# CPU Scheduling

* Scheduling of processes/work is done to finish the work on time.
* **CPU Scheduling** is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU.
* The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.
* Whenever the CPU becomes idle, the operating system must select one of the processes in the line ready for launch.
* The selection process is done by a temporary (CPU) scheduler. The Scheduler selects between memory processes ready to launch and assigns the CPU to one of them.

## **Process**:

A process is **the instance of a computer program that is being executed by one or many threads.** It contains the program code and its activity.

## **Process Scheduling**:

Process Scheduling is the process of the process manager handling the removal of an active process from the CPU and selecting another process based on a specific strategy. It is an integral part of Multi-programming applications, such operating systems allow more than one process to be loaded into usable memory at a time and the loaded shared CPU process uses repetition time.

**Types :**

Long term or Job Scheduler Short term or CPU Scheduler

Medium-term Scheduler

## 

**Scheduling:**

**It** is important in many different computer environments, One of the most important areas is scheduling which programs will work on the CPU. This task is handled by the OS of the computer and there are many different ways in which we can choose to configure programs.

**Process Scheduling**

**It** allows the OS to allocate CPU time for each process. Another important reason to use a process scheduling system is that it keeps the CPU busy at all times. This allows you to get less response time for programs.

Need for CPU scheduling algorithm?

**CPU scheduling** is the process of deciding which process will own the CPU to use while another process is suspended.

The main function is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.

In [Multiprogramming](https://www.geeksforgeeks.org/difference-between-multitasking-multithreading-and-multiprocessing/), if the long-term scheduler selects multiple I / O binding processes then most of the time, the CPU remains an idle.

Improve resource utilization.

If most operating systems change their status from performance to waiting then there may always be a chance of failure in the system.

In order to minimize this excess, the OS needs to schedule tasks in order to make full use of the CPU and avoid the possibility of deadlock.

### ****Objectives** of Process Scheduling Algorithm**:****

* Utilization of CPU at maximum level.
* **Keep CPU as busy as possible**.
* **Allocation of CPU should be fair**.
* There should be a **minimum waiting time** and the process should not starve in the ready queue.

**Throughput should be Maximum**. Number of processes that complete their execution per time unit should be maximized.

**Minimum turnaround time** Time taken by a process to finish execution should be the least.

**Minimum** **response time** It means that the time when a process produces the first response should be as less as possible.

## 

## Terminologies: **Arrival Time:**

Time at which the process arrives in the ready queue.

**Completion Time:**

Time at which process completes its execution.

**Burst Time:**

Time required by a process for CPU execution.

**Turn Around Time:**

Time Difference between completion time and arrival time.

*Turn Around Time = Completion Time  –  Arrival Time*

**Waiting Time(W.T):**

Time Difference between turn around time and burst time.

*Waiting Time = Turn Around Time  –  Burst Time*

Different **CPU Scheduling algorithms**have different structures and the choice of a particular algorithm depends on a variety of factors.

Factors

**CPU utilization:**

The main purpose of any CPU algorithm is to keep the CPU as busy as possible. CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.

**Throughput:**

The average CPU performance is the number of processes performed and completed during each unit,this is called throughput.

The output may vary depending on the length or duration of the processes.

**Turn round Time:**

For a particular process, the important conditions are how long it takes to perform that process.

The time elapsed from the time of process delivery to the time of completion is known as the **conversion time**.

Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

**Waiting Time:**

The Scheduling algorithm does not affect the time required to complete the process once it has started performing.

It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.

**Response Time:**

The process may produce something early and continue to computing the new results while the previous results are released to the user.

Another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

## 

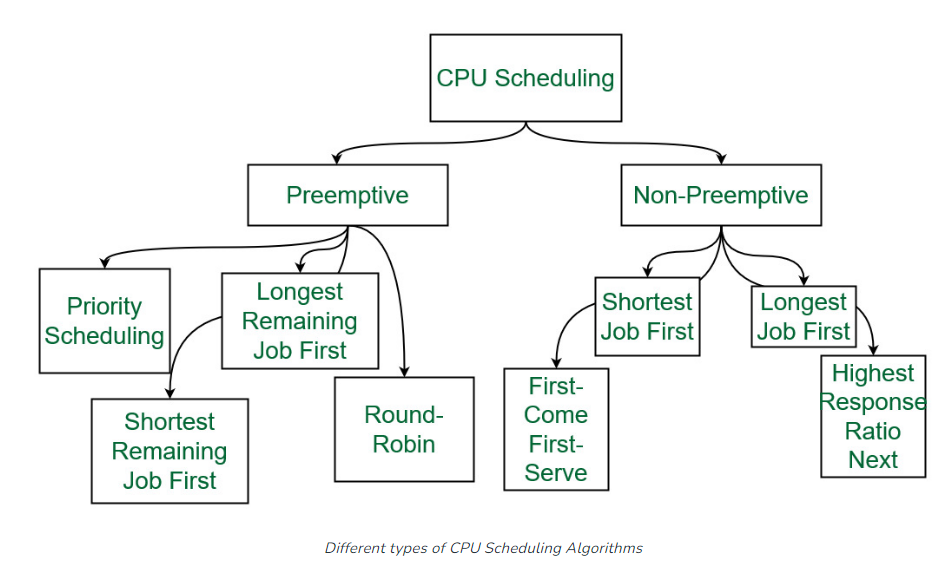
**Types** of scheduling methods:

[**Preemptive** Scheduling](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/)**:**

It is used when a process switches from running state to ready state or from the waiting state to the ready state.

[**Non-Preemptive** Scheduling](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/):

It is used when a process terminates , or when a process switches from running state to waiting state.

****

### ****1.First Come First Serve / First In First Out :****

**FCFS / FIFO** considered to be the simplest of all operating system scheduling algorithms.

It 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 :**

* FCFS supports both 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 :**

* Easy to implement
* First come, first serve method

**Disadvantages :**

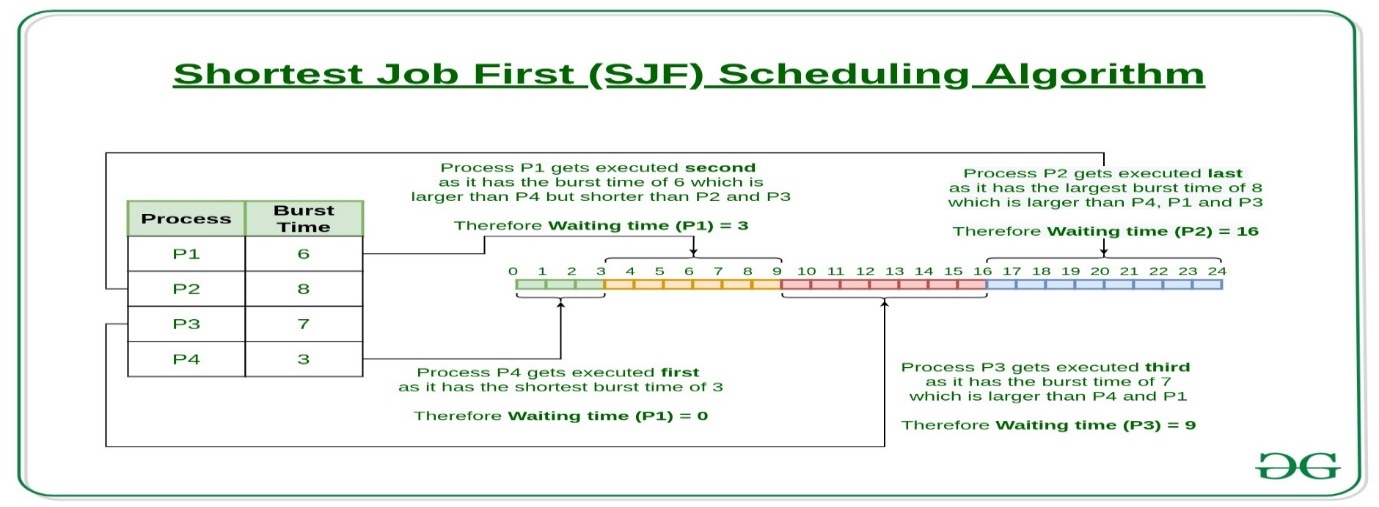
* FCFS suffers from **Convoy effect**.
* The average waiting time is much higher than the other algorithms.
* FCFS is very simple and easy to implement and hence not much efficient.

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

**It** 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.

It significantly reduces the average waiting time for other processes waiting to be executed.

**Characteristics :**

* It has the advantage of having a minimum average waiting time among all [OS 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 :**

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

**Disadvantages :**

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

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

* It 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.
* It is non-preemptive in nature.

**Characteristics:**

* 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.
* It can be of both preemptive and non-preemptive types.

**Advantages :**

* 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 :**

* Generally, it 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.
* It may lead to convoy effect.

### 4. Priority Scheduling:

* **It** is a pre-emptive method of CPU scheduling algoritham  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 are more than one processor with equal value), then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

**Characteristics :**

* Schedules tasks based on priority.
* When the higher priority work arrives while a task with less priority is executed, the higher priority work takes the place of the less priority one and the latter is suspended until the execution is complete.
* Lower is the number assigned, higher is the priority level of a process.

**Advantages :**

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

**Disadvantages :**

* [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.

.

### 5. Round robin(RR):

* **It** is an [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 FCFS.
* Round Robin CPU Algorithm generally focuses on Time Sharing technique.

**Characteristics:**

* 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 :**

* 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.

6. **Shortest Remaining Time First(SRTF):**

* **It** is the preemptive version of the SJF 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 :**

* It 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 :**

* 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:**

* 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.

### 7. Longest Remaining Time First(**LRTF)**:

* **It** 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 :**

* 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.
* LJF CPU Scheduling can be of both preemptive and non-preemptive types.

**Advantages:**

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

**Disadvantages :**

* 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.

Turn Around Time =

* waiting\_time + burst\_time
* Completion Time – Arrival Time

*Total waiting time  = Completion time – (Arrival time + Execution time)*

***Average Waiting Time = (sum of all waiting time)/(Number of processes)***

Completion Time = Start Time + Burst Time

Waiting Time = Turn Around Time – Burst Time

**average waiting time** = total\_waiting\_time / no\_of\_processes.

**average turnaround time** = total\_turn\_around\_time / no\_of\_processes.

**Service Time:**

Service time means the amount of time after which a process can start execution. It is summation of the burst time of previous processes (Processes that came before)

**Process Management**

**Program vs Process:**

A process is a program in execution.

* A process is an ‘active’ entity.
* A program is a ‘passive’ entity.

Process management refers to the techniques and strategies used by organizations to design, monitor, and control their business processes to achieve their goals efficiently and effectively.

* It involves identifying the steps involved in completing a task, assessing the resources required for each step, and determining the best way to execute the task.
* It can help organizations improve their operational efficiency, reduce costs, increase customer satisfaction, and maintain compliance with regulatory requirements.
* It involves analysing the performance of existing processes, identifying bottlenecks, and making changes to optimize the process flow.
* It includes various tools and techniques such as process (mapping, analysis, improvement, automation, and control).

**Process Management**

If OS supports multiple users than services under this are very important . In this regard os has to keep track of all the completing processes , Schedule them, dispatch them one after another. But user should feel that he has the full control of the CPU.

**Characteristics of a Process:**

**Process Id:**

A unique identifier assigned by the operating system.

**Process State:**

Can be ready, running, etc.

**CPU registers:**

Like the Program Counter (CPU registers must be saved and restored when a process is swapped in and out of CPU).

**Accounts information:**

**A**mount of CPU used for process execution, time limits, execution ID etc

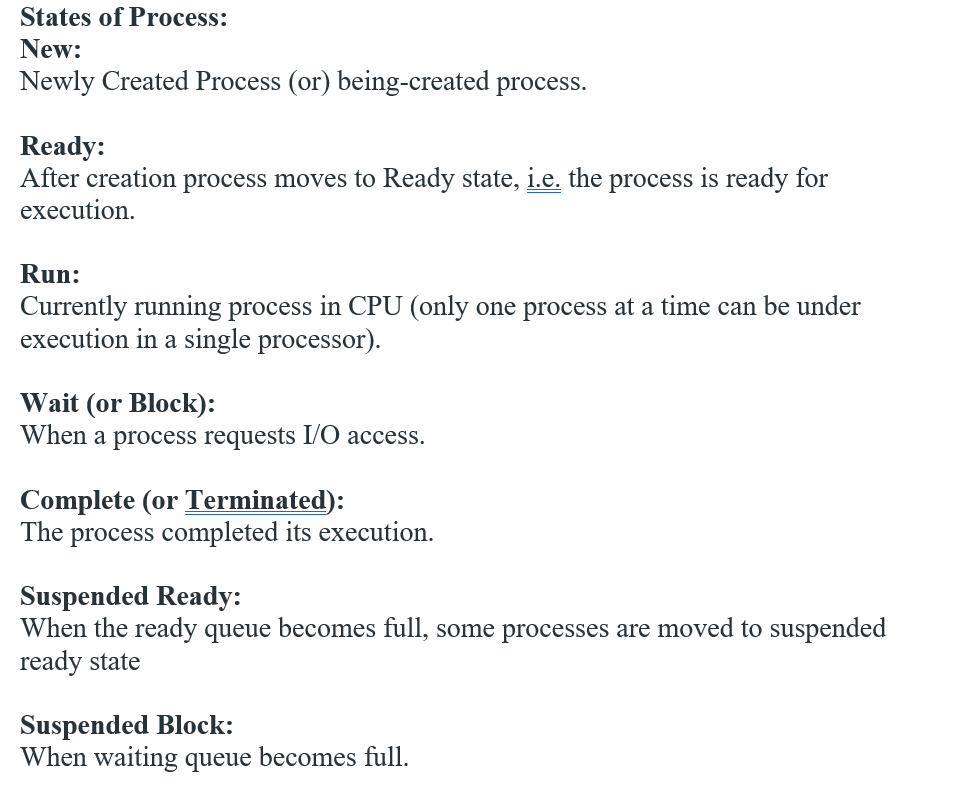
**I/O status information:**

For example, devices allocated to the process,open files, etc

**CPU scheduling information:**

For example,

* Priority (Different processes may have different priorities, i.e. a shorter process assigned high priority in the shortest job first scheduling)
* Every process has its own [process control block](http://en.wikipedia.org/wiki/Process_control_block)(PCB), i.e each process will have a unique PCB. All of the above attributes are part of the PCB.



**Context Switching:**

The process of saving the context of one process and loading the context of another process is known as Context Switching. In simple terms, it is like loading and unloading the process from the running state to the ready state.

**When does context switching happen?**

* When a high-priority process comes to a ready state (i.e. with higher priority than the running process) .
* An Interrupt occurs.
* User and kernel-mode switch (It is not necessary though).
* Preemptive CPU scheduling used.

**CPU-Bound vs I/O-Bound Processes:**

* A CPU-bound process requires more CPU time or spends more time in the running state.
* An I/O-bound process requires more I/O time and less CPU time. An I/O-bound process spends more time in the waiting state.

**Advantages** of Process Management**:**

Improved **Efficiency:** Process management can help organizations identify bottlenecks and inefficiencies in their processes, allowing them to make changes to streamline workflows and increase productivity.

Cost **Savings**: By identifying and eliminating waste and inefficiencies, process management can help organizations reduce costs associated with their business operations.  
 Improved **Quality**: Process management can help organizations improve the quality of their products or services by standardizing processes and reducing errors.  
 Increased Customer **Satisfaction**: By improving efficiency and quality, process management can enhance the customer experience and increase satisfaction.

Compliance with **Regulations**: Process management can help organizations comply with regulatory requirements by ensuring that processes are properly documented, controlled, and monitored.

**Disadvantages** of Process Management**: Time** and **Resource** Intensive: Implementing and maintaining process management initiatives can be time-consuming and require significant resources.

**Resistance** to Change:

Some employees may resist changes to established processes, which can slow down or hinder the implementation of process management initiatives.

**Overemphasis** on Process:

Overemphasis on process can lead to a lack of focus on customer needs and other important aspects of business operations.

**Risk** of Standardization: Standardizing processes too much can limit flexibility and creativity, potentially stifling innovation.

**Difficulty** in Measuring Results: Measuring the effectiveness of process management initiatives can be difficult, making it challenging to determine their impact on organizational performance.

# Process States in OS:

**New (Create):**

In this step, the process is about to be created but not yet created, it is the program that is present in secondary memory that will be picked up by OS to create the process.

**Ready:**

New -> Ready to run. After the creation of a process, the process enters the ready state(i.e. the process is loaded into the main memory).

The process here is ready to run and is waiting to get the CPU time for its execution.

Processes that are ready for execution by the CPU are maintained in a queue for ready processes.

**Run:**

The process is chosen by the CPU for execution and the instructions within the process are executed by any one of the available CPU cores.

**Blocked or Wait:**

Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or waits for the state.

The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.

**Terminated or Completed:**

Process is killed as well as [PCB](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/)is deleted.

**Suspend Ready:**

Process that was initially in the ready state but was swapped out of main memory(refer to Virtual Memory) and placed onto external storage by the scheduler is said to be in suspend ready state.

The process will transition back to a ready state whenever the process is again brought onto the main memory.

**Suspend wait or suspend blocked:**

Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready. **Types of Schedulers: Long-term – performance:** Decides how many processes should be made to stay in the ready state, this decides the degree of multiprogramming. Once a decision is taken it lasts for a long time hence called a long-term scheduler.

**Short-term – Context switching time:**

Short-term scheduler will decide which process to be executed next and then it will call the dispatcher. A dispatcher is a software that moves the process from ready to run and vice versa. In other words, it is context switching.

**Medium-term – Swapping time:**

Suspension decision is taken by the medium-term scheduler. The medium-term scheduler is used for [swapping](https://www.geeksforgeeks.org/swapping-in-operating-system/)which is moving the process from main memory to secondary and vice versa.

### [****Multiprogramming****](https://www.geeksforgeeks.org/multiprogramming-in-operating-system/)

We have many processes ready to run.

**Types :**

**Preemption –**

Process is forcefully removed from CPU. Pre-emotion is also called time sharing or multitasking.

**Non-preemption –**

Processes are not removed until they complete the execution.

### ****Degree of Multiprogramming****

The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming.

### ****Operation on the Process:****

**Creation:**

The process will be ready once it has been created, enter the ready queue (main memory), and be prepared for execution.

**Planning:**

The operating system picks one process to begin executing from among the numerous processes that are currently in the ready queue. Scheduling is the process of choosing the next process to run.

**Application:**

The processor begins running the process as soon as it is scheduled to run. During execution, a process may become blocked or wait, at which point the processor switches to executing the other processes.

**Killing or Deletion:**

The OS will terminate the process once its purpose has been fulfilled. The process’s context will be over there.

**Blocking:**

When a process is waiting for an event or resource, it is blocked.

The os will place it in a blocked state, and it will not be able to execute until the event or resource becomes available.

**Context Switching:**

When the operating system switches from executing one process to another, it must save the current process’s context and load the context of the next process to execute. This is known as context switching.

**Inter-Process Communication:**

Processes may need to communicate with each other to share data or coordinate actions.

The os provides mechanisms for inter-process communication, such as shared memory, message passing, and synchronization primitives.

**Process Synchronization:**

Multiple processes may need to access a shared resource or critical section of code simultaneously. The os provides synchronization mechanisms to ensure that only one process can access the resource or critical section at a time.

**Process States:**

Processes may be in one of several states, including ready, running, waiting, and terminated. The os manages the process states and transitions between them.

## **Features** of the Process State

* A process can move from the running state to the waiting state if it needs to wait for a resource to become available.
* A process can move from the waiting state to the ready state when the resource it was waiting for becomes available.
* A process can move from the ready state to the running state when it is selected by the os for execution.
* The scheduling algorithm used by the os determines which process is selected to execute from the ready state.
* The os may also move a process from the running state to the ready state to allow other processes to execute.
* A process can move from the running state to the terminated state when it completes its execution.
* A process can move from the waiting state directly to the terminated state if it is aborted or killed by the os or another process.
* The process state includes information about the program counter, CPU registers, memory allocation, and other resources used by the process.
* The os maintains a process control block (PCB) for each process, which contains information about the process state, priority, scheduling information, and other process-related data.
* The process state diagram is used to represent the transitions between different states of a process and is an essential concept in process management in os.

**Process Schedulers in OS**

* Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
* It is an essential part of a Multiprogramming operating system.
* Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

**Types** of process schedulers.

**Long Term or job scheduler:**   
It brings the new process to the ‘Ready State’. It controls the ***Degree of Multi-programming***, i.e., the number of processes present in a ready state at any point in time.

It is important that the long-term scheduler make a careful selection of both I/O

I/O-bound tasks are which use much of their time in input and output operations while CPU-bound processes are which spend their time on the CPU.

The job scheduler increases efficiency by maintaining a balance between the two.

They operate at a high level and are typically used in batch-processing systems.

**Short-term or CPU scheduler:**   
It is responsible for selecting one process from the ready state for scheduling it on the running state. Note: It only selects the process to schedule it doesn’t load the process on running.  Here is when all the scheduling algorithms are used.

The CPU scheduler is responsible for ensuring there is no starvation owing to high burst time processes.

***The dispatcher*** is responsible for loading the process selected by the Short-term scheduler on the CPU (Ready to Running State) Context switching is done by the dispatcher only. A dispatcher does the following:

1 Switching context.

2 Switching to user mode.

3 Jumping to the proper location in the newly loaded program.

**Medium-term scheduler:**

It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa).

Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

It is helpful in maintaining a perfect balance between the I/O bound and the CPU bound. It reduces the degree of multiprogramming.

**Some other Schedulers:**

**I/O schedulers:**

I/O schedulers are incharge of managing the execution of I/O operations such as reading and writing to discs or networks.

They can use various algorithms to determine the order in which I/O operations are executed, such as FCFS (First-Come, First-Served) or RR (Round Robin).

**Real-time schedulers:**

In real-time systems, real-time schedulers ensure that critical tasks are completed within a specified time frame.

They can prioritize and schedule tasks using various algorithms such as EDF (Earliest Deadline First) or RM (Rate Monotonic).

**Process Table and Process Control Block (PCB)**

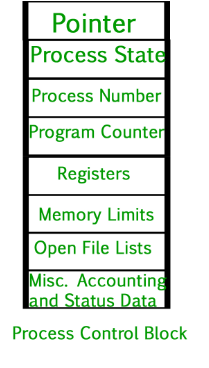
While creating a process the os performs several operations. To identify the processes, it assigns a process identification number (PID) to each process.

As the os supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process’s execution status.

Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc. All this information is required and must be saved when the process is switched from one state to another.

When the process makes a transition from one state to another, the operating system must update information in the process’s PCB. A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc.

The process table is an array of PCBs, that means logically contains a PCB for all of the current processes in the system.



**Pointer –**

It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.

**Process state –**

It stores the respective state of the process.

**Process number –**

Every process is assigned with a unique id known as process ID or PID which stores the process identifier.

**Program counter –**

It stores the counter which contains the address of the next instruction that is to be executed for the process.

**Register –**

These are the CPU registers which includes: accumulator, base, registers and general purpose registers.

**Memory limits –**

This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.

**Open files list –**

This information includes the list of files opened for a process.

**Advantages:**

**Efficient process management:**

The process table and PCB provide an efficient way to manage processes in an operating system.

The process table contains all the information about each process, while the PCB contains the current state of the process, such as the program counter and CPU registers.

**Resource management:**

The process table and PCB allow the operating system to manage system resources, such as memory and CPU time, efficiently. By keeping track of each process’s resource usage, the operating system can ensure that all processes have access to the resources they need.

**Process synchronization:**

The process table and PCB can be used to synchronize processes in an operating system. The PCB contains information about each process’s synchronization state, such as its waiting status and the resources it is waiting for.

**Process scheduling:**

The process table and PCB can be used to schedule processes for execution. By keeping track of each process’s state and resource usage, the operating system can determine which processes should be executed next.

**Disadvantages: Overhead:** The process table and PCB can introduce overhead and reduce system performance. The operating system must maintain the process table and PCB for each process, which can consume system resources.

**Complexity:**

The process table and PCB can increase system complexity and make it more challenging to develop and maintain operating systems.

The need to manage and synchronize multiple processes can make it more difficult to design and implement system features and ensure system stability.

**Scalability:**

The process table and PCB may not scale well for large-scale systems with many processes.

As the number of processes increases, the process table and PCB can become larger and more difficult to manage efficiently.

**Security:**

The process table and PCB can introduce security risks if they are not implemented correctly.

Malicious programs can potentially access or modify the process table and PCB to gain unauthorized access to system resources or cause system instability.

**Miscellaneous accounting and status data –**

This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

The process control block stores the register content also known as execution content of the processor when it was blocked from running.

This execution content architecture enables the operating system to restore a process’s execution context when the process returns to the running state.

When the process makes a transition from one state to another, the operating system updates its information in the process’s PCB.

The operating system maintains pointers to each process’s PCB in a process table so that it can access the PCB quickly.

**Thread**

**What is a Thread?**

Within a program, a thread is a separate execution path.

It is a lightweight process that the operating system can schedule and run concurrently with other threads.

The operating system creates and manages threads, and they share the same memory and resources as the program that created them.

This enables multiple threads to collaborate and work efficiently within a single program.

A thread is a single sequence stream within a process. Threads are also called as lightweight processes as it possess some of the properties of processes.

Each thread belongs to exactly one process.

Multithreading, process can consist of many threads.

These threads run in parallel improving the application performance.

Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.

Threads can share common data so they do not need to use interprocess communication.

Like the processes, threads also have states like ready, executing, blocked etc. priority can be assigned to the threads just like process and highest priority thread is scheduled first.

Each thread has its own Thread Control Block (TCB). Like process, context switch occurs for the thread and register contents are saved in (TCB).

As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

**Why Multithreading?**

* A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads.

For example,

in a browser, multiple tabs can be different threads.

MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

* Multithreading is a technique used in operating systems to improve the performance and responsiveness of computer systems.

Multithreading allows multiple threads (i.e., lightweight processes) to share the same resources of a single process, such as the CPU, memory, and I/O devices.

**Process vs Thread:**

The primary difference is that threads within the same process run in a (shared memory space), while processes run in (separate memory spaces).

Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals).

But, like process, a thread has its own program counter (PC), register set, and stack space.

**Advantages of Thread over Process**

**Responsiveness:**

If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

**Faster context switch:**

Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

**Effective utilization of multiprocessor system:**

If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

**Resource sharing:**

Resources like code, data, and files can be shared among all threads within a process. Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

**Communication:**

Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

**Enhanced throughput of the system:**

If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

**Types of Threads**

1 User Level Thread

2 Kernel Level Thread

**Threads and its types in OS**

Thread is a single sequence stream within a process.

Threads have same properties as of the process so they are called as light weight processes.

Threads are executed one after another but gives the illusion as if they are executing in parallel.

Threads are not independent of each other as they share the code, data, OS resources etc.

States of threads:

A program counter

A register set

A stack space

**Similarity between Threads and Processes –**

* Only one thread or process is active at a time.
* Within process both execute sequential.
* Both can create children.

Both can be **scheduled** by the operating system:

* Both threads and processes can be scheduled by the os to execute on the CPU.
* The os is responsible for assigning CPU time to the threads and processes based on various scheduling algorithms.

Both have their **own execution context**:

* Each thread and process has its own execution context, which includes its own register set, program counter, and stack.
* This allows each thread or process to execute independently and make progress without interfering with other threads or processes.

Both can **communicate** with each other:

* Threads and processes can communicate with each other using various inter-process communication (IPC) mechanisms such as shared memory, message queues, and pipes.
* This allows threads and processes to share data and coordinate their activities.

Both can be **preempted**:

* Threads and processes can be preempted by the operating system, which means that their execution can be interrupted at any time.
* This allows the os to switch to another thread or process that needs to execute.

Both can be **terminated:**

* Threads and processes can be terminated by the os or by other threads or processes.
* When a thread or process is terminated, all of its resources, including its execution context, are freed up and made available to other threads or processes.

**Differences** between Threads and Processes **–**

**Resources:**

* Processes have their own address space and resources, such as memory and file handles,
* Threads share memory and resources with the program that created them.

**Scheduling:**

* Processes are scheduled to use the processor by the os, whereas
* Threads are scheduled to use the processor by the os or the program itself.

**Creation:**

* The operating system creates and manages processes.
* The program or the os creates and manages threads.

**Communication:**

Processes are isolated from one another and must rely on inter-process communication mechanisms,

They generally have more difficulty communicating with one another than threads do.

Threads, on the other hand, can interact with other threads within the same programme directly.

Threads, in general, are lighter than processes and are better suited for concurrent execution within a single programme. Processes are commonly used to run separate programmes or to isolate resources between programmes.

**Types of Threads:**

* **User Level thread (ULT)**

 Is implemented in the user level library, they are not created using the

system calls.

Thread switching does not need to call OS and to cause interrupt to Kernel.

Kernel doesn’t know about the user level thread and manages them as if

they were single-threaded processes.

**Advantages –**

* Can be implemented on an OS that doesn’t support multithreading.
* Simple representation since thread has only program counter, register set, stack space.
* Simple to create since no intervention of kernel.
* Thread switching is fast since no OS calls need to be made.

**Limitations –**

* No or less co-ordination among the threads and Kernel.
* If one thread causes a page fault, the entire process blocks.
* **Kernel Level Thread (KLT) :**

Kernel knows and manages the threads. Instead of thread table in each process, the kernel itself has thread table (a master one) that keeps track of all the threads in the system.

In addition kernel also maintains the traditional process table to keep track of the processes. OS kernel provides system call to create and manage threads.

**Advantages –**

* Since kernel has full knowledge about the threads in the system, scheduler may decide to give more time to processes having large number of threads.
* Good for applications that frequently block.

**Limitations –**

* Slow and inefficient.
* It requires thread control block so it is an overhead.

**Summary:**

Each ULT has a process that keeps track of the thread using the Thread table.

Each KLT has Thread Table (TCB) as well as the Process Table (PCB).

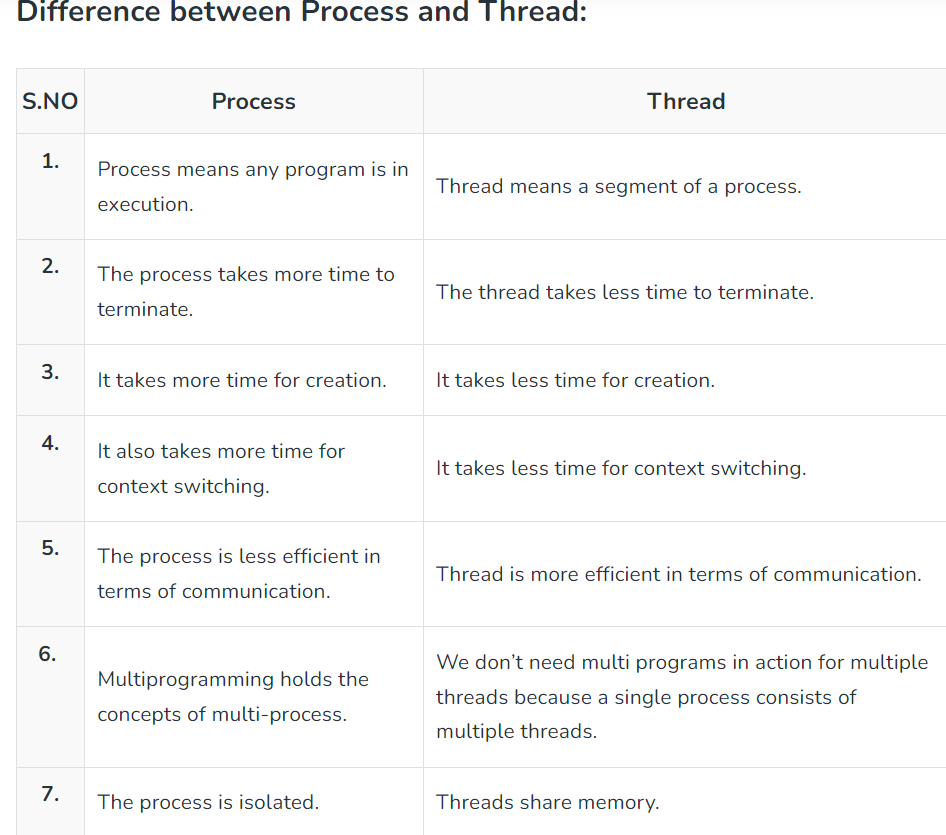
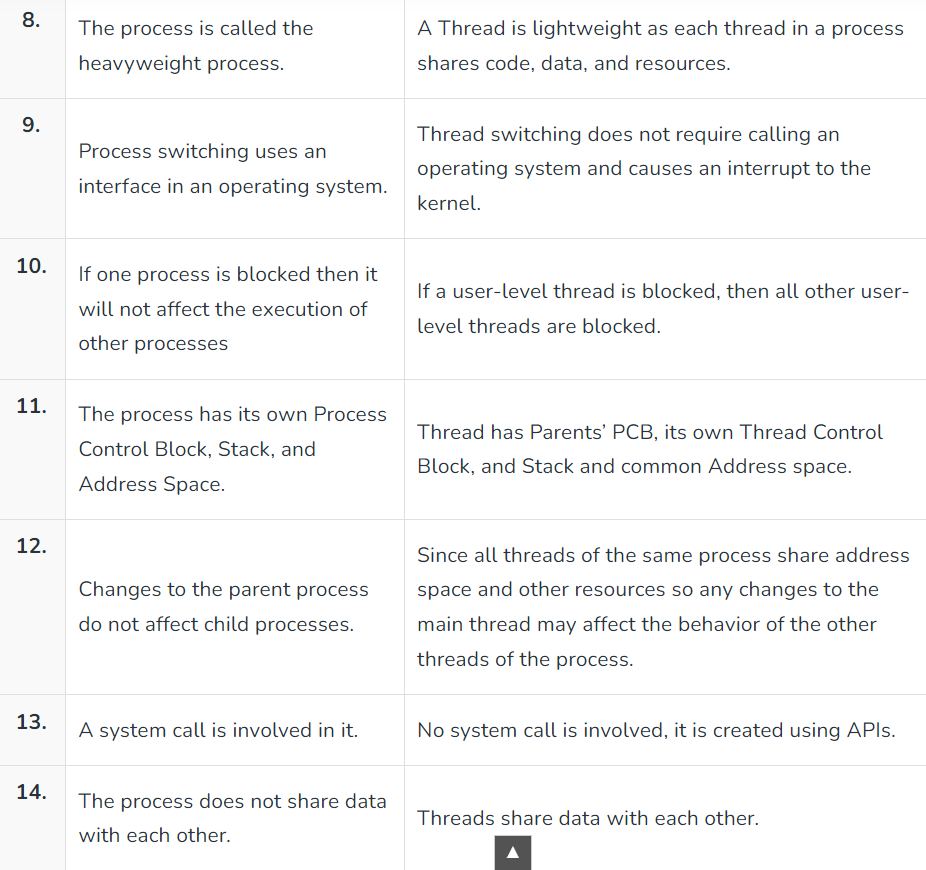
**Difference** between Process and Thread

**Process:**

* Processes are basically the programs that are dispatched from the ready state and are scheduled in the CPU for execution.
* PCB holds the concept of process. A process can create other processes which are known as Child Processes.
* The process takes more time to terminate and it is isolated means it does not share the memory with any other process.
* The process can have the following [states](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/) new, ready, running, waiting, terminated, and suspended.

**Thread:**

* Thread is the segment of a process which means a process can have multiple threads and these multiple threads are contained within a process.
* A thread has three states: Running, Ready, and Blocked.
* The [thread](https://www.geeksforgeeks.org/thread-in-operating-system/) takes less time to terminate as compared to the process but unlike the process, threads do not isolate.

# Process Synchronization

Introduction:

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 such as 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.

Process synchronization is an important aspect of modern operating systems, and it plays a crucial role in ensuring the correct and efficient functioning of multi-process systems.

Types :

**Independent Process**:

The execution of one process does not affect the execution of other processes.

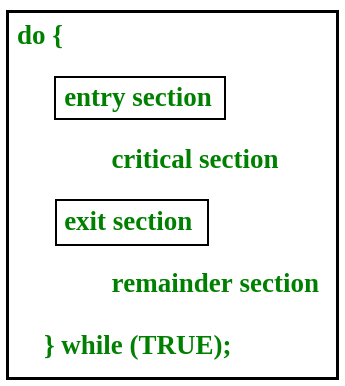
**Cooperative Process**:

A process that can affect or be affected by other processes executing in the system.

Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.

**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.

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/06/critical-section-problem.png)

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 in the critical section next, and the selection can not be postponed indefinitely.

**Bounded Waiting**:

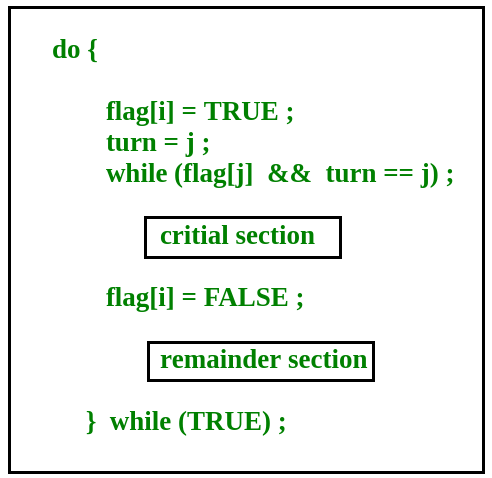
A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**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:

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

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/06/peterson.png)

**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:**

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 A semaphore is a signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that is called the wait function. A semaphore uses two atomic operations, wait and signal for process synchronization. A Semaphore is an integer variable, which can be accessed only through two operations wait() and signal(). Types of semaphores: Binary Semaphores and Counting Semaphores.

**Binary Semaphores:**

They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion.

All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0.

Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of the mutex semaphore to 0 and some other process can enter its critical section.

**Counting Semaphores:**

They can have any value and are not restricted over a certain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses.

The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available.

Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource. Advantages :

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

Disadvantages :

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

# Critical Section in Synchronization

A critical section refers to a segment of code that is executed by multiple concurrent threads or processes, and which accesses shared resources.

These resources may include shared memory, files, or other system resources that can only be accessed by one thread or process at a time to avoid data inconsistency or race conditions.

The critical section must be executed as an atomic operation, which means that once one thread or process has entered the critical section, all other threads or processes must wait until the executing thread or process exits the critical section.

The purpose of synchronization mechanisms is to ensure that only one thread or process can execute the critical section at a time.

The concept of a critical section is central to synchronization in computer systems, as it is necessary to ensure that multiple threads or processes can execute concurrently without interfering with each other.

Various synchronization mechanisms such as semaphores, mutexes, monitors, and condition variables are used to implement critical sections and ensure that shared resources are accessed in a mutually exclusive manner.

The use of critical sections in synchronization can be advantageous in improving the performance of concurrent systems, as it allows multiple threads or processes to work together without interfering with each other.

### [****Critical Section****](http://en.wikipedia.org/wiki/Critical_section)****:****

When more than one [processes](https://www.geeksforgeeks.org/operations-on-processes/)try to access the same code segment that segment is known as the critical section.

The critical section contains shared variables or resources which are needed to be synchronized to maintain the consistency of data variables.

 A critical section is a group of instructions/statements or regions of code that need to be executed atomically such as accessing a resource (file, input or output port, global data, etc.)

In concurrent programming, if one thread tries to change the value of shared data at the same time as another thread tries to read the value (i.e, data race across threads), the result is unpredictable.

The access to such shared variables (shared memory, shared files, shared port, etc.) is to be synchronized.

Although there are some properties that should be followed if any code in the critical section:

1 [**Mutual Exclusion**](https://www.geeksforgeeks.org/mutual-exclusion-in-synchronization/)

2 [**Progress**](https://www.geeksforgeeks.org/progress-of-a-process/)

**3 Bounded Waiting**

The use of critical sections in a program can cause a number of issues, including:

### ****Deadlock:****

When two or more threads or processes wait for each other to release a critical section, it can result in a deadlock situation in which none of the threads or processes can move.

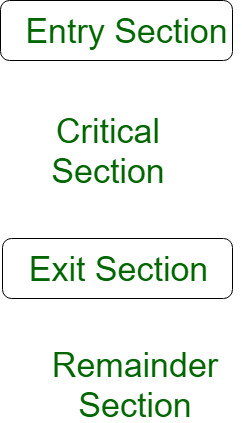
Deadlocks can be difficult to detect and resolve, and they can have a significant impact on a program’s performance and reliability.

### ****Starvation:****

When a thread or process is repeatedly prevented from entering a critical section, it can result in starvation, in which the thread or process is unable to progress. This can happen if the critical section is held for an unusually long period of time, or if a high-priority thread or process is always given priority when entering the critical section.

### ****Overhead:****

When using critical sections, threads or processes must acquire and release locks or semaphores, which can take time and resources. This may reduce the program’s overall performance.



The pseudo-code below: –

do{

flag=1;

while(flag); // (entry section)

// critical section

if (!flag)

// remainder section

} while(true);

A simple solution to the critical section can be thought of as shown below,

acquireLock();

Process Critical Section

releaseLock();

A thread must acquire a lock prior to executing a critical section. The lock can be acquired by only one thread. There are various ways to implement locks in the above pseudo-code.

**Strategies for avoiding problems:**

While (deadlocks, starvation, and overhead) are mentioned as potential issues, but there are more specific strategies for avoiding or mitigating these problems.

Solutions are:

* using timeouts to prevent deadlocks.
* implementing priority inheritance to prevent priority inversion and starvation.
* optimizing lock implementation to reduce overhead.

**Impact on scalability:**

* In process synchronization, a critical section is a section of code that accesses shared resources such as variables or data structures, and which must be executed by only one process at a time to avoid race conditions and other synchronization-related issues.
* A critical section can be any section of code where shared resources are accessed, and it typically consists of two parts: the **entry** section and the **exit** section.
* The entry section is where a process requests access to the critical section,
* The exit section is where it releases the resources and exits the critical section.
* To ensure that only one process can execute the critical section at a time, process synchronization mechanisms such as semaphores and mutexes are used.
* A semaphore is a variable that is used to indicate whether a resource is available or not, while a mutex is a binary semaphore that provides mutual exclusion to shared resources.
* When a process enters a critical section, it must first request access to the semaphore or mutex associated with the critical section.
* If the resource is available, the process can proceed to execute the critical section.
* If the resource is not available, the process must wait until it is released by the process currently executing the critical section.
* Once the process has finished executing the critical section, it releases the semaphore or mutex, allowing another process to enter the critical section if necessary.
* Proper use of critical sections and process synchronization mechanisms is essential in concurrent programming to ensure proper synchronization of shared resources and avoid race conditions, deadlocks, and other synchronization-related issues.

### Advantages of critical section : Prevents race conditions: By ensuring that only one process can execute the critical section at a time, race conditions are prevented, ensuring consistency of shared data.

### Provides mutual exclusion: Critical sections provide mutual exclusion to shared resources, preventing multiple processes from accessing the same resource simultaneously and causing synchronization-related issues.

### Reduces CPU utilization: By allowing processes to wait without wasting CPU cycles, critical sections can reduce CPU utilization, improving overall system efficiency.

### Simplifies synchronization: Critical sections simplify the synchronization of shared resources, as only one process can access the resource at a time, eliminating the need for more complex synchronization mechanisms.

### Disadvantages of critical section: Overhead: Implementing critical sections using synchronization mechanisms like semaphores and mutexes can introduce additional overhead, slowing down program execution.

**Deadlocks**:

Poorly implemented critical sections can lead to deadlocks, where multiple processes are waiting indefinitely for each other to release resources.

Can limit **parallelism**:

If critical sections are too large or are executed frequently, they can limit the degree of parallelism in a program, reducing its overall performance.

Can cause **contention**:

If multiple processes frequently access the same critical section, contention for the critical section can occur, reducing performance.

Overall, critical sections are a useful tool in process synchronization to ensure proper synchronization of shared resources and prevent race conditions.

However, they can also introduce additional overhead and can be prone to synchronization-related issues if not implemented correctly.

# Inter Process Communication (IPC)

An independent process is not affected by the execution of other processes.

A co-operating process can be affected by other executing processes.

The processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience, and modularity.

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them.

Processes can communicate with each other through both:

1. Shared Memory

2. Message passing

An operating system can implement both methods of communication.

Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it.

**i) Shared Memory Method**

**Eg: Producer-Consumer problem** There are two processes: Producer and Consumer.

The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed.

There are two versions of this problem:

1. The first one is known as the unbounded buffer problem in which the Producer can keep on producing items and there is no limit on the size of the buffer.
2. The second one is known as the bounded buffer problem in which the Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it.

The bounded buffer problem.

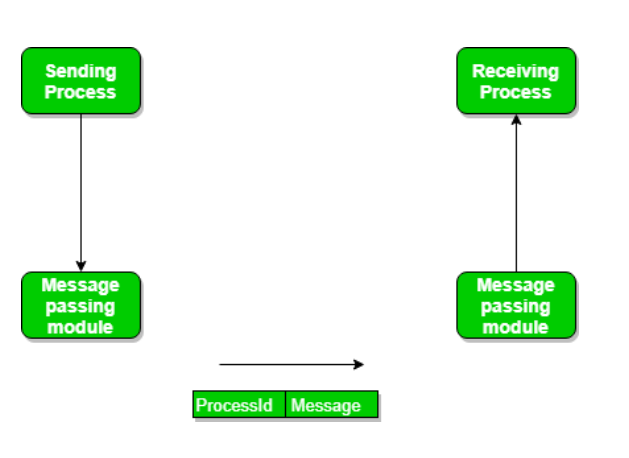
* First, the Producer and the Consumer will share some common memory, then the producer will start producing items.
* If the total produced item is equal to the size of the buffer, the producer will wait to get it consumed by the Consumer.
* Similarly, the consumer will first check for the availability of the item.
* If no item is available, the Consumer will wait for the Producer to produce it.
* If there are items available, Consumer will consume them.

**ii) Messaging Passing Method**

In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

* Establish a communication link (if a link already exists, no need to establish it again.)
* Start exchanging messages using basic primitives.

We need at least two primitives:   
– **send**(message, destination) or **send**(message).  
– **receive**(message, host) or **receive**(message).



* The message size can be of fixed size or of variable size.
* If it is of fixed size, it is easy for an OS designer but complicated for a programmer .
* If it is of variable size then it is easy for a programmer but complicated for the OS designer.

A standard message can have two parts: **header and body.**

* The **header part** is used for storing message type, destination id, source id, message length, and control information.
* The control information contains information like what to do if runs out of buffer space, sequence number, priority.
* Generally, message is sent using FIFO style.

Inter-process communication (IPC) is the mechanism through which processes or threads can communicate and exchange data with each other on a computer or across a network.

IPC is an important aspect of modern operating systems, as it enables different processes to work together and share resources, leading to increased efficiency and flexibility. Advantages of IPC:

* Enables processes to communicate with each other and share resources, leading to increased efficiency and flexibility.
* Facilitates coordination between multiple processes, leading to better overall system performance.
* Allows for the creation of distributed systems that can span multiple computers or networks.
* Can be used to implement various synchronization and communication protocols, such as semaphores, pipes, and sockets.

Disadvantages of IPC:

* Increases system complexity, making it harder to design, implement, and debug.
* Can introduce security vulnerabilities, as processes may be able to access or modify data belonging to other processes.
* Requires careful management of system resources, such as memory and CPU time, to ensure that IPC operations do not degrade overall system performance.
* Can lead to data inconsistencies if multiple processes try to access or modify the same data at the same time.
* Overall, the advantages of IPC outweigh the disadvantages, as it is a necessary mechanism for modern operating systems and enables processes to work together and share resources in a flexible and efficient manner.
* However, care must be taken to design and implement IPC systems carefully, in order to avoid potential security vulnerabilities and performance issues.

# Semaphores

Semaphores are a synchronization mechanism used to coordinate the activities of multiple processes in a computer system. They are used to enforce mutual exclusion, avoid race conditions and implement synchronization between processes.

Semaphores provide two operations: wait (P) and signal (V).

The wait operation decrements the value of the semaphore, and the signal operation increments the value of the semaphore.

When the value of the semaphore is zero, any process that performs a wait operation will be blocked until another process performs a signal operation.

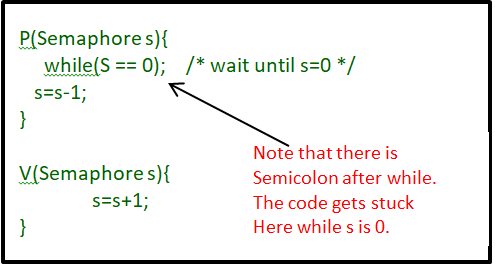
Semaphores are used to implement critical sections, which are regions of code that must be executed by only one process at a time. By using semaphores, processes can coordinate access to shared resources, such as shared memory or I/O devices.

Semaphores are of two types:

**Binary-Semaphore:**  
This is also known as a mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.

**Counting-Semaphore:**   
Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

First, look at two operations that can be used to access and change the value of the semaphore variable.



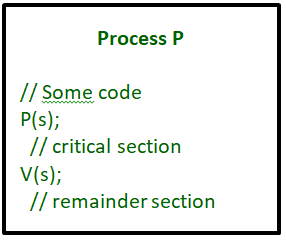
**Some points regarding P and V operation:**

1. P operation is also called wait, sleep, or down operation.

V operation is also called signal, wake-up, or up operation.

2. Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.

3.A critical section is surrounded by both operations to implement process synchronization. See the below image. The critical section of Process P is in between P and V operation.

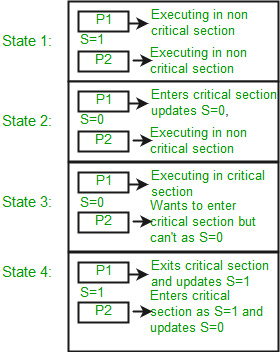


Let there be two processes P1 and P2 and a semaphore s is initialized as 1.

Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0.

Now if P2 wants to enter its critical section then it will wait until s > 0, this can only happen when P1 finishes its critical section and calls V operation on semaphore s.

This way mutual exclusion is achieved. Look at the below image for details which is a Binary semaphore.



The description above is for binary semaphore which can take only two values 0 and 1 and ensure mutual exclusion.

There is one other type of semaphore called counting semaphore which can take values greater than one.

Now suppose there is a resource whose number of instances is 4. Now we initialize S = 4 and the rest is the same as for binary semaphore. Whenever the process wants that resource it calls P or waits for function and when it is done it calls V or signal function. If the value of S becomes zero then a process has to wait until S becomes positive.

Example, Suppose there are 4 processes P1, P2, P3, P4, and they all call wait operation on S(initialized with 4). If another process P5 wants the resource then it should wait until one of the four processes calls the signal function and the value of semaphore becomes positive.

**Limitations :**

* One of the biggest limitations of semaphore is priority inversion.
* Deadlock, suppose a process is trying to wake up another process that is not in a sleep state. Therefore, a deadlock may block indefinitely.
* The operating system has to keep track of all calls to wait and signal the semaphore.

Problem in implementation of a semaphore:

The main problem with semaphores is that they require busy waiting, If a process is in the critical section, then other processes trying to enter the critical section will be waiting until the critical section is not occupied by any process.

Whenever any process waits then it continuously checks for semaphore value (look at this line while (s==0); in P operation) and waste CPU cycle.

There is also a chance of “spinlock” as the processes keep on spins while waiting for the lock.

In order to avoid this another implementation is provided below:

In this implementation whenever the process waits it is added to a waiting queue of processes associated with that semaphore. This is done through the system call block() on that process.

When a process is completed it calls the signal function and one process in the queue is resumed. It uses the wakeup() system call.

**Advantages** of Semaphores:

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

Disadvantages of Semaphores:

* It Can lead to performance degradation due to overhead associated with wait and signal operations.
* Can result in deadlock if used incorrectly.
* A semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.
* It can cause performance issues in a program if not used properly.
* It can be difficult to debug and maintain.
* It can be prone to race conditions and other synchronization problems if not used correctly.
* It can be vulnerable to certain types of attacks, such as denial of service attacks.

# Mutex vs Semaphore

mutexes and semaphores are kernel resources that provide synchronization services (also called *synchronization primitives*).

**The**[**producer-consumer**](http://en.wikipedia.org/wiki/Producer-consumer_problem)**problem:**

A producer thread collects the data and writes it to the buffer.

A consumer thread processes the collected data from the buffer.

The objective is, both the threads should not run at the same time.

**Using Mutex:**

* A mutex provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work.
* As long as the buffer is filled by the producer, the consumer needs to wait, and vice versa.
* At any point of time, only one thread can work with the *entire* buffer.
* The concept can be generalized using semaphore.

**Using Semaphore:**

A semaphore is a generalized mutex. In lieu of a single buffer, we can split the 4 KB buffer into four 1 KB buffers (identical resources).

A semaphore can be associated with these four buffers. The consumer and producer can work on different buffers at the same time.

**Misconception:**

* A mutex is a binary semaphore. *But it is not*!
* The purpose of mutex and semaphore are different. Maybe, due to similarity in their implementation a mutex would be referred to as a binary semaphore.
* A mutex is a **locking mechanism** used to synchronize access to a resource.
* Only one task (can be a thread or process based on OS abstraction) can acquire the mutex.
* It means there is ownership associated with a mutex, and only the owner can release the lock (mutex).
* Semaphore is **signaling mechanism** (“I am done, you can carry on” kind of signal).

For example,

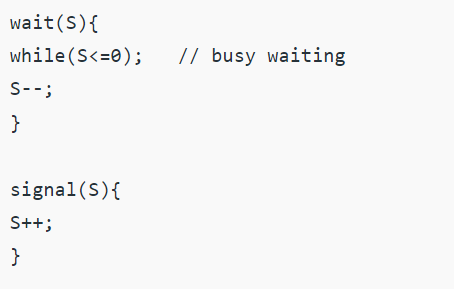
if you are listening to songs (assume it as one task) on your mobile phone and at the same time, your friend calls you, an interrupt is triggered upon which an interrupt service routine (ISR) signals the call processing task to wakeup.

# Producer Consumer Problem using Semaphores

Producer consumer problem is a classical synchronization problem. We can solve this problem by using semaphores.

A [**semaphore**](https://www.geeksforgeeks.org/semaphores-in-process-synchronization/) S is an integer variable that can be accessed only through two standard operations : wait() and signal().

The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1



Types:

**Binary Semaphore :**

This is similar to mutex lock but not the same thing.

It can have only two values – 0 and 1. Its value is initialized to 1.

It is used to implement the solution of critical section problem with multiple processes.

**Counting Semaphore :** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances. 

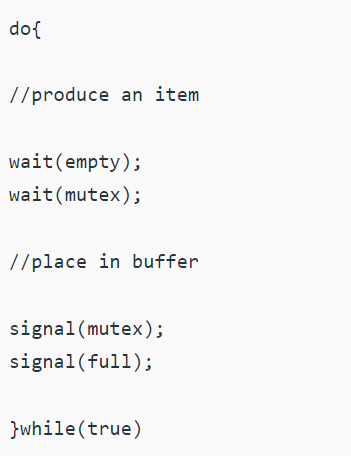
**Problem Statement –**

We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

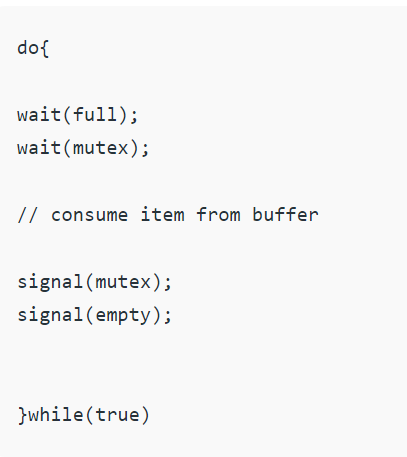
**Initialization of semaphores –**   
mutex = 1   
Full = 0 // Initially, all slots are empty. Thus full slots are 0   
Empty = n // All slots are empty initially

**Solution for Producer –**



When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

**Solution for Consumer –**



As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment.

Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

**Deadlock in OS**

A process in operating system uses resources in the following way.

* Requests a resource
* Use the resource
* Releases the resource

*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.

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.



**Examples Of Deadlock**

1. The system has 2 tape drives. P1 and P2 each hold one tape drive and each needs another one.
2. Semaphores A and B, initialized to 1, P0, and P1 are in deadlock as follows:

* P0 executes wait(A) and preempts.
* P1 executes wait(B).
* Now P0 and P1 enter in deadlock.

| **P0** | **P1** |
| --- | --- |
| wait(A); | wait(B) |
| wait(B); | wait(A) |

 Deadlock occurs if both processes progress to their second request.

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

***Mutual Exclusion:***

Two or more resources 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 waiting for each other in circular form.

**Methods for handling deadlock** :  
  
**1) Deadlock prevention or avoidance:**

**Prevention:**

The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free.  
One can zoom into each category individually, Prevention is done by negating one of the above-mentioned necessary conditions for deadlock.

Prevention can be done in four different ways:

      1. Eliminate mutual exclusion                                        3. Allow preemption

 2. Solve hold and Wait                                                   4. Circular wait

**Avoidance:**  
Avoidance is kind of futuristic. By using the strategy of “Avoidance”, we have to make an assumption.

We need to ensure that all information about resources that the process will need is known to us before the execution of the process.

We use Banker’s algorithm to avoid deadlock.

In prevention and avoidance, we get the correctness of data but performance decreases.

**2) Deadlock detection and recovery:**

If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

* In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
* If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.
* In Deadlock detection and recovery, we get the correctness of data but performance decreases.

**Safe State:**

A safe state can be defined as a state in which there is no deadlock.

It is achievable if:

If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.

All the requested resources are allocated to the process.

**Deadlock Avoidance Algorithm/**

**What is a deadlock detection algorithm in os?**  
A deadlock detection algorithm is a technique used by an os to identify deadlocks in the system.

This algorithm checks the status of processes and resources to determine whether any deadlock has occurred and takes appropriate actions to recover from the deadlock. The algorithm employs several times varying data structures:

**Available –**  
A vector of length m indicates the number of available resources of each type.

**Allocation –**  
An n\*m matrix defines the number of resources of each type currently allocated to a process. The column represents resource and rows represent a process.

**Request –**  
An n\*m matrix indicates the current request of each process. If request[i][j] equals k then process Pi is requesting k more instances of resource type Rj.

[**Bankers Algorithm**](https://www.geeksforgeeks.org/operating-system-bankers-algorithm/)**:**

Now, the Bankers algorithm includes a **Safety Algorithm / Deadlock**

**Banker’s Algorithm:**

A resource allocation algorithm that ensures that the system is always in a safe state, where deadlocks cannot occur.

The choice of algorithm depends on the specific requirements of the system and the trade-offs between performance, complexity, and accuracy.

If a system does not employ either a deadlock prevention or [deadlock avoidance algorithm](https://www.geeksforgeeks.org/operating-system-bankers-algorithm-print-safe-state-safe-sequences/) then a deadlock situation may occur.

In this case-

Apply an algorithm to examine the system’s state to determine whether deadlock has occurred.

Apply an algorithm to recover from the deadlock.

**Advantages** of Deadlock Detection :

Improved **System Stability**:

Deadlocks are a major concern in operating systems, and detecting and resolving deadlocks can help to improve the stability of the system.

Better **Resource Utilization**:

By detecting deadlocks and freeing resources, the operating system can ensure that resources are efficiently utilized and that the system remains responsive to user requests.

**Easy Implementation**:

Some deadlock detection algorithms, such as the Wait-For Graph, are relatively simple to implement and can be used in a wide range of operating systems and systems with different resource allocation and synchronization requirements.

**Disadvantages** of Deadlock Detection : Performance **Overhead:**

Deadlock detection algorithms can introduce a significant overhead in terms of performance, as the system must regularly check for deadlocks and take appropriate action.

**Complexity**:

Some deadlock detection algorithms, such as the Resource Allocation Graph or Timestamping, are more complex to implement and require a deeper understanding of the system and its behavior.

Overall, the choice of deadlock detection algorithm depends on the specific requirements of the system, the trade-offs between performance, complexity, and accuracy, and the risk tolerance of the system.

The os must balance these factors to ensure that deadlocks are detected and resolved effectively and efficiently.

Deadlock **Detection** And **Recovery**

Deadlock detection and recovery is the process of detecting and resolving deadlocks in an operating system.

A deadlock occurs when two or more processes are blocked, waiting for each other to release the resources they need. This can lead to a system-wide stall, where no process can make progress.

**Prevention**:

The operating system takes steps to prevent deadlocks from occurring by ensuring that the system is always in a safe state, where deadlocks cannot occur. This is achieved through resource allocation algorithms such as the Banker’s Algorithm.

**Detection and Recovery:**

If deadlocks do occur, the operating system must detect and resolve them. Deadlock detection algorithms, such as the Wait-For Graph, are used to identify deadlocks, and recovery algorithms, such as the Rollback and Abort algorithm, are used to resolve them.

The recovery algorithm releases the resources held by one or more processes, allowing the system to continue to make progress.

**Difference Between Prevention and Detection/Recovery:**

* Prevention aims to avoid deadlocks altogether by carefully managing resource allocation, while
* Detection and recovery aim to identify and resolve deadlocks that have already occurred.

Deadlock detection and recovery is an important aspect of os design and management, as it affects the stability and performance of the system.

The choice of deadlock detection and recovery approach depends on the specific requirements of the system and the trade-offs between performance, complexity, and risk tolerance.

The operating system must balance these factors to ensure that deadlocks are effectively detected and resolved.

**Deadlock Detection :**

**1. If resources have a single instance –**  
In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph.

The presence of a cycle in the graph is a sufficient condition for deadlock. 



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed. 

**If there are multiple instances of resources –**  
Detection of the cycle is necessary but not a sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

**Deadlock Recovery :**   
A traditional operating system such as Windows doesn’t deal with deadlock recovery as it is a time and space-consuming process.

* Real-time operating systems use Deadlock recovery.

**Killing the process –**  
Killing all the processes involved in the deadlock or Killing process one by one.

After killing each process check for deadlock again and keep repeating the process till the system recovers from deadlock.

Killing all the processes one by one helps a system to break circular wait conditions.

**Resource Preemption –**  
Resources are preempted from the processes involved in the deadlock, and preempted resources are allocated to other processes so that there is a possibility of recovering the system from the deadlock.

* In this case, the system goes into starvation.

# Deadlock Prevention And Avoidance **Characteristics**  Deadlock has following characteristics.

Mutual Exclusion Hold and Wait

No pre-emption Circular wait

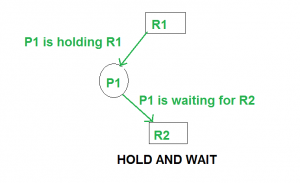
**Deadlock Prevention**   
We can prevent Deadlock by eliminating any of the above four conditions. 

**Eliminate Mutual Exclusion**It is not possible to dis-satisfy the mutual exclusion 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.

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 required by other high priority processes.

**Eliminate Circular Wait**   
Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

**Detection and Recovery:**

Another approach to dealing with deadlocks is to detect them when they occur and recover from them.

This can involve killing one or more of the processes involved in the deadlock or releasing some of the resources they hold.

**Deadlock Avoidance**   
Deadlock avoidance can be done with Banker’s Algorithm.

**Need = maximum resources - currently allocated resources.**

**Note:** Deadlock prevention is more strict than Deadlock Avoidance.

# Banker’s Algorithm in OS

**Why it is named so?**

Banker’s algorithm is named so because it is used in the banking system to check whether a loan can be sanctioned to a person or not.

Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that the bank has and if the remaining amount is greater than S then only the loan is sanctioned.

It is done because if all the account holders come to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in a safe state always.

It is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn’t allow the request made by the process.

**Inputs to Banker’s Algorithm:**

* Max need of resources by each process.
* Currently, allocated resources by each process.
* Max free available resources in the system.

**The request will only be granted under the below condition:**

* If the request made by the process is less than equal to max need to that process.
* If the request made by the process is less than equal to the freely available resource in the system.

Banker’s algorithm consists of a Safety algorithm and a Resource request algorithm.

* As the processes enter the system, they must predict the maximum number of resources needed which is not impractical to determine.
* In this algorithm, the number of processes remain fixed which is not possible in interactive systems.
* This algorithm requires that there should be a fixed number of resources to allocate. If a device breaks and becomes suddenly unavailable the algorithm would not work.
* Overhead cost incurred by the algorithm can be high when there are many processes and resources because it has to be invoked for every processes.

**Memory Management**

# Fixed (static) Partitioning in OS:

Fixed partitioning, also known as static partitioning. It is a memory allocation technique used in operating systems to divide the physical memory into fixed-size partitions or regions, each assigned to a specific process or user.

Each partition is typically allocated at system boot time and remains dedicated to a specific process until it terminates or releases the partition.

In fixed partitioning, the memory is divided into fixed-size chunks, with each chunk being reserved for a specific process.

When a process requests memory, the operating system assigns it to the appropriate partition.

Each partition is of the same size, and the memory allocation is done at system boot time.

It is predictable, meaning the operating system can ensure a minimum amount of memory for each process. Third, it can prevent processes from interfering with each other’s memory space, improving the security and stability of the system.

Overall, fixed partitioning is a useful memory allocation technique in situations where the number of processes is fixed, and the memory requirements for each process are known in advance. It is commonly used in embedded systems, real-time systems, and systems with limited memory resources.

In os, Memory Management is the function responsible for allocating and managing a computer’s main memory. Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.

There are two Memory Management Techniques:

**Contiguous**, and **Non-Contiguous**.

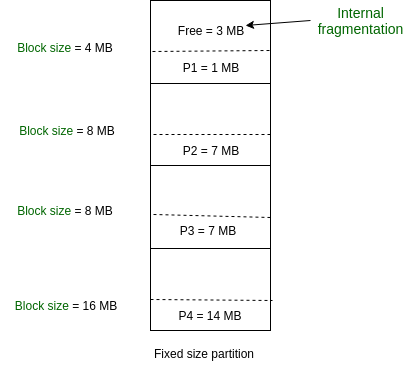
In Contiguous Technique, executing process must be loaded entirely in the main memory. Contiguous Technique can be divided into:

Fixed (or static) partitioning  Variable (or dynamic) partitioning 

**Fixed Partitioning:**   
This is the oldest and simplest technique used to put more than one process in the main memory.

In this partitioning, the number of partitions (non-overlapping) in RAM is **fixed but the size** of each partition may or **may not be the same**.

As it is a **contiguous** allocation, hence no spanning is allowed. Here partitions are made before execution or during system configure.



As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory.   
Hence, Internal Fragmentation in first block is (4-1) = 3MB.   
Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

Suppose process P5 of size 7MB comes. But this process cannot be accommodated in spite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

**Advantages :**

**Easy to implement:**

The algorithms needed to implement Fixed Partitioning are straightforward and easy to implement.

**Low overhead:**

Fixed Partitioning requires minimal overhead, which makes it ideal for systems with limited resources.

**Predictable:**

Fixed Partitioning ensures a predictable amount of memory for each process.

**No external fragmentation:**

Fixed Partitioning eliminates the problem of external fragmentation.

**Suitable for systems with a fixed number of processes:**

Fixed Partitioning is well-suited for systems with a fixed number of processes and known memory requirements.

**Prevents processes from interfering with each other:**

Fixed Partitioning ensures that processes do not interfere with each other’s memory space.

**Efficient use of memory:**

Fixed Partitioning ensures that memory is used efficiently by allocating it to fixed-sized partitions.

**Good for batch processing:**

Fixed Partitioning is ideal for batch processing environments where the number of processes is fixed.

**Better control over memory allocation:**

Fixed Partitioning gives the operating system better control over the allocation of memory.

**Easy to debug:**  Fixed Partitioning is easy to debug since the size and location of each process are predetermined.

**Disadvantages:**

**Internal Fragmentation:**   
Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation. 

**External Fragmentation:**   
The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed). 

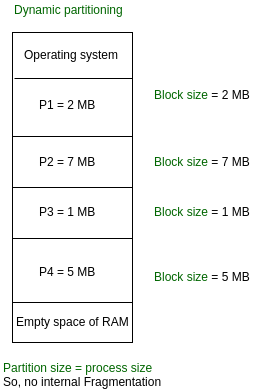
**Limit process size:**   
Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid. 

**Limitation on Degree of Multiprogramming:**   
Partitions in Main Memory are made before execution or during system configure. Main Memory is divided into a fixed number of partitions. Suppose if there are partitions in RAM and are the number of processes, then condition must be fulfilled. Number of processes greater than the number of partitions in RAM is invalid in Fixed Partitioning.

**Variable Partitioning –**

It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

1. Initially RAM is empty and partitions are made during the run-time according to process’s need instead of partitioning during system configure.
2. The size of partition will be equal to incoming process.
3. The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.
4. Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory’s size.



**Advantages of Variable Partitioning –**

**No Internal Fragmentation:**

In variable Partitioning, space in main memory is allocated strictly according to the need of process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.

**No restriction on Degree of Multiprogramming:**

More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is empty.

**No Limitation on the size of the process:**

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be loaded and process can not be divided as it is invalid in contiguous allocation technique. Here, In variable partitioning, the process size can’t be restricted since the partition size is decided according to the process size.

**Disadvantages:**

**Difficult Implementation:**

Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.

**External Fragmentation:**

There will be external fragmentation inspite of absence of internal fragmentation.

### some important points about variable (or dynamic) partitioning in OS

### Variable (or dynamic) partitioning is a memory allocation technique that allows memory partitions to be created and resized dynamically as needed.

The operating system maintains a table of free memory blocks or holes, each of which represents a potential partition. When a process requests memory, the operating system searches the table for a suitable hole that can accommodate the requested amount of memory.

Dynamic partitioning reduces internal fragmentation by allocating memory more efficiently, allows multiple processes to share the same memory space, and is flexible in accommodating processes with varying memory requirements.

dynamic partitioning can also lead to external fragmentation and requires more complex memory management algorithms, which can make it slower than fixed partitioning.

# Non-Contiguous Allocation in Operating System

Non-contiguous allocation, also known as dynamic or linked allocation, is a memory allocation technique used in operating systems to allocate memory to processes that do not require a contiguous block of memory. In this technique, each process is allocated a series of non-contiguous blocks of memory that can be located anywhere in the physical memory.

Non-contiguous allocation involves the use of pointers to link the non-contiguous memory blocks allocated to a process. These pointers are used by the operating system to keep track of the memory blocks allocated to the process and to locate them during the execution of the process.

### There are several advantages to non-contiguous allocation.

First, it reduces internal fragmentation since memory blocks can be allocated as needed, regardless of their physical location. Second, it allows processes to be allocated memory in a more flexible and efficient manner since the operating system can allocate memory to a process wherever free memory is available.

### Disadvantages. It can lead to external fragmentation, where the available memory is broken into small, non-contiguous blocks, making it difficult to allocate large blocks of memory to a process. Additionally, the use of pointers to link memory blocks can introduce additional overhead, leading to slower memory allocation and deallocation times.

In summary, non-contiguous allocation is a useful memory allocation technique in situations where processes do not require a contiguous block of memory. It is commonly used in operating systems, such as Unix and Linux, where processes often require variable amounts of memory that are not contiguous.

[Paging](https://www.geeksforgeeks.org/operating-system-paging/) and [Segmentation](https://www.geeksforgeeks.org/operating-systems-segmentation/) are the two ways that allow a process’s physical address space to be non-contiguous.

It has the **advantage** of reducing memory wastage but it increases the overheads due to address translation.

It slows the execution of the memory because time is consumed in address translation.

In non-contiguous allocation, the Operating system needs to maintain the table which is called the **Page Table** for each process which contains the base address of each block that is acquired by the process in memory space.

In non-contiguous memory allocation, different parts of a process are allocated to different places in Main Memory. Spanning is allowed which is not possible in other techniques like Dynamic or Static Contiguous memory allocation. That’s why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.

There are five types of Non-Contiguous Allocation of Memory in the Operating System:

1. Paging
2. Multilevel Paging
3. Inverted Paging
4. Segmentation
5. Segmented Paging

**Working:**   
Here a process can be spanned across different spaces in the main memory in a non-consecutive manner. Suppose process P of size 4KB. Consider main memory has two empty slots each of size 2KB. Hence total free space is, 2\*2= 4 KB. In contiguous memory allocation, process P cannot be accommodated as spanning is not allowed.

In contiguous allocation, space in memory should be allocated to the whole process. If not, then that space remains unallocated. But in Non-Contiguous allocation, the process can be divided into different parts and hence filling the space in the main memory. In this example, process P can be divided into two parts of equal size – 2KB. Hence one part of process P can be allocated to the first 2KB space of main memory and the other part of the process can be allocated to the second 2KB space of main memory. The below diagram will explain in a better way:



But, in what manner we divide a process to allocate them into main memory is very important to understand. The process is divided after analysing the number of empty spaces and their size in the main memory. Then only we do divide our process. It is a very time-consuming process. Their number as well as their sizes changing every time due to execution of already present processes in main memory.

In order to avoid this time-consuming process, we divide our process in secondary memory in advance before reaching the main memory for its execution. Every process is divided into various parts of equal size called Pages. We also divide our main memory into different parts of equal size called Frames. It is important to understand that:

Size of page in process

= Size of frame in memory

Although their numbers can be different. Below diagram will make you understand it in a better way: consider empty main memory having a size of each frame is 2 KB, and two processes P1 and P2 are 2 KB each.



Resolvent main memory,



Concluding, we can say that Paging allows the memory address space of a process to be non-contiguous. Paging is more flexible as only pages of a process are moved. It allows more processes to reside in main memory than Contiguous memory allocation.

# Logical and Physical Address in Operating System

### Introduction:

In operating systems, logical and physical addresses are used to manage and access memory.

**Logical address**: A logical address, also known as a virtual address, is an address generated by the CPU during program execution. It is the address seen by the process and is relative to the program’s address space. The process accesses memory using logical addresses, which are translated by the operating system into physical addresses.

**Physical address**: A physical address is the actual address in main memory where data is stored. It is a location in physical memory, as opposed to a virtual address. Physical addresses are used by the memory management unit (MMU) to translate logical addresses into physical addresses.  
The translation from logical to physical addresses is performed by the operating system’s memory management unit.

The MMU uses a page table to translate logical addresses into physical addresses. The page table maps each logical page number to a physical frame number.

**Similarity between logical and physical addresses**

* Both logical and physical addresses are used to identify a specific location in memory.
* Both logical and physical addresses can be represented in different formats, such as binary, hexadecimal, or decimal.
* Both logical and physical addresses have a finite range, which is determined by the number of bits used to represent them.

### Important points about logical and physical addresses

The use of logical addresses provides a layer of abstraction that allows processes to access memory without knowing the physical memory location.  
Logical addresses are mapped to physical addresses using a page table.

The page table contains information about the mapping between logical and physical addresses.  
The MMU translates logical addresses into physical addresses using the page table. This translation is transparent to the process and is performed by hardware.  
The use of logical and physical addresses allows the operating system to manage memory more efficiently by using techniques such as paging and segmentation.

**Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.

This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program’s perspective.   
The hardware device called [Memory-Management](https://www.geeksforgeeks.org/memory-management-in-operating-system/) Unit is used for mapping logical address to its corresponding physical address.

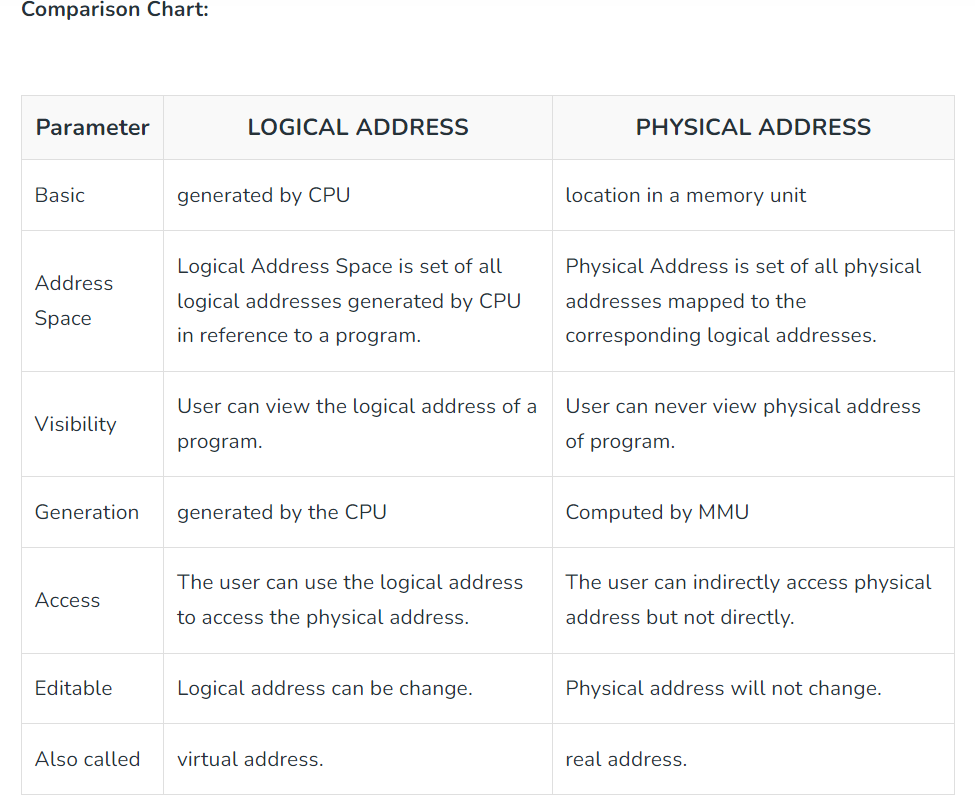
**Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address.

The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.

The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

**Differences Between Logical and Physical Address**

* Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
* Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
* The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
* Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time [address binding](https://www.geeksforgeeks.org/address-binding-and-its-types/) method.
* The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).



# Paging in OS

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.

The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging.

The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory.

This scheme permits the physical address space of a process to be non – contiguous.

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.

The Logical address Space is also splitted into fixed-size blocks, called **pages**.

* Page Size = Frame Size

Address generated by CPU is divided into

* **Page number(p):**

Number of bits required to represent the pages in Logical Address Space or Page number

* **Page offset(d):**

Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

**Frame range:**

This is the variety of the frame within the physical cope with area that consists of the byte or phrase being addressed.

**Frame offset:**

The number of bits required to represent the frame offset relies upon on the size of every frame.

Paging is a memory management technique used in operating systems to manage memory and allocate memory to processes. In paging, memory is divided into fixed-size blocks called pages, and processes are allocated memory in terms of these pages. Each page is of the same size, and the size is typically a power of 2, such as 4KB or 8KB.

In paging, the physical memory is divided into fixed-size blocks called page frames, which are the same size as the pages used by the process.

The process’s logical address space is also divided into fixed-size blocks called pages, which are the same size as the page frames.

When a process requests memory, the operating system allocates one or more page frames to the process and maps the process’s logical pages to the physical page frames.

The mapping between logical pages and physical page frames is maintained by the page table, which is used by the memory management unit to translate logical addresses into physical addresses.

The page table maps each logical page number to a physical page frame number.

important points about paging :

**Reduces internal fragmentation:**

Paging facilitates to lessen internal fragmentation by using allocating memory in fixed-size blocks (pages),         which might be usually a whole lot smaller than the size of the process’s facts segments. This lets in for greater efficient use of memory in view       that there are fewer unused bytes in each block.

**Enables reminiscence to be allotted on call for:**

Paging enables memory to be allocated on call for, this means that that memory is most             effective allocated when it’s far needed. This allows for extra efficient use of memory in view that only the pages that are absolutely used by the       manner want to be allocated inside the physical memory.

**3 Protection and sharing of memory:**

Paging allows for protection and sharing of reminiscence between methods, as each procedure has its  own web page table that maps its logical deal with area to its physical address space. This permits for techniques to proportion facts at the same     time as preventing unauthorized get right of entry to to every other’s memory.

**4. External fragmentation:**

Paging can result in outside fragmentation, wherein memory turns into fragmented into small, non-contiguous blocks. This can make it difficult to allocate massive blocks of reminiscence to a method seeing that there may not be enough contiguous free memory to be had.

**5. Overhead :**

Paging involves overhead because of the renovation of the web page table and the translation of logical addresses to physical addresses. The working device must maintain the page table for each manner and perform deal with translation whenever a procedure  accesses memory, that can slow down the machine.

**Frame Number:**

It gives the frame number in which the current page you are looking for is present. The number of bits required depends on the number of frames. Frame bit is also known as address translation bit.

Number of bits for frame = Size of physical memory/frame size

A Page Table is a data structure used by the OS to keep track of the mapping between virtual addresses used by a process and the corresponding physical addresses in the system’s memory.

A Page Table Entry (PTE) is an entry in the Page Table that stores information about a particular page of memory. Each PTE contains information such as the physical address of the page in memory, whether the page is present in memory or not, whether it is writable or not, and other access permissions.

The size and format of a PTE can vary depending on the architecture of the system and the operating system used. In general, a PTE contains enough information to allow the operating system to manage memory efficiently and to protect the system from malicious or accidental access to memory.

The number of Page Table Entries in a Page Table depends on the size of the virtual address space used by a process and the size of the memory pages used by the system. For example, if the virtual address space of a process is 32 bits, and the system uses 4 KB pages, then the Page Table will have 2^20 (1 million) entries, with each entry being 4 bytes in size.

Advantages of using a Page Table:

**Efficient use of memory**:

Virtual memory allows the operating system to allocate only the necessary amount of physical memory needed by a process, which reduces memory waste and increases overall system performance.

**Protection**:

Page Tables allow the operating system to control access to memory and protect sensitive data from unauthorized access. Each PTE can be configured with access permissions, such as read-only or no access, to prevent accidental or malicious modification of memory.

**Flexibility:**

Virtual memory allows multiple processes to share the same physical memory space, which increases system flexibility and allows for better resource utilization.

**Address translation:**

Page Tables provide the mechanism for translating virtual addresses used by a process into physical addresses in memory, which allows for efficient use of memory and simplifies memory management.

**SEGMENTS**

* A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments.
* Segmentation gives the user’s view of the process which paging does not give. Here the user’s view is mapped to physical memory.

Types of segmentation:

**Virtual memory segmentation:**

Each process is divided into a number of segments, not all of which are resident at any one point in time.

**Simple segmentation:**

* Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.
* There is no simple relationship between logical addresses and physical addresses in segmentation.

**Segment Table:**

* A table stores the information about all such segments and is called Segment Table.
* It maps two-dimensional Logical address into one-dimensional Physical address. It’s each table entry has:

**Base Address:**

Itcontains the starting physical address where the segments reside in memory.

**Limit:**

It specifies the length of the segment.

CPU Generated address is divided into:

**Segment number (s):**

Number of bits required to represent the segment.

**Segment offset (d):**

Number of bits required to represent the size of the segment.

**Advantages** :

* No Internal fragmentation.
* Segment Table consumes less space in comparison to Page table in paging.
* segmentation improves CPU utilization.
* Users can divide user programs into modules via segmentation. These modules are nothing more than the separate processes’ codes.
* The user specifies the segment size, whereas in paging, the hardware determines the page size.

Segmentation is a method that can be used to segregate data from security operations.

**Flexibility**:

* Segmentation provides a higher degree of flexibility than paging.
* Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.

**Sharing**:

* Segmentation allows for sharing of memory segments between processes.
* This can be useful for inter-process communication or for sharing code libraries.

**Protection**:

* Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process’s memory segment.
* This can help increase the security and stability of the system.

**Disadvantages :**

* As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing **External fragmentation**.
* Overhead is associated with keeping a segment table for each activity.
* Due to the need for two memory accesses, one for the segment table and the other for main memory, access time to retrieve the instruction increases.

**Fragmentation**:

* Segmentation can lead to external fragmentation as memory becomes divided into smaller segments.
* This can lead to wasted memory and decreased performance.

**Overhead**:

* The use of a segment table can increase overhead and reduce performance.
* Each segment table entry requires additional memory, and accessing the table to retrieve memory locations can increase the time needed for memory operations.

**Complexity**:

* Segmentation can be more complex to implement and manage than paging.
* In particular, managing multiple segments per process can be challenging, and the potential for segmentation faults can increase as a result.

# Virtual Memory in OS

* Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory.
* It is a technique that is implemented using both hardware and software.
* It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.
* All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time.
* A process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.

A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution.

The combination of dynamic run-time address translation and use of page or segment table permits this.

This means that the required pages need to be loaded into memory whenever required.

Virtual memory is implemented using Demand Paging or Demand Segmentation.

**Demand Paging:**

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

steps are as follows:

* If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
* The OS puts the interrupted process in a blocking state.
* For the execution to proceed the OS must bring the required page into the memory.
* The OS will search for the required page in the logical address space.
* The required page will be brought from logical address space to physical address space.
* The page replacement algorithms are used for the decision-making of replacing the page in physical address space.
* The page table will be updated accordingly.
* The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

**Advantages:**

* More processes may be maintained in the main memory.
* we are going to load only some of the pages of any particular process, there is room for more processes.
* This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
* A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
* It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

**Disadvantages:**

* It can slow down the system performance, as data needs to be constantly transferred between the physical memory and the hard disk.
* It can increase the risk of data loss or corruption, as data can be lost if the hard disk fails or if there is a power outage while data is being transferred to or from the hard disk.
* It can increase the complexity of the memory management system, as the operating system needs to manage both physical and virtual memory.

**Page Fault Service Time:**

The time taken to service the page fault is called page fault service time. The page fault service time includes the time taken to perform all the above steps.

**Swapping:**

* Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the normal page replacement process.
* Suspending a process ensures that it is not runnable while it is swapped out.
* At some later time, the system swaps back the process from the secondary storage to the main memory.

**Thrashing:**

* When a process is busy swapping pages in and out then this situation is called thrashing.
* At any given time, only a few pages of any process are in the main memory and therefore more processes can be maintained in memory.
* Furthermore, time is saved because unused pages are not swapped in and out of memory.
* In the steady-state practically, all of the main memory will be occupied with process pages, so that the processor and OS have direct access to as many processes as possible.
* Thus when the OS brings one page in, it must throw another out.
* If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing.
* The performance of a virtual memory management system depends on the total number of page faults, which depend on “paging policies” and “frame allocation”

**Frame Allocation:**

Number of frames allocating to each process  in either static or dynamic.

* **Static Allocation:**The number of frame allocation to a process is fixed.
* **Dynamic Allocation:**The number of frames allocated to a process changes.

**Paging Policies:**

* **Fetch Policy:**It decides when a page should be loaded into memory.
* **Replacement Policy:**It decides which page in memory should be replaced.
* **Placement Policy:**It decides where in memory should a page be loaded.

# Page Replacement Algorithms

In an os that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

**Page Fault:**

* A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.
* Since actual physical memory is much smaller than virtual memory, page faults happen.
* In case of a page fault, OS might have to replace one of the existing pages with the newly needed page.
* Different page replacement algorithms suggest different ways to decide which page to replace.
* The target for all algorithms is to reduce the number of page faults.

**Page Replacement Algorithms:**

**First In First Out (FIFO):**

* This is the simplest page replacement algorithm.
* In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.
* When a page needs to be replaced page in the front of the queue is selected for removal.

**Optimal Page replacement:**

* In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Least Recently Used:**

* In this algorithm, page will be replaced which is least recently used.

**Most Recently Used (MRU):**

* In this algorithm, page will be replaced which has been used recently. Belady’s anomaly can occur in this algorithm.

# Disk Scheduling Algorithms

**Disk scheduling**is done by OS to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Disk scheduling is important because:

* Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
* Two or more requests may be far from each other so can result in greater disk arm movement.
* Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

some of the important terms:

**SeekTime:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

**Rotational Latency:**

Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

**TransferTime:**  Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

**Disk Access Time:**

Disk Access Time = Seek Time + Rotational Latency + Transfer Time

Total Seek Time = Total head Movement \* Seek Time

**Disk Response Time:**

Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time*is the response time of the all requests. *Variance Response Time*is measure of how individual request are serviced with respect to average response time.

**Disk Scheduling Algorithms:**

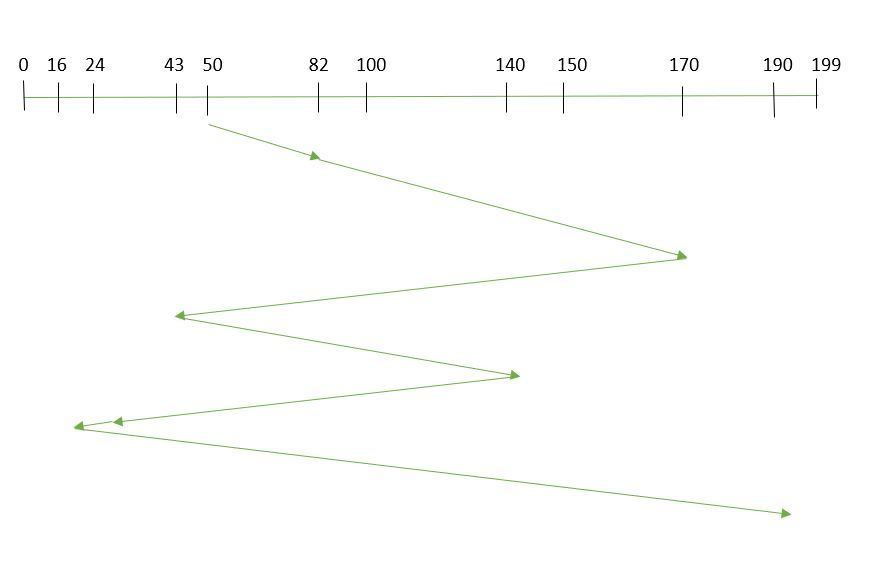
So the disk scheduling algorithm that gives minimum variance response time is better.

**FCFS:**

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

**Example:**

Suppose the order of request is- (82,170,43,140,24,16,190)  
And current position of Read/Write head is: 50



So, total overhead movement  (total distance covered by the disk arm) : =(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

Advantages:

* Every request gets a fair chance
* No indefinite postponement

Disadvantages:

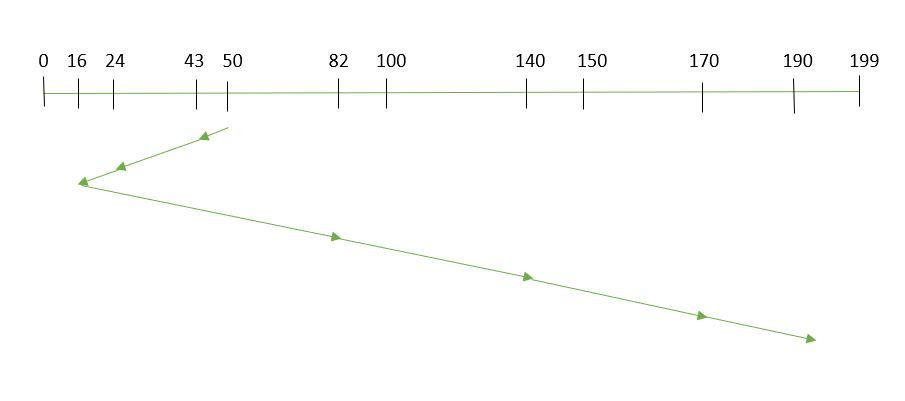
* Does not try to optimize seek time
* May not provide the best possible service

**SSTF:**

In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first.

SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

#### **Example:**

1. Suppose the order of request is- (82,170,43,140,24,16,190)  
   And current position of Read/Write head is : 50 

So,

total overhead movement (total distance covered by the disk arm) =(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

**Advantages:**

* Average Response Time decreases
* Throughput increases

**Disadvantages:**

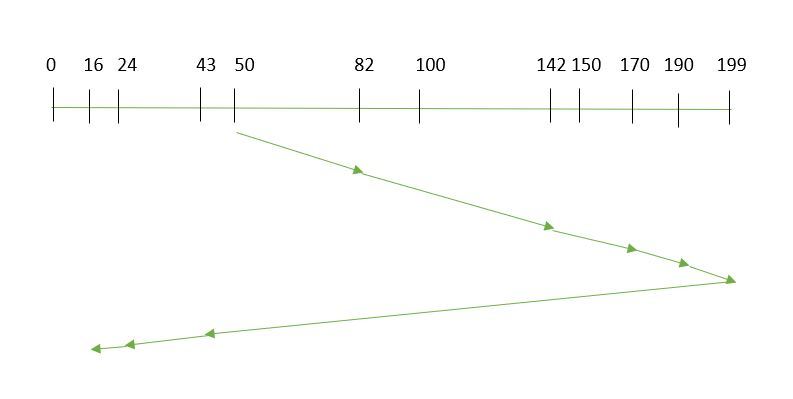
* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
* High variance of response time as SSTF favors only some requests

**SCAN:**

In SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. This algorithm works as an elevator and is hence also known as an **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

**Example:**

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



Therefore, the total overhead movement  (total distance covered by the disk arm)  is calculated as:

1. =(199-50)+(199-16) =332

Advantages:

* High throughput
* Low variance of response time
* Average response time

Disadvantages:

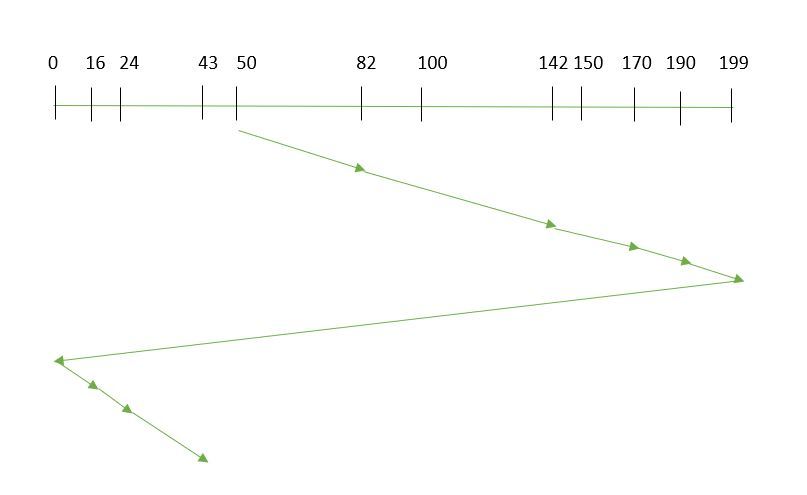
* Long waiting time for requests for locations just visited by disk arm

**C - SCAN**:

These situations are avoided in *CSCAN*algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

#### **Example:**

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move  **“towards the larger value”.**



so, the total overhead movement  (total distance covered by the disk arm) is calculated as:

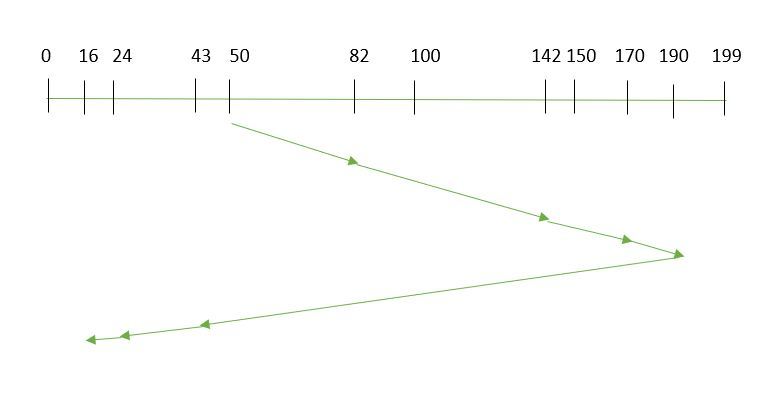
=(199-50)+(199-0)+(43-0) =391

Advantages:

* Provides more uniform wait time compared to SCAN

**LOOK:**  It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.  
 **Example:**

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



So, the total overhead movement  (total distance covered by the disk arm) is calculated as:

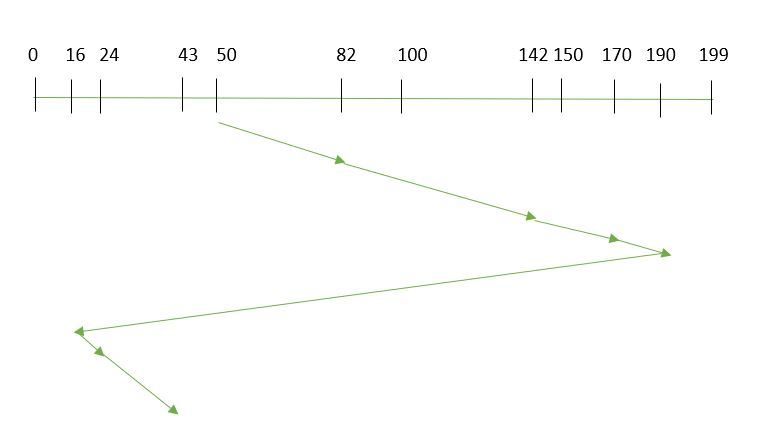
1. =(190-50)+(190-16) =314

**C - LOOK:**

As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end’s last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

#### **Example:**

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”**



So, the total overhead movement  (total distance covered by the disk arm) is calculated as:

1. =(190-50)+(190-16)+(43-16) =341

**RSS**– It stands for random scheduling and just like its name it is nature. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm sits perfectly. Which is why it is usually used for analysis and simulation.

**LIFO**– In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is newest or last entered is serviced first, and then the rest in the same order.

**Advantages**

* Maximizes locality and resource utilization
* Can seem a little unfair to other requests and if new requests keep coming in, it cause starvation to the old and existing ones.

**N-STEP SCAN**– It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N request and this way all get requests to get a guaranteed service

**Advantages**

* It eliminates the starvation of requests completely

**FSCAN**– This algorithm uses two sub-queues. During the scan all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

**Advantages**

* FSCAN along with N-Step-SCAN prevents “arm stickiness” (phenomena in I/O scheduling where the scheduling algorithm continues to service requests at or near the current sector and thus prevents any seeking)
* Each algorithm is unique in its own way. Overall Performance depends on the number and type of requests.

***Note:****Average Rotational latency is generally taken as 1/2(Rotational latency).*

**Shortest Seek Time First (SSTF) –**   
Basic idea is the tracks which are closer to current disk head position should be serviced first in order to *minimise the seek operations*.

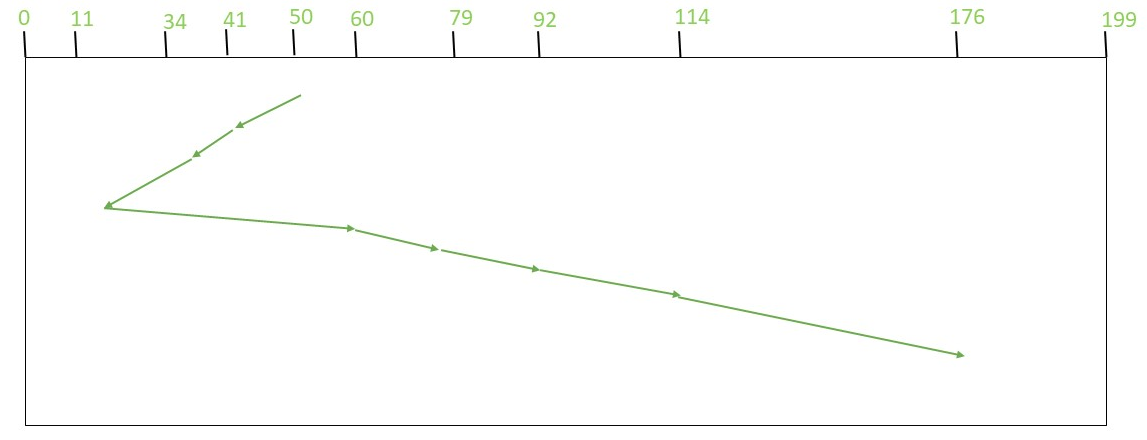
**Advantages:**

* Better performance than FCFS scheduling algorithm.
* It provides better throughput.
* This algorithm is used in Batch Processing system where throughput is more important.
* It has less average response and waiting time.

**Disadvantages:**

* Starvation is possible for some requests as it favours easy to reach request and ignores the far away processes.
* There is lack of predictability because of high variance of response time.
* Switching direction slows things down.

**Example –**   
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}   
Initial head position = 50



Therefore, total seek count is calculated as:

= (50-41)+(41-34)+(34-11)+(60-11)+(79-60)+(92-79)+(114-92)+(176-114)

= 204

**Which can also be directly calculated as: (50-11)+(176-11)**

**SCAN (Elevator) algorithm**   
In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the **elevator algorithm**.

**Advantages:**

* This algorithm is simple and easy to understand.
* SCAN algorithm have no starvation.
* This algorithm is better than FCFS Scheduling algorithm .

**Disadvantages:**

* More complex algorithm to implement.
* This algorithm is not fair because it cause long waiting time for the cylinders just visited by the head.
* It causes the head to move till the end of the disk in this way the requests arriving ahead of the arm position would get immediate service but some other requests that arrive behind the arm position will have to wait for the request to complete.

**Example:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

Direction = left (We are moving from right to left)

**Output:**

Total number of seek operations = 226

Seek Sequence is

41

34

11

0

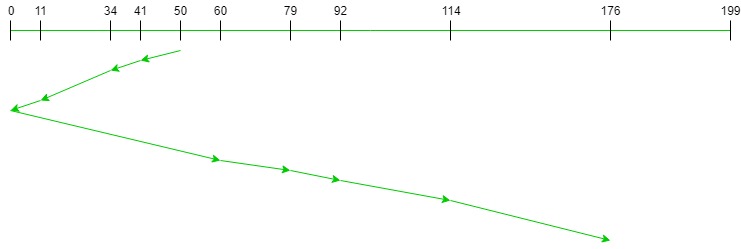
60

79

92

114

176



Therefore, the total seek count is calculated as:

= (50-41)+(41-34)+(34-11)

+(11-0)+(60-0)+(79-60)

+(92-79)+(114-92)+(176-114)

= 226

### ****C-SCAN (Circular Elevator) Disk Scheduling Algorithm****

The circular SCAN (C-SCAN) scheduling algorithm is a modified version of the SCAN disk scheduling algorithm that deals with the inefficiency of the SCAN algorithm by servicing the requests more uniformly.

Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the “Circular Elevator Algorithm” as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

#### **Advantages :**

* Works well with moderate to heavy loads.
* It provides better response time and uniform waiting time.

#### **Disadvantages :**

* May not be fair to service requests for tracks at the extreme end.
* It has more seek movements as compared to the SCAN Algorithm.

**Examples:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

Direction = right(We are moving from left to right)

**Output:**

Initial position of head: 50

Total number of seek operations = 389

Seek Sequence is

60

79

92

114

176

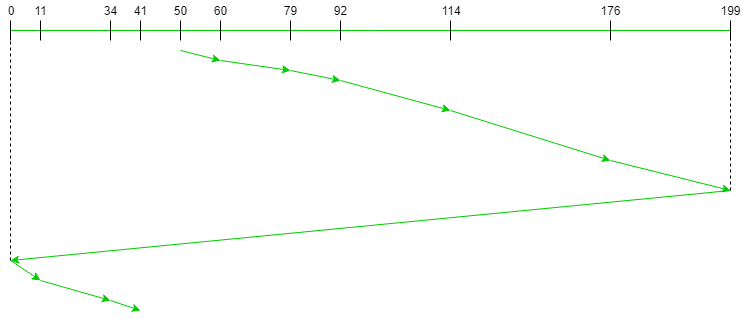
199

0

11

34

41



Therefore, the total seek count is calculated as:   
 = (60-50)+(79-60)+(92-79)+(114-92)+(176-114)+(199-176)+(199-0)+(11-0)+(34-11)+(41-34)

= 389

# LOOK Disk Scheduling Algorithm

The LOOK disk scheduling algorithm is a variation of the SCAN algorithm for disk scheduling. It is used to reduce the amount of time it takes to access data on a hard disk drive by minimizing the seek time between read/write operations.

The LOOK algorithm operates by scanning the disk in a specific direction, but instead of going all the way to the end of the disk before reversing direction like the SCAN algorithm, it reverses direction as soon as it reaches the last request in the current direction.

The **basic steps** involved in the LOOK algorithm:

* Determine the initial direction of disk head movement.
* Sort the pending disk requests in the order in which they will be serviced.
* Scan the disk in the chosen direction, servicing requests as they are encountered.
* When the last request in the current direction has been serviced, reverse direction and continue scanning until all requests have been serviced.

Advantages:

* It can provide better performance than the FCFS (first-come, first-served) and SSTF (shortest-seek-time-first) algorithms because it reduces the number of head movements required to access data on the disk.
* It is relatively simple to implement and does not require a large amount of memory or processing power.
* It is efficient in terms of disk usage because it scans only the areas of the disk where data is located.

Disadvantages :

* It may not be optimal in situations where there are large amounts of data to be read or written in one direction, as it could lead to a large number of requests being queued up in the opposite direction.
* It may not be suitable for real-time systems where fast response times are critical, as it does not prioritize requests based on their urgency or importance.
* It may lead to starvation of requests that are located far away from the current position of the disk head.

**LOOK Disk Scheduling Algorithm:**  
LOOK is the advanced version of [SCAN (elevator) disk scheduling algorithm](https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/) which gives slightly better seek time than any other algorithm in the hierarchy *(FCFS->SRTF->SCAN->C-SCAN->LOOK)*.

The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also “looks” ahead as if there are more tracks that are needed to be serviced in the same direction.

If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction. The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

**Algorithm:**

* Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
* The initial direction in which head is moving is given and it services in the same direction.
* The head services all the requests one by one in the direction head is moving.
* The head continues to move in the same direction until all the request in this direction are finished.
* While moving in this direction calculate the absolute distance of the track from the head.
* Increment the total seek count with this distance.
* Currently serviced track position now becomes the new head position.
* Go to step 5 until we reach at last request in this direction.
* If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

**Examples:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

Direction = right (We are moving from left to right)

**Output:**

Initial position of head: 50

Total number of seek operations = 291

Seek Sequence is

60

79

92

114

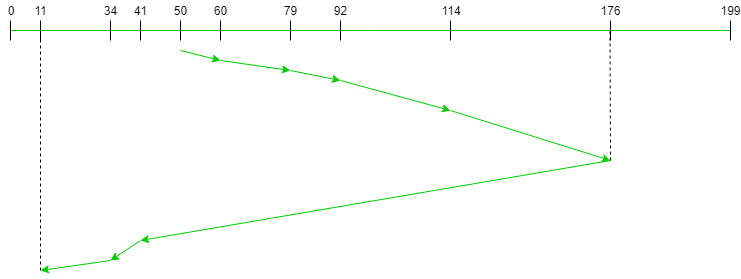
176

41

34

11

The following chart shows the sequence in which requested tracks are serviced using LOOK.



Therefore, the total seek count is calculated as:

= (60-50)+(79-60)+(92-79) +(114-92)+(176-114) +(176-41)+(41-34)+(34-11)

# C-LOOK Disk Scheduling Algorithm

The C-LOOK disk scheduling algorithm is a variation of the LOOK algorithm for disk scheduling. It operates by scanning the disk in one direction only, servicing all requests along the way until the last request in that direction has been serviced, and then immediately returning to the beginning of the disk to repeat the process.

Here are the **basic steps** involved in the C-LOOK algorithm:

* Determine the initial position of the disk head.
* Sort the pending disk requests in the order in which they will be serviced.
* Scan the disk in the chosen direction, servicing requests as they are encountered.
* When the last request in the current direction has been serviced, immediately return to the beginning of the disk and repeat the process.

Advantages:

* It can provide better performance than the LOOK algorithm because it reduces the number of head movements required to access data on the disk.
* It is relatively simple to implement and does not require a large amount of memory or processing power.
* It can be efficient in terms of disk usage because it scans only the areas of the disk where data is located.

Disadvantages :

* It may not be optimal in situations where there are large amounts of data to be read or written in one direction, as it could lead to a large number of requests being queued up in the opposite direction.
* It may not be suitable for real-time systems where fast response times are critical, as it does not prioritize requests based on their urgency or importance.
* It may lead to starvation of requests that are located far away from the current position of the disk head.

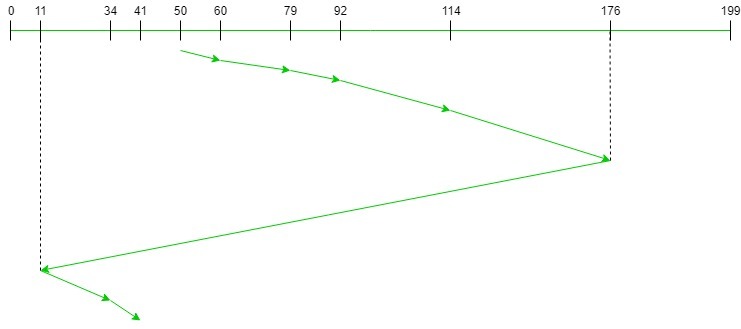
**C-LOOK (Circular LOOK) Disk Scheduling Algorithm:**   
 **C-LOOK** is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for C-LOOK.   
In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

**Algorithm-**

* Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
* The initial direction in which the head is moving is given and it services in the same direction.
* The head services all the requests one by one in the direction it is moving.
* The head continues to move in the same direction until all the requests in this direction have been serviced.
* While moving in this direction, calculate the absolute distance of the tracks from the head.
* Increment the total seek count with this distance.
* Currently serviced track position now becomes the new head position.
* Go to step 5 until we reach the last request in this direction.
* If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
* Reverse the direction and go to step 3 until all the requests have not been serviced.

**Examples:**

***Input:****Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}   
Initial head position = 50   
Direction = right (Moving from left to right)****Output:****Initial position of head: 50   
Total number of seek operations = 321   
Seek Sequence is   
60   
79   
92   
114   
176   
11   
34   
41*The following chart shows the sequence in which requested tracks are serviced using C-LOOK.



Therefore, the total seek count = (60 – 50) + (79 – 60) + (92 – 79) + (114 – 92) + (176 – 114) + (176 – 11) + (34 – 11) + (41 – 34) = 321