU request

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on [Shortest Job First.](https://www.geeksforgeeks.org/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/)

### 3. Longest Job First(LJF):

**Longest Job First(LJF)** scheduling process is just opposite of shortest job first (SJF), as the name suggests this algorithm is based upon the fact that the process with the largest burst time is processed first. Longest Job First is non-preemptive in nature.

**Characteristics of LJF:**

* Among all the processes waiting in a waiting queue, CPU is always assigned to the process having largest burst time.
* If two processes have the same burst time then the tie is broken using [FCFS](https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/)i.e. the process that arrived first is processed first.
* LJF CPU Scheduling can be of both preemptive and non-preemptive types.

**Advantages of LJF:**

* No other task can schedule until the longest job or process executes completely.
* All the jobs or processes finish at the same time approximately.

**Disadvantages of LJF:**

* Generally, the LJF algorithm gives a very high [average waiting time](https://www.geeksforgeeks.org/difference-between-turn-around-time-tat-and-waiting-time-wt-in-cpu-scheduling/) and[average turn-around time](https://www.geeksforgeeks.org/difference-between-turn-around-time-tat-and-waiting-time-wt-in-cpu-scheduling/) for a given set of processes.
* This may lead to convoy effect.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on the [Longest job first scheduling](https://www.geeksforgeeks.org/longest-job-first-ljf-cpu-scheduling-algorithm/).

### 4. Priority Scheduling:

**Preemptive Priority CPU Scheduling Algorithm** is a pre-emptive method of [CPU scheduling algorithm](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/) that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there is more than one process with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

**Characteristics of Priority Scheduling:**

* Schedules tasks based on priority.
* When the higher priority work arrives and a task with less priority is executing, the higher priority proess will takes the place of the less priority proess and
* The later is suspended until the execution is complete.
* Lower is the number assigned, higher is the priority level of a process.

**Advantages of Priority Scheduling:**

* The average waiting time is less than FCFS
* Less complex

**Disadvantages of Priority Scheduling:**

* One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the [Starvation Problem](https://www.geeksforgeeks.org/starvation-and-aging-in-operating-systems/). This is the problem in which a process has to wait for a longer amount of time to get scheduled into the CPU. This condition is called the starvation problem.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on [Priority Preemptive Scheduling algorithm](https://www.geeksforgeeks.org/preemptive-priority-cpu-scheduling-algortithm/).

### 5. Round robin:

**Round Robin** is a [CPU scheduling algorithm](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/) where each process is cyclically assigned a fixed time slot. It is the [preemptive](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/)version of[First come First Serve CPU Scheduling algorithm](https://www.geeksforgeeks.org/first-come-first-serve-cpu-scheduling-non-preemptive/). Round Robin CPU Algorithm generally focuses on Time Sharing technique.

**Characteristics of Round robin:**

* It’s simple, easy to use, and starvation-free as all processes get the balanced CPU allocation.
* One of the most widely used methods in CPU scheduling as a core.
* It is considered preemptive as the processes are given to the CPU for a very limited time.

**Advantages of Round robin:**

* Round robin seems to be fair as every process gets an equal share of CPU.
* The newly created process is added to the end of the ready queue.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on the [Round robin Scheduling algorithm](https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/).

### 6. Shortest Remaining Time First:

**Shortest remaining time first** is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated to the job closest to completion. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

**Characteristics of** **Shortest remaining time first:**

* SRTF algorithm makes the processing of the jobs faster than SJF algorithm, given it’s overhead charges are not counted.
* The context switch is done a lot more times in SRTF than in SJF and consumes the CPU’s valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.

**Advantages of SRTF:**

* In SRTF the short processes are handled very fast.
* The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.

**Disadvantages of SRTF:**

* Like the shortest job first, it also has the potential for process starvation.
* Long processes may be held off indefinitely if short processes are continually added.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on the [shortest remaining time first](https://www.geeksforgeeks.org/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/).

### 7. Longest Remaining Time First:

**The longest remaining time first** is a preemptive version of the longest job first scheduling algorithm. This scheduling algorithm is used by the operating system to program incoming processes for use in a systematic way. This algorithm schedules those processes first which have the longest processing time remaining for completion.

**Characteristics of longest remaining time first:**

* Among all the processes waiting in a waiting queue, the CPU is always assigned to the process having the largest burst time.
* If two processes have the same burst time then the tie is broken using [FCFS](https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/)i.e. the process that arrived first is processed first.
* LRTF CPU Scheduling can be of both preemptive and non-preemptive.
* No other process can execute until the longest task executes completely.
* All the jobs or processes finish at the same time approximately.

**Disadvantages of LRTF:**

* This algorithm gives a very high [average waiting time](https://www.geeksforgeeks.org/difference-between-turn-around-time-tat-and-waiting-time-wt-in-cpu-scheduling/) and[average turn-around time](https://www.geeksforgeeks.org/difference-between-turn-around-time-tat-and-waiting-time-wt-in-cpu-scheduling/) for a given set of processes.
* This may lead to a convoy effect.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on the [longest remaining time first](https://www.geeksforgeeks.org/longest-remaining-time-first-lrtf-cpu-scheduling-algorithm/).

### 8. Highest Response Ratio Next:

**Highest Response Ratio Next**is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms. The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once selected will run till completion.

**Characteristics of Highest Response Ratio Next:**

* The **criteria** for HRRN is**Response Ratio,**and the **mode** is **Non-Preemptive.**
* HRRN is considered as the modification of [Shortest Job First](https://www.geeksforgeeks.org/shortest-job-first-cpu-scheduling-with-predicted-burst-time/) to reduce the problem of [starvation](https://www.geeksforgeeks.org/starvation-aging-operating-systems/).
* In comparison with SJF, during the HRRN scheduling algorithm, the CPU is allotted to the next process which has the **highest response ratio** and not to the process having less burst time.

***Response Ratio = (W + S)/S***

*Here,****W****is the waiting time of the process so far and****S****is the Burst time of the process.*

**Advantages of HRRN:**

* HRRN Scheduling algorithm generally gives better performance than the[shortest job first](https://www.geeksforgeeks.org/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/) Scheduling.
* There is a reduction in waiting time for longer jobs and also it encourages shorter jobs.

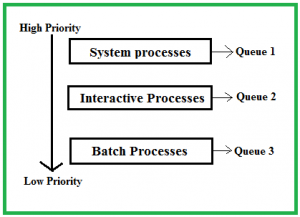
**Disadvantages of HRRN:**

* The implementation of HRRN scheduling is not possible as it is not possible to know the burst time of every job in advance.
* In this scheduling, there may occur an overload on the CPU.

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on [Highest Response Ratio Next](https://www.geeksforgeeks.org/highest-response-ratio-next-hrrn-cpu-scheduling/).

### 9. Multiple Queue Scheduling:

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 a **background (batch)** process. These two classes have different scheduling needs. For this kind of situation **Multilevel Queue Scheduling** is used.



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

* **System Processes:**The CPU itself has its process to run, generally termed as System Process.
* **Interactive Processes:**An Interactive Process is a type of process in which there should be the same type of interaction.
* **Batch Processes:**Batch processing is generally a technique in the Operating system that collects the programs and data together in the form of a **batch** before the **processing** starts.

**Advantages of multilevel queue scheduling:**

* The main merit of the multilevel queue is that it has a low scheduling overhead.

**Disadvantages of multilevel queue scheduling:**

* Starvation problem
* It is inflexible in nature

To learn about how to implement this CPU scheduling algorithm, please refer to our detailed article on [Multilevel Queue Scheduling](https://www.geeksforgeeks.org/multilevel-queue-mlq-cpu-scheduling/).

### 10. ****Multilevel Feedback Queue Scheduling****:

**Multilevel Feedback Queue Scheduling (MLFQ)** CPU Scheduling is like  **Multilevel Queue Scheduling**but in this process can move between the queues. And thus, much more efficient than multilevel queue scheduling.

**Characteristics of Multilevel Feedback Queue Scheduling:**

* In a[multilevel queue-scheduling](https://www.geeksforgeeks.org/multilevel-queue-mlq-cpu-scheduling/) algorithm, processes are permanently assigned to a queue on entry to the system, and processes are not allowed to move between queues.
* As the processes are permanently assigned to the queue, this setup has the advantage of low scheduling overhead,
* But on the other hand disadvantage of being inflexible.

**Advantages of Multilevel feedback queue scheduling:**

* It is more flexible
* It allows different processes to move between different queues

**Disadvantages of Multilevel feedback queue scheduling:**

* It also produces CPU overheads
* It is the most complex algorithm.

**Real Time Scheduling**

Real-time[systems](https://www.geeksforgeeks.org/real-time-systems/) are systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In particular, these tasks are related to control of certain events (or) reacting to them. Real-time tasks can be classified as hard real-time tasks and soft real-time tasks.

A hard real-time task must be performed at a specified time which could otherwise lead to huge losses. In soft real-time tasks, a specified deadline can be missed. This is because the task can be rescheduled (or) can be completed after the specified time.

In real-time systems, the scheduler is considered as the most important component which is typically a short-term task scheduler. The main focus of this scheduler is to reduce the response time associated with each of the associated processes instead of handling the deadline.

The scheduling algorithm are classified as follows :-

1. **Static table-driven approaches:**   
   These algorithms usually perform a static analysis associated with scheduling and capture the schedules that are advantageous. This helps in providing a schedule that can point out a task with which the execution must be started at run time.
2. **Static priority-driven preemptive approaches:**   
   Similar to the first approach, these type of algorithms also uses static analysis of scheduling. The difference is that instead of selecting a particular schedule, it provides a useful way of assigning priorities among various tasks in preemptive scheduling.
3. **Dynamic planning-based approaches:**   
   Here, the feasible schedules are identified dynamically (at run time). It carries a certain fixed time interval and a process is executed if and only if satisfies the time constraint.
4. **Dynamic best effort approaches:**   
   These types of approaches consider deadlines instead of feasible schedules. Therefore the task is aborted if its deadline is reached. This approach is used widely is most of the real-time systems.

**Advantages of Scheduling in Real-Time Systems :**

* **Meeting Timing Constraints:** Scheduling ensures that real-time tasks are executed within their specified timing constraints. It guarantees that critical tasks are completed on time, preventing potential system failures or losses.
* **Resource Optimization:**Scheduling algorithms allocate system resources effectively, ensuring efficient utilization of processor time, memory, and other resources. This helps maximize system throughput and performance.
* **Priority-Based Execution:**Scheduling allows for priority-based execution, where higher-priority tasks are given precedence over lower-priority tasks. This ensures that time-critical tasks are promptly executed, leading to improved system responsiveness and reliability.
* **Predictability and Determinism:**Real-time scheduling provides predictability and determinism in task execution. It enables developers to analyze and guarantee the worst-case execution time and response time of tasks, ensuring that critical deadlines are met.
* **Control Over Task Execution:** Scheduling algorithms allow developers to have fine-grained control over how tasks are executed, such as specifying task priorities, deadlines, and inter-task dependencies. This control facilitates the design and implementation of complex real-time systems.

**Disadvantages of Scheduling in Real-Time Systems :**

* **Increased Complexity:**Real-time scheduling introduces additional complexity to system design and implementation. Developers need to carefully analyze task requirements, define priorities, and select suitable scheduling algorithms. This complexity can lead to increased development time and effort.
* **Overhead:** Scheduling introduces some overhead in terms of context switching, task prioritization, and scheduling decisions. This overhead can impact system performance, especially in cases where frequent context switches or complex scheduling algorithms are employed.
* **Limited Resources:** Real-time systems often operate under resource-constrained environments. Scheduling tasks within these limitations can be challenging, as the available resources may not be sufficient to meet all timing constraints or execute all tasks simultaneously.
* **Verification and Validation:** Validating the correctness of real-time schedules and ensuring that all tasks meet their deadlines require rigorous testing and verification techniques. Verifying timing constraints and guaranteeing the absence of timing errors can be a complex and time-consuming process.
* **Scalability:**Scheduling algorithms that work well for smaller systems may not scale effectively to larger, more complex real-time systems. As the number of tasks and system complexity increases, scheduling decisions become more challenging and may require more advanced algorithms or approaches.

**Synchronization Problems**

These problems are used for testing nearly every newly proposed synchronization scheme. The following problems of synchronization are considered as classical problems:

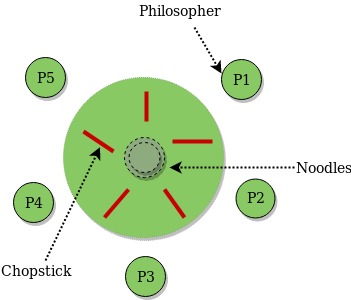
**1.** Bounded-buffer (or Producer-Consumer) Problem,  
**2.** Dining-Philosophers Problem,  
**3.** Readers and Writers Problem,  
**4.** Sleeping Barber Problem

### ****Bounded-Buffer (or Producer-Consumer) Problem****

The [Bounded Buffer problem](https://www.geeksforgeeks.org/producer-consumer-solution-using-semaphores-java/) is also called the producer-consumer problem. This problem is generalized in terms of the Producer-Consumer problem. The solution to this problem is, to create two counting semaphores “full” and “empty” to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

### ****Dining-Philosophers Problem****

The [Dining Philosopher Problem](https://www.geeksforgeeks.org/operating-system-dining-philosopher-problem-using-semaphores/) states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.



### ****Readers and Writers Problem****

Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Precisely in OS we call this situation as the [readers-writers problem](https://www.geeksforgeeks.org/readers-writers-problem-set-1-introduction-and-readers-preference-solution/). Problem parameters:

* One set of data is shared among a number of processes.
* Once a writer is ready, it performs its write. Only one writer may write at a time.
* If a process is writing, no other process can read it.
* If at least one reader is reading, no other process can write.
* Readers may not write and only read.

### ****Sleeping Barber Problem****

Barber shop with one barber, one barber chair and N chairs to wait in. When no customers the barber goes to sleep in barber chair and must be woken when a customer comes in. When barber is cutting hair new customers take empty seats to wait, or leave if no vacancy. This is basically the [Sleeping Barber Problem.](https://www.geeksforgeeks.org/operating-system-sleeping-barber-problem/)