**Operating System – Definition:**

An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.

An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, a memory management module, I/O programs, and a file system.

**Functions of Operating system –** Operating system performs three functions: 

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used efficiently.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions at the same time without interfering with service.
4. **Throughput:** An OS should be constructed so that It can give maximum

**throughput** (Number of tasks per unit time).

**Major Functionalities of Operating System:**

* **Resource Management:**When parallel accessing happens in the OS means when multiple users are accessing the system the OS works as Resource Manager, its responsibility is to provide hardware to the user. It decreases the load in the system.
* **Process Management:**It includes various tasks like **scheduling**, **termination**of the process. OS manages various tasks at a time. Here **CPU Scheduling**happens means all the tasks would be done by the many algorithms that use for scheduling.
* **Storage Management:**The **file system** mechanism used for the management of the storage. **NIFS**, **CFS**, **CIFS**, **NFS**, etc. are some file systems. All the data stores in various tracks of Hard disks that all managed by the storage manager. It included **Hard Disk**.
* **Memory Management:**Refers to the management of primary memory. The operating system has to keep track, how much memory has been used and by whom. It has to decide which process needs memory space and how much. OS also has to allocate and deallocate the memory space.
* **Security/Privacy Management:**Privacy is also provided by the Operating system by means of passwords so that unauthorized applications can’t access programs or data. For example, Windows uses **Kerberos** authentication to prevent unauthorized access to data.

[**Types of Operating System**](https://www.geeksforgeeks.org/operating-system-types-operating-systems-awaiting-author/)

* Batch Operating System- Sequence of jobs in a program on a computer without manual interventions.
* Time-sharing operating System- allows many users to share the computer resources. (Max utilization of the resources).
* Distributed operating System- Manages a group of different computers and makes appear to be a single computer.
* Network operating system- computers running in different operating systems can participate in a common network (It is used for security purposes).
* Real-time operating system – meant applications to fix the deadlines.

**Examples of Operating System are –**

* Windows (GUI based, PC)
* GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
* macOS (Macintosh), used for Apple’s personal computers and workstations (MacBook, iMac).
* Android (Google’s Operating System for smartphones/tablets/smartwatches)
* iOS (Apple’s OS for iPhone, iPad, and iPod Touch)

**File Management –**   
A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files. An Operating System carries out the following file management activities. It keeps track of where information is stored, user access settings and status of every file, and more… These facilities are collectively known as the file system.

Moreover, Operating System also provides certain services to the computer system in one form or the other.   
The Operating System provides certain services to the users which can be listed in the following manner:

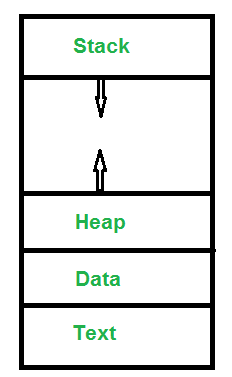
1. **Program Execution**: The Operating System is responsible for the execution of all types of programs whether it be user programs or system programs. The Operating System utilizes various resources available for the efficient running of all types of functionalities.
2. **Handling Input/output Operations**: The Operating System is responsible for handling all sorts of inputs, i.e., from the keyboard, mouse, desktop, etc. The Operating System does all interfacing in the most appropriate manner regarding all kinds of Inputs and Outputs.   
   For example, there is a difference in the nature of all types of peripheral devices such as mice or keyboards, the Operating System is responsible for handling data between them.
3. **Manipulation of File System**: The Operating System is responsible for making decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc. The Operating System decides how the data should be manipulated and stored.
4. **Error Detection and Handling**: The Operating System is responsible for the detection of any type of error or bugs that can occur while any task. The well-secured OS sometimes also acts as a countermeasure for preventing any sort of breach to the Computer System from any external source and probably handling them.
5. **Resource Allocation:** The Operating System ensures the proper use of all the resources available by deciding which resource to be used by whom for how much time. All the decisions are taken by the Operating System.
6. **Accounting:** The Operating System tracks an account of all the functionalities taking place in the computer system at a time. All the details such as the types of errors that occurred are recorded by the Operating System.
7. **Information and Resource Protection:** The Operating System is responsible for using all the information and resources available on the machine in the most protected way. The Operating System must foil an attempt from any external resource to hamper any sort of data or information.

[**Virtual Memory**](https://www.geeksforgeeks.org/virtual-memory-operating-systems/)**:**   
This is a technique used by the operating systems which allows the user can load programs that are larger than the main memory of the computer. In this technique, the program is executed even if the complete program cannot be loaded inside the main memory leading to efficient memory utilization.

**Introduction of Process Management**

**Program vs Process**   
A process is a program in execution. For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process.

**What does a process look like in memory?**



***Text Section****:* A Process, sometimes known as the Text Section, also includes the current activity represented by the value of the ***Program Counter***.   
***Stack****:* The stack contains temporary data, such as function parameters, returns addresses, and local variables.   
***Data Section****:* Contains the global variable.   
***Heap Section****:* Dynamically allocated memory to process during its run time.

**States of Process:**   
A process is in one of the following states:

**1. New:** Newly Created Process (or) being-created process.

**2. Ready:** After creation process moves to Ready state, i.e. the

process is ready for execution.

**3. Run:** Currently running process in CPU (only one process at

a time can be under execution in a single processor).

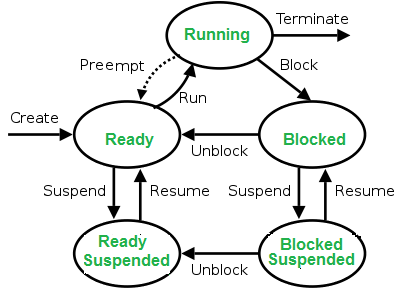
**4. Wait (or Block):** When a process requests I/O access.

**5. Complete (or Terminated):** The process completed its execution.

**6. Suspended Ready:** When the ready queue becomes full, some processes

are moved to suspended ready state

**7. Suspended Block:** When waiting queue becomes full.



**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?**   
1. When a high-priority process comes to a ready state (i.e. with higher priority than the running process)   
2. An Interrupt occurs   
3. User and kernel-mode switch (It is not necessary though)   
4. Pre-emptive 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.

**Types of schedulers:**

1. **Long term – performance –** Makes a decision about 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 long term scheduler.
2. **Short term – Context switching time –** Short term scheduler will decide which process to be executed next and then it will call dispatcher. A dispatcher is a software that moves process from ready to run and vice versa. In other words, it is context switching.
3. **Medium term – Swapping time –** Suspension decision is taken by medium term scheduler. Medium term scheduler is used for swapping that is moving the process from main memory to secondary and vice versa.

**Multiprogramming –** We have many processes ready to run. There are two types of multiprogramming:

1. **Pre-emption –** Process is forcefully removed from CPU. Pre-emption is also called as time sharing or multitasking.
2. **Non pre-emption –** Processes are not removed until they complete the execution.

**Process Table and Process Control**

While creating a process the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process. As the operating system 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 these 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 PCB’s, that means logically contains a PCB for all of the current processes in the system.

# Thread in Operating System

A thread is a path of execution within a process. A process can contain multiple threads.

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. More advantages of multithreading are discussed below

**Advantages of Thread over Process**  
1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

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

3. 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.

4. 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.

5. 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.

6. 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**  
There are two types of threads.  
User Level Thread  
Kernel Level Thread

## **Difference between Process and Thread:**

|  | **Process** | **Thread** |
| --- | --- | --- |
| 1. | Process means any program is in execution. | Thread means segment of a process. |
| 2. | Process takes more time to terminate. | Thread takes less time to terminate. |
| 3. | It takes more time for creation. | It takes less time for creation. |
| 4. | It also takes more time for context switching. | It takes less time for context switching. |
| 5. | Process is less efficient in term of communication. | Thread is more efficient in term of communication. |
| 6. | Multi programming holds the concepts of multi process. | We don’t need multi programs in action for multiple threads because a single process consists of multiple threads. |
| 7. | Process is isolated. | Threads share memory. |
| 8. | Process is called heavy weight process. | A Thread is lightweight as each thread in a process shares code, data and resources. |
| 9. | Process switching uses interface in operating system. | Thread switching does not require to call a operating system and cause an interrupt to the kernel. |
| 10. | If one process is blocked then it will not affect the execution of other process | Second thread in the same task could not run, while one server thread is blocked. |
| 11. | Process has its own Process Control Block, Stack and Address Space. | Thread has Parents’ PCB, its own Thread Control Block and Stack and common Address space. |
| 12. | If one process is blocked, then no other process can execute until the first process is unblocked. | While one thread is blocked and waiting, a second thread in the same task can run. |
| 13. | Changes to the parent process does not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behaviour of the other threads of the process. |

# CPU Scheduling in Operating Systems

**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

**Why do we need scheduling?**  
A typical process involves both I/O time and CPU time. In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

| Parameter | **PREEMPTIVE SCHEDULING** | **NON-PREEMPTIVE SCHEDULING** |
| --- | --- | --- |
| Basic | In this, resources (CPU Cycle) are allocated to a process for a limited time. | Once resources (CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state. |
| Interrupt | Process can be interrupted in between. | Process cannot be interrupted until it terminates itself or its time is up. |
| Starvation | If a process having high priority frequently arrives in the ready queue, a low priority process may starve. | If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve. |
| Overhead | It has overheads of scheduling the processes. | It does not have overheads. |
| Flexibility | flexible | rigid |
| Cost | cost associated | no cost associated |
| CPU Utilization | In preemptive scheduling, CPU utilization is high. | It is low in non preemptive scheduling. |
| Examples | Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First. | Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First. |

# FCFS CPU Scheduling

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.   
First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.   
In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

# Shortest Job First (or SJF) CPU Scheduling

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

* Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
* It is a Greedy Algorithm.

**Algorithm:**

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.

3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

### **Shortest Remaining Time First (SRTF) scheduling**

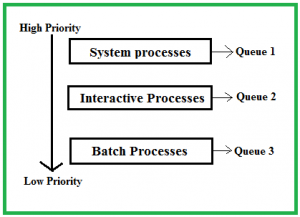
### In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

**Advantage:**   
1- Short processes are handled very quickly.   
2- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.   
3- When a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.  
**Disadvantage:**   
1- Like shortest job first, it has the potential for process starvation.   
2- Long processes may be held off indefinitely if short processes are continually added.

# Multilevel Queue (MLQ) CPU Scheduling

The ready queue can be divided into different classes where each class has its own scheduling needs.

**Ready Queue** is divided into separate queues for each class of processes. For example, let us take three different types of processes System processes, Interactive processes, and Batch Processes. All three processes have their own queue.



All three different type of processes has their own queue. Each queue has its own Scheduling algorithm. For example, queue 1 and queue 2 uses **Round Robin** while queue 3 can use **FCFS** to schedule their processes.

**Advantages:**

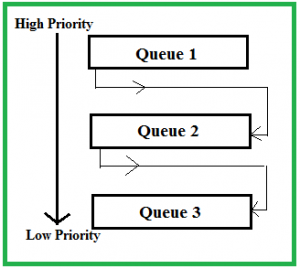
* The processes are permanently assigned to the queue, so it has advantage of low scheduling overhead.

**Disadvantages:**

* Some processes may starve for CPU if some higher priority queues are never becoming empty.
* It is inflexible in nature.

# Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling

This Scheduling is like Multilevel Queue(MLQ) Scheduling but in this process can move between the queues. **Multilevel Feedback Queue Scheduling (MLFQ)** keeps analyzing the behavior (time of execution) of processes and according to which it changes its priority.



**Advantages:**

1. It is more flexible.
2. It allows different processes to move between different queues.
3. It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.

**Disadvantages:**

1. For the selection of the best scheduler, it requires some other means to select the values.
2. It produces more CPU overheads.
3. It is the most complex algorithm.

# Introduction of Process Synchronization

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

* **Independent Process** : Execution of one process does not affects the execution of other processes.
* **Cooperative Process** : Execution of one process affects the execution of other processes.

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

**Race Condition**  
When more than one processes are executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.

**Critical Section Problem**

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.

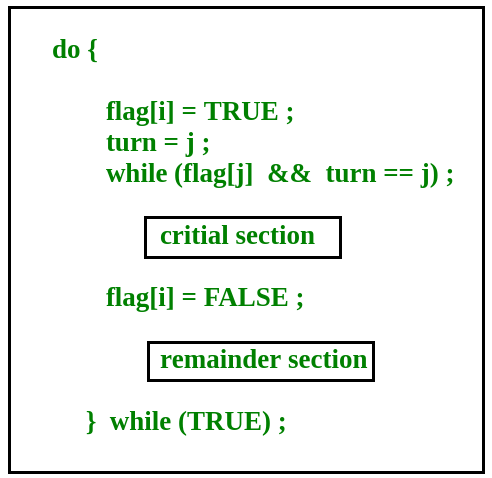
Any solution to the critical section problem must satisfy three requirements:

* **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 cannot 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.



Peterson’s Solution preserves all three conditions:

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

Disadvantages of Peterson’s Solution

* + It involves Busy waiting
  + It is limited to 2 processes.

**TestAndSet**  
TestAndSet is a hardware solution to the synchronization problem. In TestAndSet, we have a shared lock variable which can take either of the two values, 0 or 1.

0 Unlock

1 Lock

Before entering into the critical section, a process inquiry about the lock. If it is locked, it keeps on waiting until it becomes free and if it is not locked, it takes the lock and executes the critical section.

In TestAndSet, Mutual exclusion and progress are preserved but bounded waiting cannot be preserved.

**Semaphores**

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().

There are two types of semaphores:

**1. Binary Semaphores**

**2. 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.

counting semaphore is a semaphore that has multiple values of the counter. The value can range over an unrestricted domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses.

**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.

# Deadlock in Operating System

A process in operating system uses resources in the following way.   
1) Requests a resource   
2) Use the resource   
3) Releases the resource

**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.

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

**Methods for handling deadlock**

1. Deadlock prevention or avoidance: The idea is to not let the system into a deadlock state.

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

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

4. Deadlock Avoidance: Banker’s algorithm

Banker’s algorithm:

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

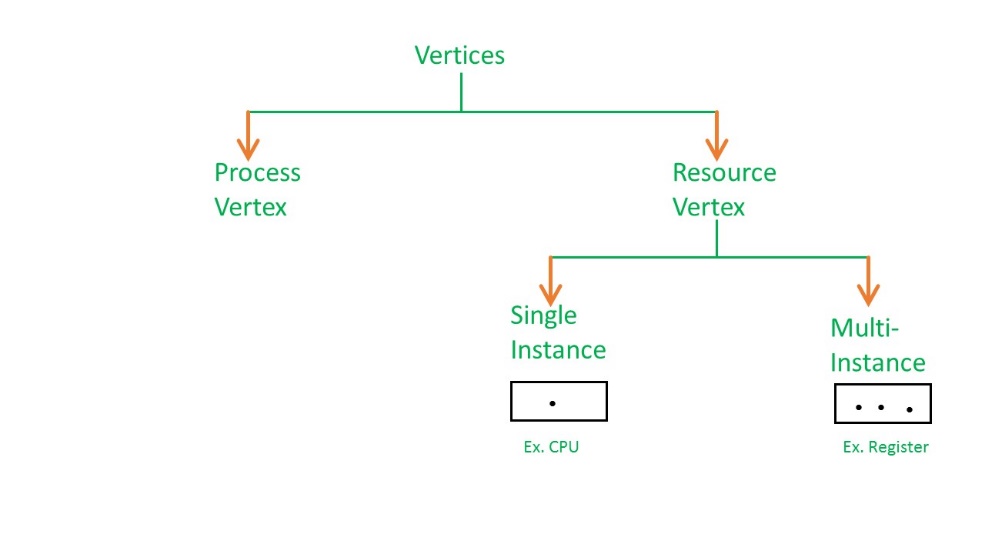
**Resource Allocation Graph (RAG)**

Allocation, request, available of resource all that thing to understand what is the state of the system. To understand the state of the system we represent the same information in the graph. That graph is called **Resource Allocation Graph (RAG).**

RAG also contains vertices and edges. In RAG vertices are two type –

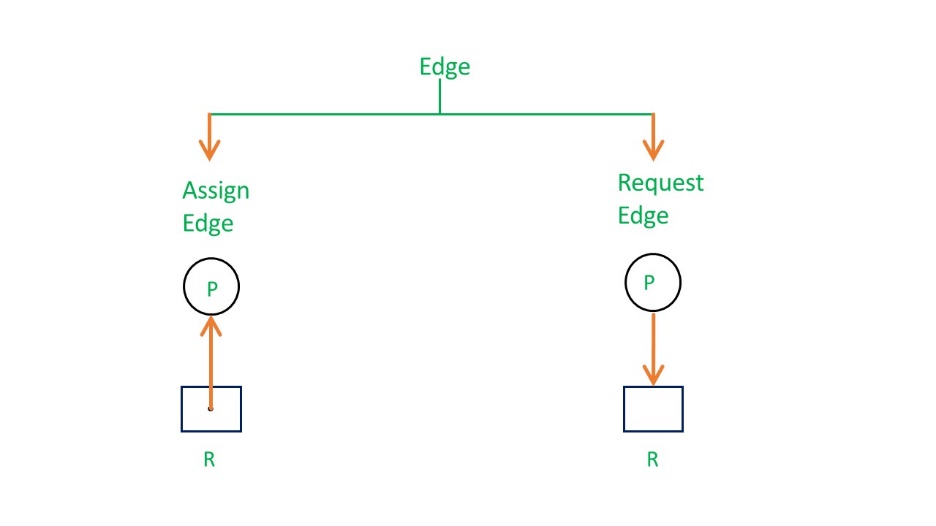
**1. Process vertex –** Every process will be represented as a process vertex. Generally, the process will be represented with a circle.  
**2. Resource vertex –** Every resource will be represented as a resource vertex. It is also two type –

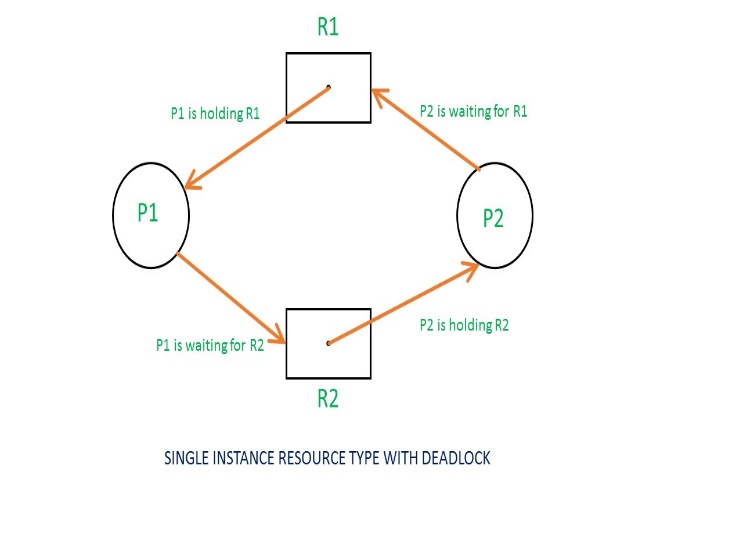
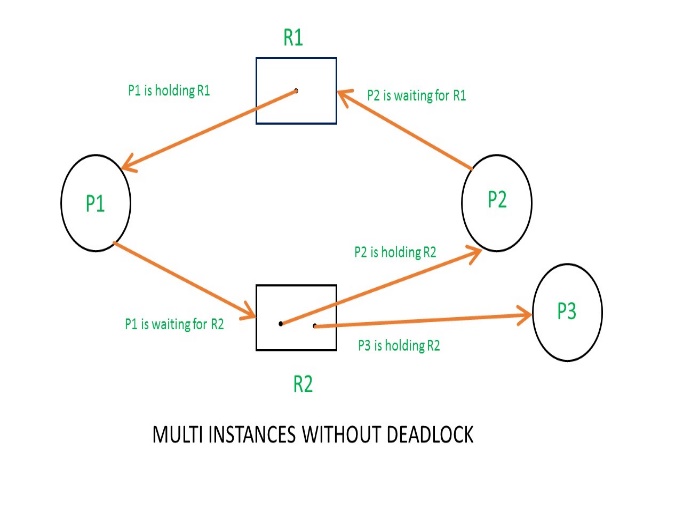
* **Single instance type resource –** It represents as a box, inside the box, there will be one dot. So the number of dots indicate how many instances are present of each resource type.
* **Multi-resource instance type resource –** It also represents as a box, inside the box, there will be many dots present.

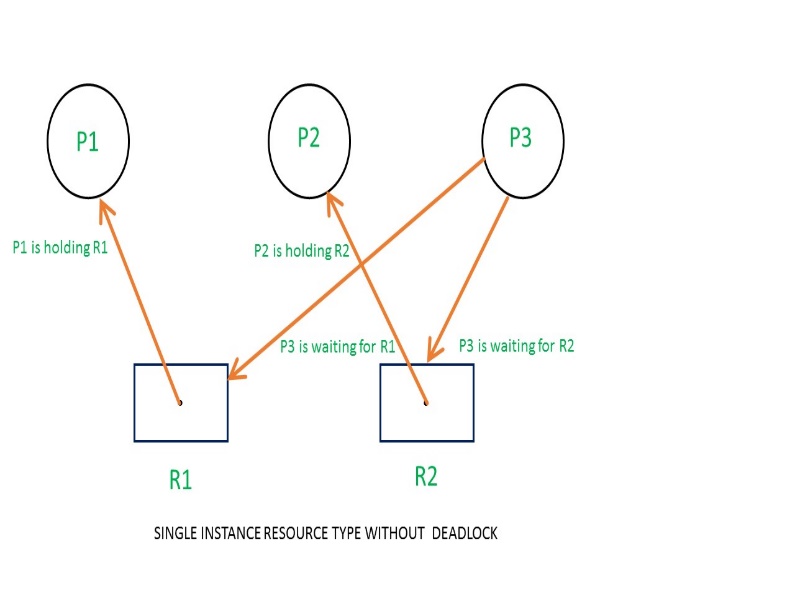


There are two types of edges in RAG –

**1. Assign Edge –** If you already assign a resource to a process then it is called Assign edge.  
**2. Request Edge –** It means in future the process might want some resource to complete the execution, that is called request edge.



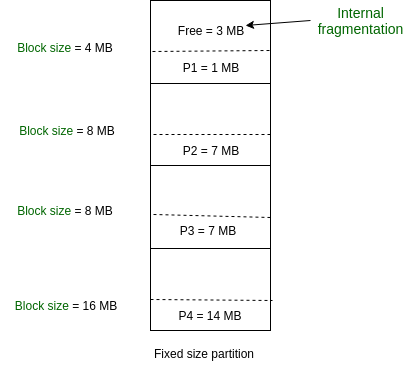




**Memory Management**

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:

1. Fixed (or static) partitioning
2. 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.

**Advantages of Fixed Partitioning –**

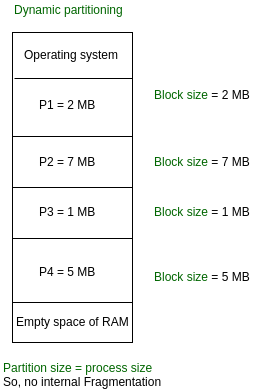
1. **Easy to implement:**   
   Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into a certain partition without focusing on the emergence of Internal and External Fragmentation.
2. **Little OS overhead:**   
   Processing of Fixed Partitioning requires lesser excess and indirect computational power.

**Disadvantages of Fixed Partitioning –**

1. **Internal Fragmentation:**   
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.
2. **External Fragmentation:**   
   The total unused space 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).
3. **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.
4. **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. 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 avoid 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 –**

1. **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.
2. **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.
3. **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 cannot 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 of Variable Partitioning –**

1. **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.
2. **External Fragmentation:**  
   There will be external fragmentation inspite of absence of internal fragmentation.

For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let’s suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.

# Paging in Operating System

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU

- Physical Address (represented in bits): An address actually available on memory unit

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

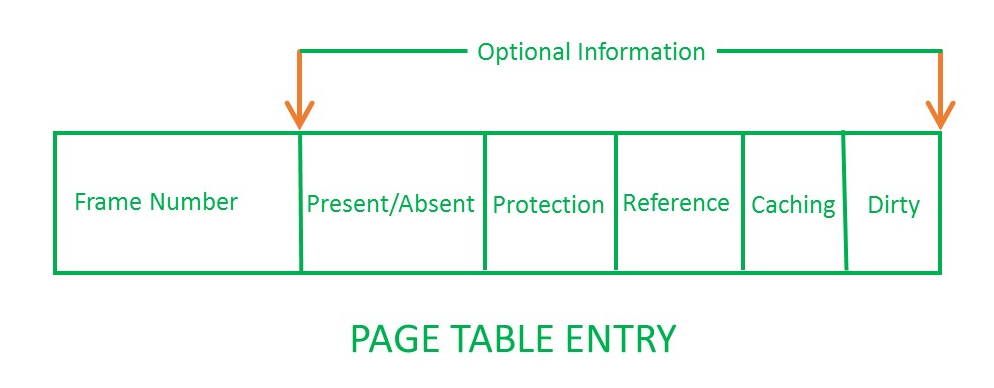
Address generated by CPU is divided into

* **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.

Physical Address is divided into

* **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
* **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

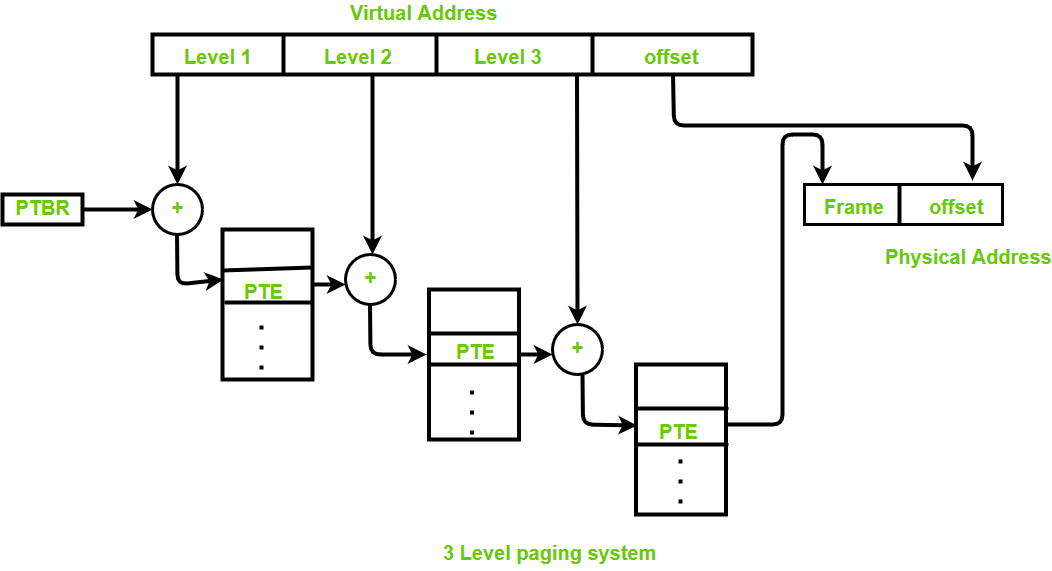
# Page Table Entries in Page Table



1. **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.
2. Number of bits for frame = Size of physical memory/frame size
3. **Present/Absent bit –** Present or absent bit says whether a particular page you are looking for is present or absent. In case if it is not present, that is called Page Fault. It is set to 0 if the corresponding page is not in memory. Used to control page fault by the operating system to support virtual memory. Sometimes this bit is also known as **valid/invalid** bits.
4. **Protection bit –** Protection bit says that what kind of protection you want on that page. So, these bit for the protection of the page frame (read, write etc).
5. **Referenced bit –** Referenced bit will say whether this page has been referred in the last clock cycle or not. It is set to 1 by hardware when the page is accessed.
6. **Caching enabled/disabled –** Sometimes we need the fresh data. Let us say the user is typing some information from the keyboard and your program should run according to the input given by the user. In that case, the information will come into the main memory. Therefore, main memory contains the latest information which is typed by the user. Now if you try to put that page in the cache, that cache will show the old information. So, whenever freshness is required, we don’t want to go for caching or many levels of the memory. The information present in the closest level to the CPU and the information present in the closest level to the user might be different. So we want the information has to be consistency, which means whatever information user has given, CPU should be able to see it as first as possible. That is the reason we want to disable caching. So, this bit **enables or disable** caching of the page.
7. **Modified bit –** Modified bit says whether the page has been modified or not. Modified means sometimes you might try to write something on to the page. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk or it has to be written back or it has to be saved back. It is set to 1 by hardware on write-access to page which is used to avoid writing when swapped out. Sometimes this modified bit is also called as the **Dirty bit**.

# Multilevel Paging in Operating System

**Multilevel Paging** is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table are stores actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).



**Disadvantage:**   
Extra memory references to access address translation tables can slow programs down by a factor of two or more.

# Inverted Page Table in Operating System

Most of the Operating Systems implement a separate page table for each process, sometimes when a process is very large in size and it occupies virtual memory then with the size of the process, it’s page table size also increases substantially.

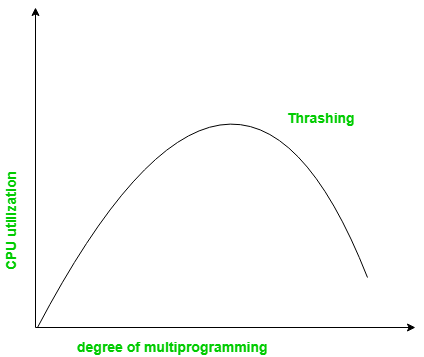
An alternate approach is to use the **Inverted Page Table** structure that consists of one-page table entry for every frame of the main memory. So, the number of page table entries in the Inverted Page Table reduces to the number of frames in physical memory and a single page table is used to represent the paging information of all the processes.

**Advantages and Disadvantages:**

* **Reduced memory space –**  
  Inverted Page tables typically reduces the amount of memory required to store the page tables to a size bound of physical memory. The maximum number of entries could be the number of page frames in the physical memory.
* **Longer lookup time –**  
  Inverted Page tables are sorted in order of frame number but the memory look-up takes place with respect to the virtual address, so, it usually takes a longer time to find the appropriate entry but often these page tables are implemented using hash data structures for a faster lookup.

# Thrashing

**Thrashing** is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



### **Causes of Thrashing**

- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.

- CPU utilization is plotted against the degree of multiprogramming.

- As the degree of multiprogramming increases, CPU utilization also increases.

- If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.

### **How to Eliminate Thrashing**

**Adjust the swap file size:** If the system swap file is not configured correctly, disk thrashing can also happen to you.

**Increase the amount of RAM:** As insufficient memory can cause disk thrashing; one solution is to add more RAM to the laptop. With more memory, your computer can handle tasks easily and don't have to work excessively. Generally, it is the best long-term solution.

**Decrease the number of applications running on the computer:** If there are too many applications running in the background, your system resource will consume a lot. And the remaining system resource is slow that can result in thrashing. So while closing, some applications will release some resources so that you can avoid thrashing to some extent.

**Replace programs:** Replace those programs that are heavy memory occupied with equivalents that use less memory.

# Segmentation in Operating System

In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

There are types of segmentation:

1. **Virtual memory segmentation –**  
   Each process is divided into a number of segments, not all of which are resident at any one point in time.
2. **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. A table stores the information about all such segments and is called Segment Table.

**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.

**Advantages of Segmentation –**

* No Internal fragmentation.
* Segment Table consumes less space in comparison to Page table in paging.

**Disadvantage of Segmentation –**

* As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

# Overlays in Memory Management

The main problem in Fixed partitioning is the size of a process has to be limited by the maximum size of the partition, which means a process can never be span over another. In order to solve this problem, we used some solution which is called as Overlays.

The concept of **overlays** is that whenever a process is running it will not use the complete program at the same time, it will use only some part of it. Then overlays concept says that whatever part you required, you load it and once the part is done, then you just unload it, means just pull it back and get the new part you required and run it.

**Advantage –**

* Reduce memory requirement.
* Reduce time requirement.

**Disadvantage –**

* Overlap map must be specified by programmer.
* Programmer must know memory requirement.
* Overlapped module must be completely disjoint.
* Programming design of overlays structure is complex and not possible in all cases.

# Virtual Memory in Operating System

Virtual memory is a feature of an operating system that **enables a computer to be able to compensate shortages of physical memory by transferring pages of data from random access memory to disk storage**. This process is done temporarily and is designed to work as a combination of RAM and space on the hard disk.

**Advantages:**

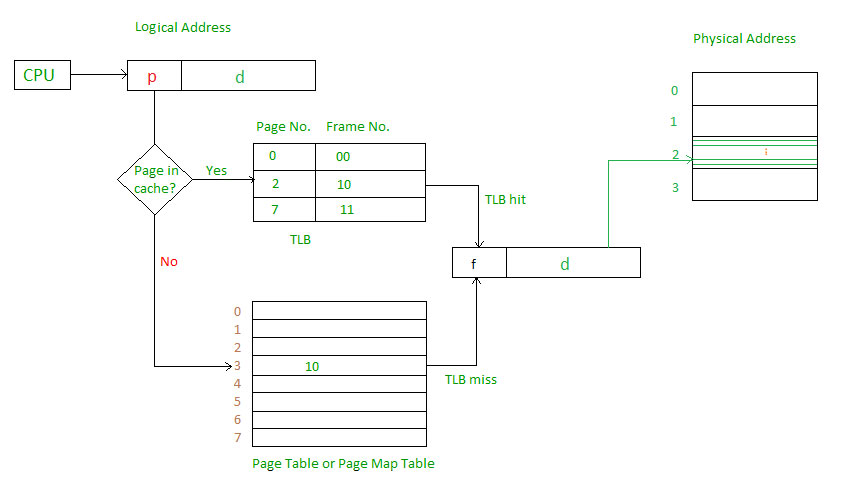
* More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes 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.

# Translation Lookaside Buffer (TLB) in Paging

In Operating System for each process page table will be created, which will contain [Page Table Entry (PTE)](https://www.geeksforgeeks.org/operating-system-page-table-entries/). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing.

Now the question is where to place the page table, such that overall access time (or reference time) will be less.

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table.



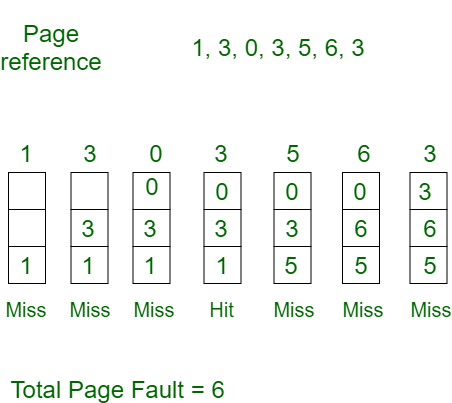
**Effective memory access time (EMAT):** TLB is used to reduce effective memory access time as it is a high speed associative cache.   
**EMAT = h\*(c+m) + (1-h) \*(c+2m)**  
where, h = hit ratio of TLB   
m = Memory access time   
c = TLB access time

# Page Replacement Algorithms in Operating Systems

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

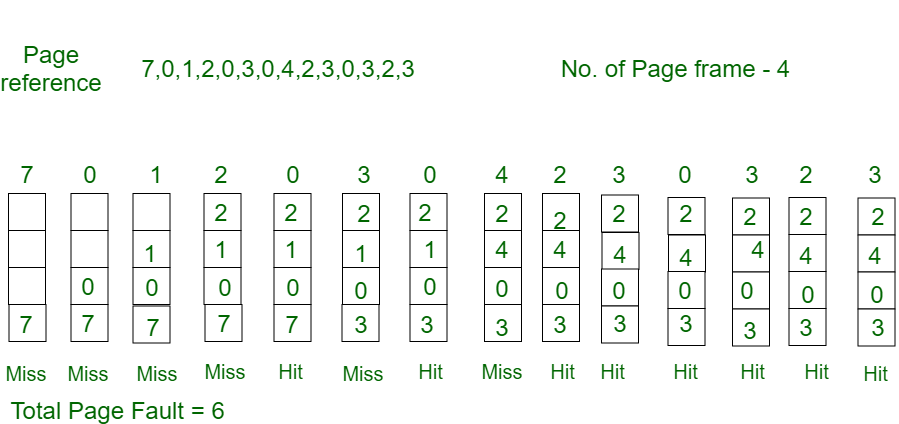
**Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

**Page Replacement Algorithms:**

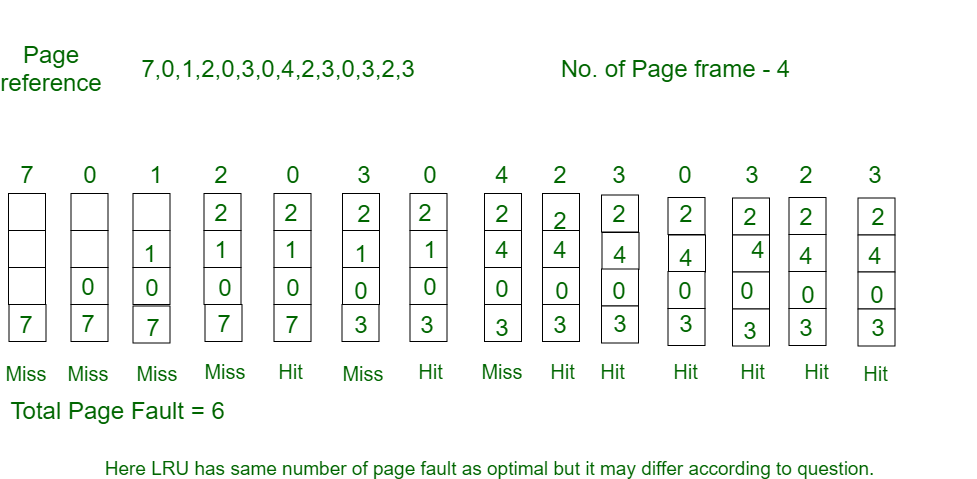
**1. First in First Out (FIFO) –**   
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.   
**Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 pages frames. Find number of page faults.

[**Belady’s anomaly**](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/)**–** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.  For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

**2. Optimal Page replacement –**   
In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.   
**Example-2:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



**3. Least Recently Used –**   
In this algorithm page will be replaced which is least recently used.   
**Example-3** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.



**Disk Management**

**Disk scheduling**is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

* **Seek Time:** 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.
* **Transfer Time:** 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 is:

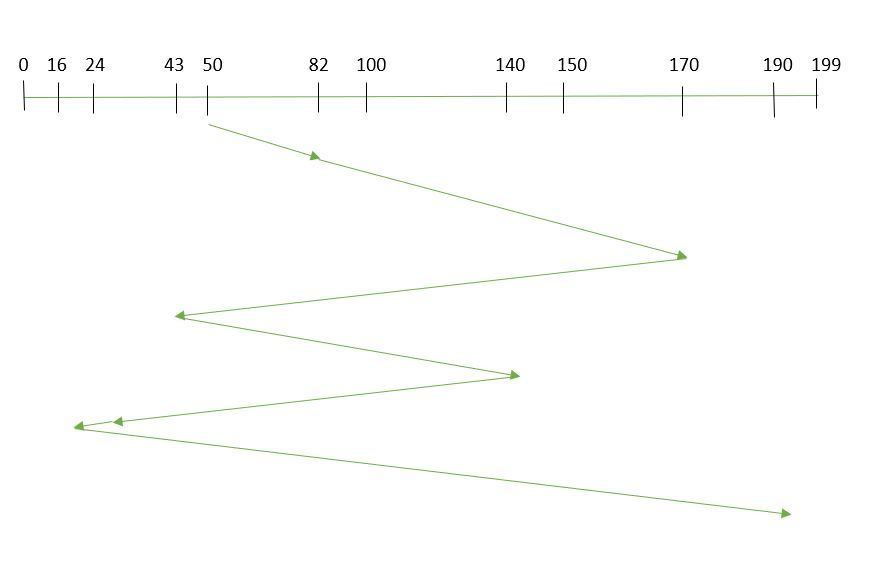
Disk Access Time = Seek Time + Rotational Latency + Transfer Time

**Disk Scheduling Algorithms**

**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 seek time:   
= (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

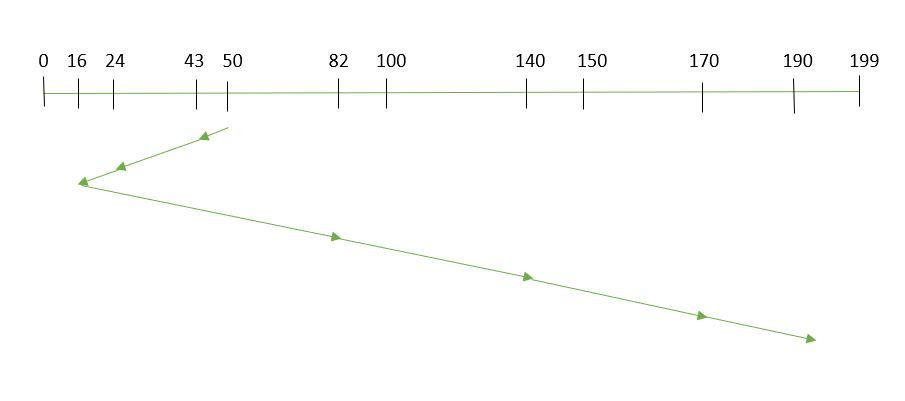
Disadvantages: 

* 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

total seek time:

= (50-43) +(43-24) +(24-16) +(82-16) +(140-82) +(170-140) +(190-170)   
= 208

Advantages: 

* Average Response Time decreases
* Throughput increases

Disadvantages: 

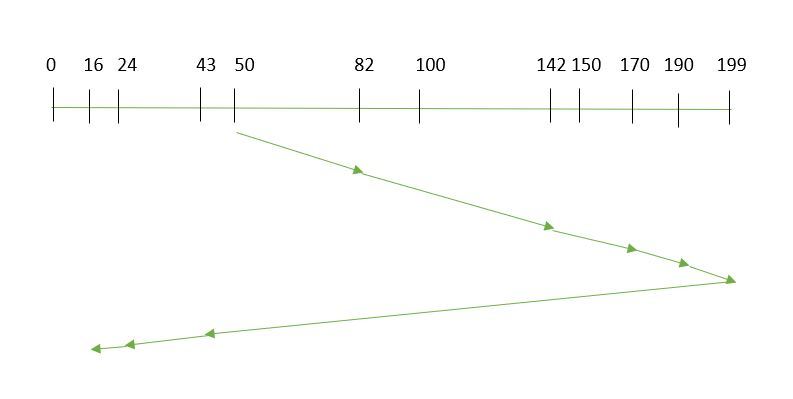
* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has higher seek time as compared to incoming requests
* High variance of response time as SSTF favours only some requests

**SCAN:**In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **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 seek time is calculated as:

So, total seek time: 

= (199-50) + (199-16)   
= 332

Advantages: 

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

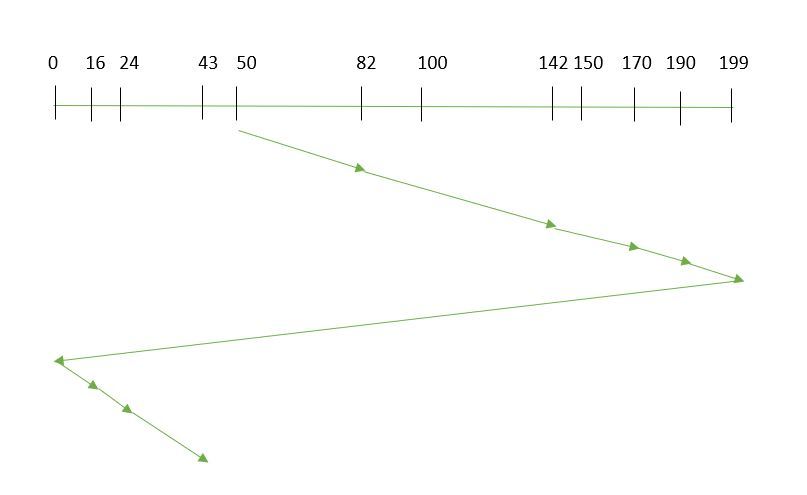
Disadvantages: 

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

**CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

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”.** 

Seek time is calculated as:

=(199-50)+(199-0)+(43-0)   
=391

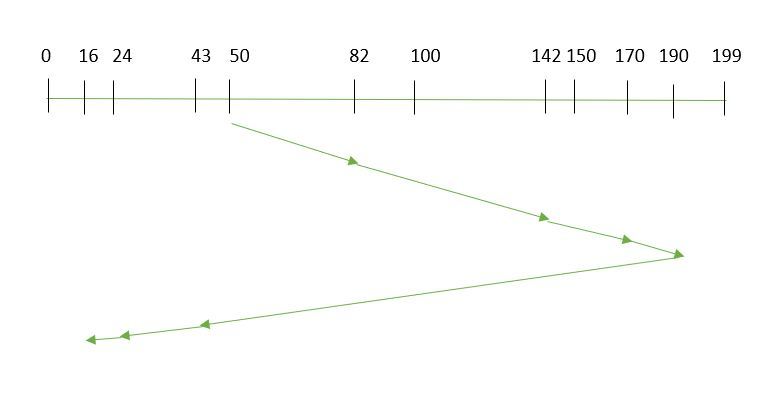
Advantages: 

* Provides more uniform wait time compared to SCAN

1. **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”.**





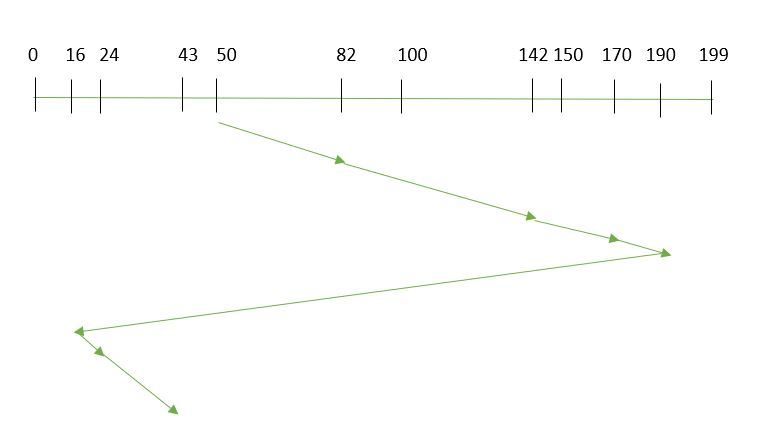
So, the seek time is calculated as:

1. =(190-50)+(190-16)   
   =314

1. **CLOOK:** 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 seek time is calculated as:

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