# Introduction of Operating System

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs conveniently and efficiently.

An operating system is a software that manages computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

**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 four 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.

# Operating System - Services

An Operating System provides services to both the users and to the programs.

* It provides programs an environment to execute.
* It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

* Program execution
* I/O operations
* File System manipulation
* Communication
* Error Detection
* Resource Allocation
* Protection

## **Program execution**

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

* Loads a program into memory.
* Executes the program.
* Handles program's execution.
* Provides a mechanism for process synchronization.
* Provides a mechanism for process communication.
* Provides a mechanism for deadlock handling.

## **I/O Operation**

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

* I/O operation means read or write operation with any file or any specific I/O device.
* Operating system provides the access to the required I/O device when required.

## **File system manipulation**

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

* Program needs to read a file or write a file.
* The operating system gives the permission to the program for operation on file.
* Permission varies from read-only, read-write, denied and so on.
* Operating System provides an interface to the user to create/delete files.
* Operating System provides an interface to the user to create/delete directories.
* Operating System provides an interface to create the backup of file system.

## **Communication**

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

* Two processes often require data to be transferred between them
* Both the processes can be on one computer or on different computers, but are connected through a computer network.
* Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

## **Error handling**

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

* The OS constantly checks for possible errors.
* The OS takes an appropriate action to ensure correct and consistent computing.

## **Resource Management**

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

* The OS manages all kinds of resources using schedulers.
* CPU scheduling algorithms are used for better utilization of CPU.

## **Protection**

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

* The OS ensures that all access to system resources is controlled.
* The OS ensures that external I/O devices are protected from invalid access attempts.
* The OS provides authentication features for each user by means of passwords.

## **Interrupt handlers**

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a call back function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

The process operating system as User Interface:

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral devices, and storage devices. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

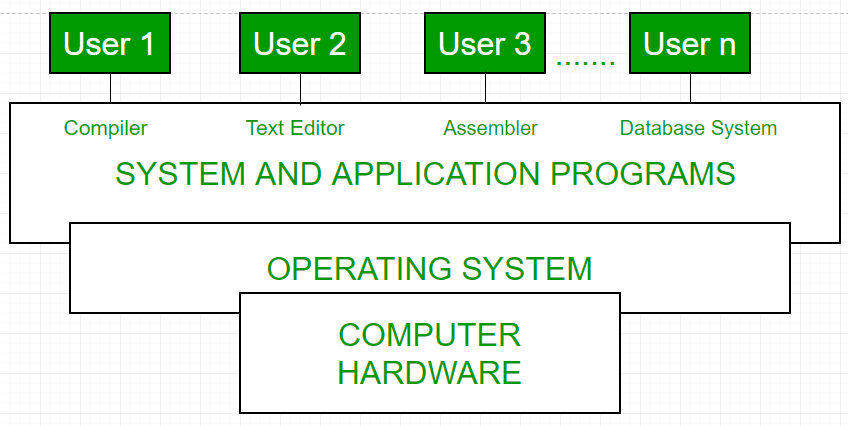


Fig1: Conceptual view of a computer system

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen, and controlling peripheral devices.   
OS is designed to serve two basic purposes: 

1. It controls the allocation and use of the computing System’s resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes it feasible for coding, creation, debugging of application programs.

The Operating system must support the following tasks. The tasks are:  

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high-level language to machine language.
3. Provide a loader program to move the compiled program code to the computer’s memory for execution.
4. Provide routines that handle the details of I/O programming.

**I/O System Management –**   
The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that resides in a separate process associated with that device.   
The I/O subsystem consists of 

* A memory Management component that includes buffering caching and spooling.
* A general device driver interface.

Drivers for specific hardware devices.

**Assembler –**   
The input to an assembler is an assembly language program. The output is an object program plus information that enables the loader to prepare the object program for execution. At one time, the computer programmer had at his disposal a basic machine that interpreted, through hardware, certain fundamental instructions. He would program this computer by writing a series of ones and Zeros (Machine language), place them into the memory of the machine.

**Compiler –**   
The High-level languages- examples are FORTRAN, COBOL, ALGOL, and PL/I are processed by compilers and interpreters. A compiler is a program that accepts a source program in a “high-level language “and produces a corresponding object program. An interpreter is a program that appears to execute a source program as if it was machine language. The same name (FORTRAN, COBOL, etc.) is often used to designate both a compiler and its associated language.

**Loader –**   
A Loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating, and direct-linking. In general, the loader must load, relocate and link the object program. The loader is a program that places programs into memory and prepares them for execution. In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary device and a loader places it in the core. The loader places into memory the machine language version of the user’s program and transfers control to it. Since the loader program is much smaller than the assembler, those make more core available to the user’s program.

**History of Operating system –**   
The operating system has been evolving through the years. The following table shows the history of OS. 

|  |  |  |  |
| --- | --- | --- | --- |
| Generation | Year | Electronic device used | Types of OS Device |
| First | 1945-55 | Vacuum Tubes | Plug Boards |
| Second | 1955-65 | Transistors | Batch Systems |
| Third | 1965-80 | Integrated Circuits(IC) | Multiprogramming |
| Fourth | Since 1980 | Large Scale Integration | PC |
|  |  |  |  |

[**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 –

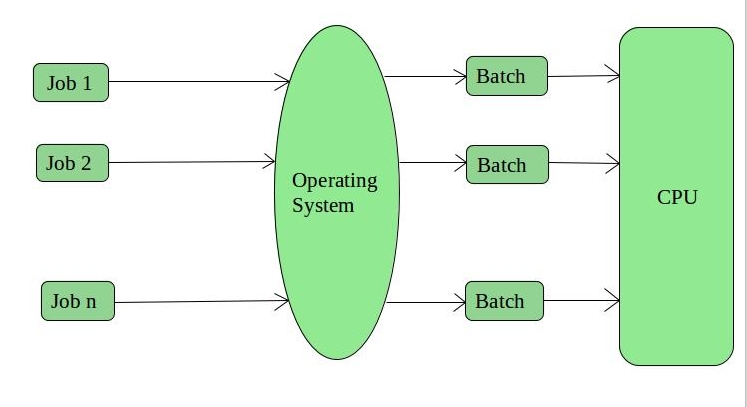
* Windows (GUI based, PC)
* GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
* macOS (Macintosh), used for Apple’s personal computers and workstations (MacBook, iMac).
* Android (Google’s Operating System for smartphones/tablets/smartwatches)
* iOS (Apple’s OS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

# Types of Operating Systems

An Operating System performs all the basic tasks like managing files, processes, and memory. Thus operating system acts as the manager of all the resources, i.e. **resource manager**. Thus, the operating system becomes an interface between user and machine.

**Types of Operating Systems:** Some widely used operating systems are as follows-

**1. Batch Operating System –**   
This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having the same requirement and group them into batches. It is the responsibility of the operator to sort jobs with similar needs.



**Advantages of Batch Operating System:**

* It is very difficult to guess or know the time required for any job to complete. Processors of the batch systems know how long the job would be when it is in queue
* Multiple users can share the batch systems
* The idle time for the batch system is very less
* It is easy to manage large work repeatedly in batch systems

**Disadvantages of Batch Operating System:**

* The computer operators should be well known with batch systems
* Batch systems are hard to debug
* It is sometimes costly
* The other jobs will have to wait for an unknown time if any job fails

**Examples of Batch based Operating System:** Payroll System, Bank Statements, etc.

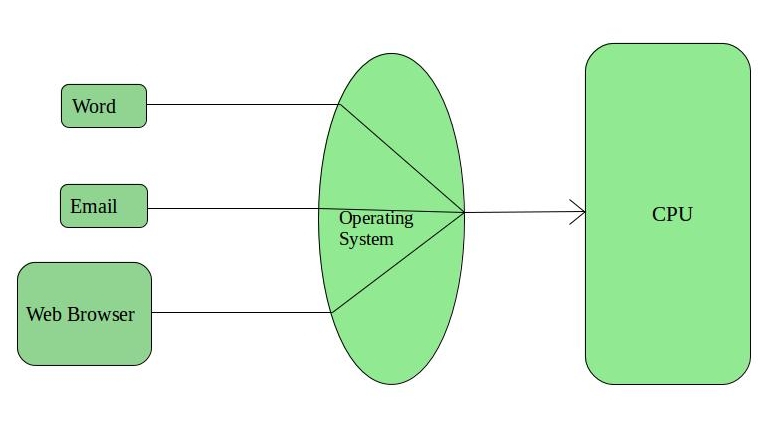
**Batch Operating system** is one of the important type of operating system.

The users who using a batch operating system do not interact with the computer directly. Each user prepares its job on an off-line device like punch cards and submits it to the computer operator. To speed up the processing, jobs with similar needs are batched together and run as a group. The programmers exit their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems that occurs with Batch Systems are as follows −

* There is a lack of interaction between the user and the job.
* CPU is being often idle, because the speed of the mechanical I/O devices is slower than the CPU.
* It is difficult to provide the desired priority.

**2. Time-Sharing Operating Systems –**   
Each task is given some time to execute so that all the tasks work smoothly. Each user gets the time of CPU as they use a single system. These systems are also known as Multitasking Systems. The task can be from a single user or different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to the next task.



**Advantages of Time-Sharing OS:**

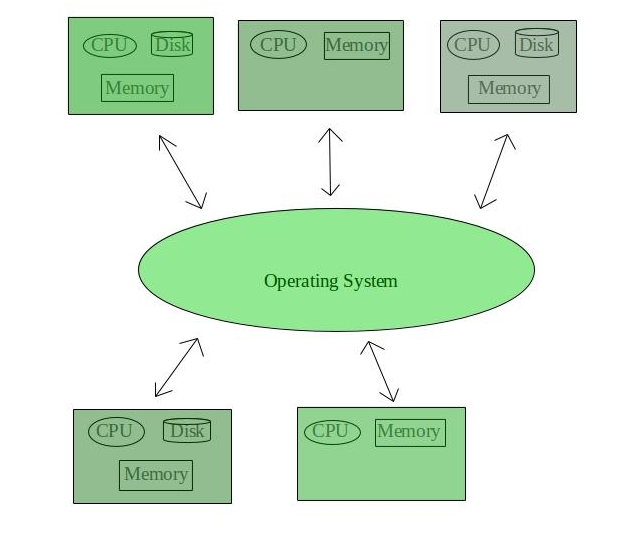
* Each task gets an equal opportunity
* Fewer chances of duplication of software
* CPU idle time can be reduced

**Disadvantages of Time-Sharing OS:**

* Reliability problem
* One must have to take care of the security and integrity of user programs and data
* Data communication problem

**Examples of Time-Sharing OSs are:** Multics, Unix, etc.

**3. Distributed Operating System –**   
These types of the operating system is a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, with a great pace. Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as **loosely coupled systems** or distributed systems. These system’s processors differ in size and function. The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network i.e., remote access is enabled within the devices connected in that network. 



**Advantages of Distributed Operating System:**

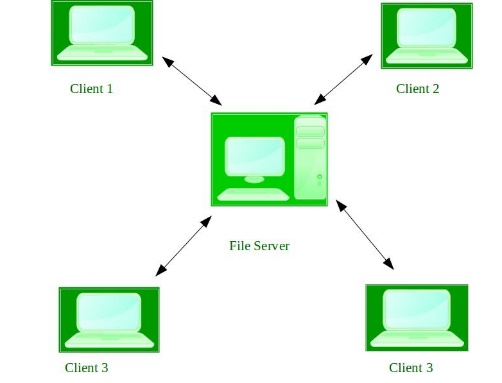
* Failure of one will not affect the other network communication, as all systems are independent from each other
* Electronic mail increases the data exchange speed
* Since resources are being shared, computation is highly fast and durable
* Load on host computer reduces
* These systems are easily scalable as many systems can be easily added to the network
* Delay in data processing reduces

**Disadvantages of Distributed Operating System:**

* Failure of the main network will stop the entire communication
* To establish distributed systems the language which is used are not well defined yet
* These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

**Examples of Distributed Operating System are-** LOCUS, etc.

**4. Network Operating System –**   
These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions. These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections, etc. and that’s why these computers are popularly known as **tightly coupled systems**.



**Advantages of Network Operating System:**

* Highly stable centralized servers
* Security concerns are handled through servers
* New technologies and hardware up-gradation are easily integrated into the system
* Server access is possible remotely from different locations and types of systems

**Disadvantages of Network Operating System:**

* Servers are costly
* User has to depend on a central location for most operations
* Maintenance and updates are required regularly

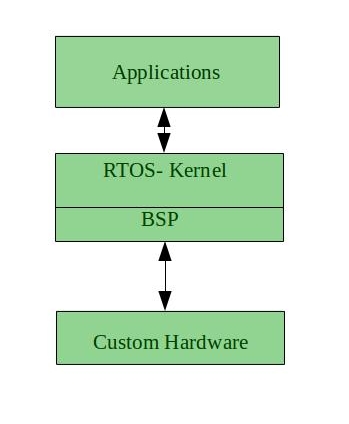
**Examples of Network Operating System are:** Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc.

**5. Real-Time Operating System –**   
These types of OSs serve real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

**Real-time systems** are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

**Two types of Real-Time Operating System which are as follows:**

* **Hard Real-Time Systems:**   
  These OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or airbags which are required to be readily available in case of any accident. Virtual memory is rarely found in these systems.
* **Soft Real-Time Systems:**   
  These OSs are for applications where for time-constraint is less strict.



**Advantages of RTOS:**

* **Maximum Consumption:** Maximum utilization of devices and system, thus more output from all the resources
* **Task Shifting:** The time assigned for shifting tasks in these systems are very less. For example, in older systems, it takes about 10 microseconds in shifting one task to another, and in the latest systems, it takes 3 microseconds.
* **Focus on Application:** Focus on running applications and less importance to applications which are in the queue.
* Real-time**operating system in**the **embedded system:** Since the size of programs are small, RTOS can also be used in embedded systems like in transport and others.
* **Error Free:** These types of systems are error-free.
* **Memory Allocation:** Memory allocation is best managed in these types of systems.

**Disadvantages of RTOS:**

* **Limited Tasks:** Very few tasks run at the same time and their concentration is very less on few applications to avoid errors.
* **Use heavy system resources:** Sometimes the system resources are not so good and they are expensive as well.
* **Complex Algorithms:** The algorithms are very complex and difficult for the designer to write on.
* **Device driver and interrupt signals:** It needs specific device drivers and interrupts signals to respond earliest to interrupts.
* **Thread Priority:** It is not good to set thread priority as these systems are very less prone to switching tasks.

**Examples of Real-Time Operating Systems are:** Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

# Multiprogramming Operating System

## **What is the Multiprogramming Operating System?**

A multiprogramming operating system may run many programs on a single processor computer. If one program must wait for an input/output transfer in a multiprogramming operating system, the other programs are ready to use the CPU. As a result, various jobs may share CPU time. However, the execution of their jobs is not defined to be at the same time period.

When a program is being performed, it is known as a **"Task", "Process"**, and **"Job"**. Concurrent program executions improve system resource consumption and throughput as compared to serial and batch processing systems.

The primary goal of multiprogramming is to manage the entire system's resources. The key components of a multiprogramming system are the file system, command processor, transient area, and I/O control system. As a result, multiprogramming operating systems are designed to store different programs based on sub-segmenting parts of the transient area. The resource management routines are linked with the operating system core functions.

## **Types of the Multiprogramming Operating System**

There are mainly two types of multiprogramming operating systems. These are as follows:

1. **Multitasking Operating System**
2. **Multiuser Operating System**

### **Multitasking Operating System**

A multitasking [operating system](https://www.javatpoint.com/os-tutorial) enables the execution of two or more programs at the same time. The operating system accomplishes this by shifting each program into and out of memory one at a time. When a program is switched out of memory, it is temporarily saved on disk until it is required again.

### **Multiuser Operating System**

A multiuser operating system allows many users to share processing time on a powerful central computer from different terminals. The operating system accomplishes this by rapidly switching between terminals, each of which receives a limited amount of processor time on the central computer. The operating system changes among terminals so quickly that each user seems to have continuous access to the central computer. If there are many users on a system like this, the time it takes the central computer to reply can become more obvious.

## **Working of the Multiprogramming Operating System**

Multiple users can accomplish their jobs simultaneously in the multiprogramming system, and it can be stored in the main memory. When one program is engaged in I/O operations, the CPU may deliver time to various programs while sitting in idle mode.

When one application is waiting for an I/O transfer, another is ready to use the processor at all times, and numerous programs may share [CPU](https://www.javatpoint.com/cpu-full-form) time. All jobs are not run simultaneously, but there could be numerous jobs running on the processor at the same time, and parts of other processes being executed first, then another segment, etc. As a result, the overall goal of a multiprogramming system is to keep the CPU busy until some tasks are available in the job pool. Thus, the numerous programs can run on a single processor computer, and the CPU is never idle.

(Non preemptive)

## **Examples of Multiprogramming Operating System**

There are various examples of multiprogramming operating systems, including download apps, transfer data, [MS-Excel](https://www.javatpoint.com/excel-tutorial), [Google Chrome](https://www.javatpoint.com/google-chrome), [Firefox browser](https://www.javatpoint.com/mozilla-firefox), and many more apps. Other examples are Windows O/S, UNIX O/S, Microcomputers such as XENIX, MP/M, and ESQview.

## **Advantages and Disadvantages of Multiprogramming Operating System**

There are various advantages and disadvantages of the multiprogramming operating system. Some of the advantages and disadvantages are as follows:

### **Advantages**

There are various advantages of the multiprogramming operating system. Some of the advantages are as follows:

1. It provides less response time.
2. It may help to run various jobs in a single application simultaneously.
3. It helps to optimize the total job throughput of the computer.
4. Various users may use the multiprogramming system at once.
5. Short-time jobs are done quickly in comparison to long-time jobs.
6. It may help to improve turnaround time for short-time tasks.
7. It helps in improving CPU utilization and never gets idle.
8. The resources are utilized smartly.

### **Disadvantages**

There are various disadvantages of the multiprogramming operating system. Some of the disadvantages are as follows:

1. It is highly complicated and sophisticated.
2. The CPU scheduling is required.
3. Memory management is needed in the operating system because all types of tasks are stored in the main memory.
4. The harder task is to handle all processes and tasks.
5. If it has a large number of jobs, then long-term jobs will require a long wait.

## What is Multitasking in OS?

**Definition** – Multitasking operating system provides the interface for executing the multiple program tasks by single user at a same time on the one computer system. For example, any editing task can be performed while other programs are executing concurrently. Other example, user can open Gmail and Power Point same time.

## **Types of Multitasking Operating System**

There are **different types of multitasking OS** and below explained each one in detail:

### **True Multitasking**

True multitasking is the capable for executing and process multiple tasks concurrently without taking delay instead of switching tasks from one processor to other processor. It can perform couple of tasks in parallel with underlying the H/W or S/W.

### **Preemptive Multitasking**

Preemptive multitasking is special task that is assigned to [**computer operating system**](https://digitalthinkerhelp.com/what-is-operating-system-and-its-types-uses/), in which it takes decision that how much time spent by one task before assigning other task for using the operating system. Operating system has control for completing this entire process, so it is known as “Preemptive”.

### **Cooperative Multitasking**

Cooperative multitasking is known as “Non-Preemptive Multitasking”. Main goal of Cooperative multitasking is to run currently task, and to release the CPU to allow another task run. This task is performed by calling taskYIELD().Context-switch is executed when this function is called.

## **Advantages of Multitasking Operating System**

**Time Shareable**

In which, all tasks are allocated specific piece of time, so they do not need for waiting time for CPU.

**Manage Several Users**

This [**operating system**](https://digitalthinkerhelp.com/what-is-operating-system-and-its-types-uses/) is more comfort for handling the multiple users concurrently, and several programs can run smoothly without degradation of system’s performance.

**Secured Memory**

Multitasking operating system has well defined memory management, because this operating system does not provide any types of permissions of unwanted programs to wasting the memory.

**Great Virtual Memory**

Multitasking operating system contains the best virtual memory system. Due to virtual memory, any program do not need long waiting g time for completion their tasks, if this problem is occurred then those programs are transferred to virtual memory.

**Background Processing**

Multitasking operating system creates the better environment to execute the background programs. These background programs are not [transparent](https://en.wikipedia.org/wiki/Transparency) for normal users, but these programs help to run other programs smoothly such as firewall, antivirus software, and more.

**Good Reliability**

Multitasking operating system provides the several flexibilities for multiple users, and they are more satisfied to them. On which, every users can operate single or multiple programs with smoothly.

**Use Multiple Programs**

Users can operate multiple programs such as internet browser, PowerPoint, MS Excel, games, and other utilities concurrently.

**Optimize Computer Resources**

Multitasking operating system is able to handle smoothly multiple [**computers’ resources**](https://digitalthinkerhelp.com/types-of-computer-hardware-components-devices-parts/) such as RAM, [**input/output devices**](https://digitalthinkerhelp.com/types-of-computer-hardware-components-devices-parts/), CPU, hard disk, and more.

## **Disadvantages of Multitasking Operating System**

**Memory Boundation**

Computer can get slow performance, due to run multiple programs at a same time because [**main memory**](https://digitalthinkerhelp.com/computer-memory-primary-secondary-storage-devices/) gets more load while loading multiple programs. CPU is not able to provide separate time for every program, and its response time gets increase. Main reason of occurring this problem is that it uses to less capacity RAM. So, for getting solution can be increased the RAM capacity.

**Processor Boundation**

Computer can run programs slowly due to slow speed of their processors, and its response time can increase while handling multiple programs. Need better processing power,  to overcome this problem.

**CPU Heat up**

Multiple processors become busier at a time for executing any task in multitasking nature, So CPU produces more heat.

## **Examples of Multitasking Operating System**

There are some **examples of multi tasking OS** like as –

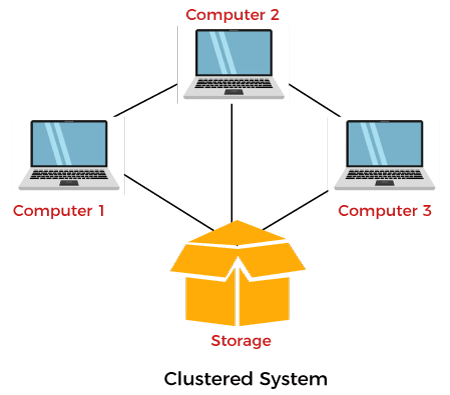
* Windows XP
* Windows Vista
* Windows 7
* Windows 8
* Windows 10
* Windows 2000
* IBM’s OS/390
* Linux
* UNIX

# Clustered Operating System

Cluster systems are similar to parallel systems because both systems use multiple CPUs. The primary difference is that clustered systems are made up of two or more independent systems linked together. They have independent computer systems and a shared storage media, and all systems work together to complete all tasks. All cluster nodes use two different approaches to interact with one another, like **message passing interface (MPI)** and **parallel virtual machine (PVM)**.

In this article, you will learn about the Clustered Operating system, its types, classification, advantages, and disadvantages.

## **What is the Clustered Operating System?**



Cluster operating systems are a combination of **software** and **hardware** clusters. Hardware clusters aid in the sharing of high-performance disks among all computer systems, while software clusters give a better environment for all systems to operate. A cluster system consists of various nodes, each of which contains its cluster software. The cluster software is installed on each node in the clustered system, and it monitors the cluster system and ensures that it is operating properly. If one of the clustered system's nodes fails, the other nodes take over its storage and resources and try to restart.

Cluster components are generally linked via fast area networks, and each node executing its instance of an operating system. In most cases, all nodes share the same hardware and operating system, while different hardware or different operating systems could be used in other cases. The primary purpose of using a cluster system is to assist with weather forecasting, scientific computing, and supercomputing systems.

There are two clusters available to make a more efficient cluster. These are as follows:

1. **Software Cluster**
2. **Hardware Cluster**

**Software Cluster**

The Software Clusters allows all the systems to work together.

**Hardware Cluster**

It helps to allow high-performance disk sharing among systems.

## **Types of Clustered Operating System**

There are mainly three types of the clustered operating system:

1. **Asymmetric Clustering System**
2. **Symmetric Clustering System**
3. **Parallel Cluster System**

### **Asymmetric Clustering System**

In the asymmetric cluster system, one node out of all nodes is in hot standby mode, while the remaining nodes run the essential applications. Hot standby mode is completely fail-safe and also a component of the cluster system. The node monitors all server functions; the hot standby node swaps this position if it comes to a halt.

### **Symmetric Clustering System**

Multiple nodes help run all applications in this system, and it monitors all nodes simultaneously. Because it uses all hardware resources, this cluster system is more reliable than asymmetric cluster systems.

### **Parallel Cluster System**

A parallel cluster system enables several users to access similar data on the same shared storage system. The system is made possible by a particular software version and other apps.

## **Classification of clusters**

Computer clusters are managed to support various purposes, from general-purpose business requirements like web-service support to computation-intensive scientific calculations. There are various classifications of clusters. Some of them are as follows:

**1. Fail Over Clusters**

The process of moving applications and data resources from a failed system to another system in the cluster is referred to as fail-over. These are the databases used to cluster important missions, application servers, mail, and file.

**2. Load Balancing Cluster**

The cluster requires better load balancing abilities amongst all available computer systems. All nodes in this type of cluster can share their computing workload with other nodes, resulting in better overall performance. For example, a web-based cluster can allot various web queries to various nodes, so it helps to improve the system speed. When it comes to grabbing requests, only a few cluster systems use the round-robin method.

**3. High Availability Clusters**

These are also referred to as "HA clusters". They provide a high probability that all resources will be available. If a failure occurs, such as a system failure or the loss of a disk volume, the queries in the process are lost. If a lost query is retried, it will be handled by a different cluster computer. It is widely used in news, email, FTP servers, and the web.

## **Advantages and Disadvantages of Cluster Operating System**

Various advantages and disadvantages of the Clustered Operating System are as follows:

### **Advantages**

Various advantages of Clustered Operating System are as follows:

**1. High Availability**

Although every node in a cluster is a standalone computer, the failure of a single node doesn't mean a loss of service. A single node could be pulled down for maintenance while the remaining clusters take on a load of that single node.

**2. Cost Efficiency**

When compared to highly reliable and larger storage mainframe computers, these types of cluster computing systems are thought to be more cost-effective and cheaper. Furthermore, most of these systems outperform mainframe computer systems in terms of performance.

**3. Additional Scalability**

A cluster is set up in such a way that more systems could be added to it in minor increments. Clusters may add systems in a horizontal fashion. It means that additional systems could be added to clusters to improve their performance, fault tolerance, and redundancy.

**4. Fault Tolerance**

Clustered systems are quite fault-tolerance, and the loss of a single node does not result in the system's failure. They might also have one or more nodes in hot standby mode, which allows them to replace failed nodes.

**5. Performance**

The clusters are commonly used to improve the availability and performance over the single computer systems, whereas usually being much more cost-effective than the single computer system of comparable speed or availability.

**6. Processing Speed**

The processing speed is also similar to mainframe systems and other types of supercomputers on the market.

### **Disadvantages**

Various disadvantages of the Clustered Operating System are as follows:

**1. Cost-Effective**

One major disadvantage of this design is that it is not cost-effective. The cost is high, and the cluster will be more expensive than a non-clustered server management design since it requires good hardware and a design.

**2. Required Resources**

Clustering necessitates the use of additional servers and hardware, making monitoring and maintenance difficult. As a result, infrastructure must be improved.

**3. Maintenance**

It isn't easy to system establishment, monitor, and maintenance this system.

# Embedded Operating System

An embedded operating system is a computer operating system designed for use in embedded computer systems. These operating systems are designed to be **small, resource-efficient, dependable**, and reduce many features that aren't required by specialized applications.

The hardware that runs an embedded operating system is usually quite resource-constrained. Embedded hardware systems are typically quite specific, and it means that these systems are designed to cover certain tasks due to limited resources.

In this article, you will learn about the embedded operating system with its types and many other features.

## **What is Embedded Operating System?**

An embedded operating system is a computer operating system designed for use in embedded computer systems. It has limited features. The term "embedded operating system" also refers to a "real-time operating system". The main goal of designing an embedded [operating system](https://www.javatpoint.com/os-tutorial) is to perform specified tasks for non-computer devices. It allows the executing programming codes that deliver access to devices to complete their jobs.

An embedded operating system is a combination of software and hardware. It produces an easily understandable result by humans in many formats such as **images, text**, and **voice**. Embedded operating systems are developed with programming code, which helps convert hardware languages into software languages like [**C**](https://www.javatpoint.com/c-programming-language-tutorial) and [**C++**](https://www.javatpoint.com/cpp-tutorial).

The embedded operating system improves overall efficiency by controlling all hardware resources and minimizing response times for specific tasks for which devices were built.

## **How does an Embedded System work?**

People commonly wonder how an embedded system works since there is a high need for complex product technology, which provides opportunities for embedded software developers. In contrast to a desktop PC, which loads or runs applications, an embedded operating system is built for fewer tasks and typically handles a single application on a device.

Due to the limited scope of operating system functions, it must be reliable and run smoothly with its size, processing power, and requirements. That specific application is essential to the end product's functionality. Wind River VxWorks, Embedded Linux and Android, and QNX are some of the top embedded operating systems for commercial and industrial applications.

## **History of Embedded Operating System**

The **Apollo Guidance Computer (AGC)** introduced embedded operating systems in the **1960s**, while Linux and Android are more new developments to the market. The **AGC** is a non-profit organization that promotes each **Apollo command module (ACM)**, and **Apollo Lunar Module (ALM)** had it installed. The AGC offered computation and electronic interfaces for the navigation, guidance, and control of the spacecraft.

Since then, embedded computer systems have gone a long way, and the evolution appears to be far from over, owing to advancements in microcontroller technology from **8-bit** to **16-bit,** and eventually **32-bit**. Since embedded computer systems are growing more complex, operating systems have grown essential to manage embedded software effectively.

## **Types of Embedded Operating System**

There are various types of Embedded operating systems. Some of them are as follows:

### **Real-Time Operating System**

A **real-time operating system (RTOS)** is a deterministic operating system with limited functionalities that allows multi-threaded applications by giving processed outputs within set time limitations. Since some apps are time-critical, they must be executed exactly when they are expected to maintain the entire system functioning.

The real-time operating system is dependent on clock interruptions. **Interrupt Service Routine (ISR)** interruptions are generated by this system. The Priority system was implemented by **RTOS** for the execution of all types of processes. The process and the **RTOS** are synchronized and can communicate with one another. The **RTOS** is stored on a [**ROM (Read Only Memory)**](https://www.javatpoint.com/rom) chip because this chip can store data for a long time.

### **Multi-tasking Operating System**

The [multitasking operating system](https://www.javatpoint.com/multitasking-operating-system) may execute multiple tasks at the same time. In a multitasking operating system, multiple tasks and processes run at the same time. If the system contains more than one processor, it may perform a wide range of functions.

The multitasking operating system is switched between the multiple tasks. Some tasks are waiting for events to occur, while others are receiving events and preparing to run. When using a multitasking operating system, software development is easier since different software components may be made independent of each other.

### **Preemptive Operating System**

A multitasking operating system that interprets task preemption is known as a preemptive operating system. A task with a higher priority is always defined and executed before a task with a lower priority. Such multitasking operating systems improve system reaction to events and simplify software development, resulting in a more dependable system. The system designer may calculate the time required for service interpreters in the system and the time required by the scheduler to switch tasks. Such systems can fail to meet a system's deadline, and the program is unaware of the missed deadline. CPU load can be naturally measured in a preemptive operating system by defining a lower priority process that does nothing except increment the counter.

### **Rate Monotonic Operating System**

Some embedded systems are designed to use a specific task scheduling method known as **'Rate Monotonic Scheduling'**. It is an operating system that assures that tasks in a system may operate for a specific amount of time and duration of time. It is a priority-based scheduling algorithm. It is used in operating systems as a preemptive. It means that all tasks can be interrupted or suspended by other tasks within a short period of time. It is generally used to perform shorter tasks with higher priority.

### **Single System Control Loop**

It is a very simple type of operating system designed to perform only one function. It is used in several devices, including smartphones, thermostats or temperature controls, digital controllable equipment, etc. Users may set any point of temperature variable as desired in this type of OS. Several sensors are included in this system to determine various temperature points in the environment.

## **Characteristics of Embedded Operating System**

There are various characteristics of an embedded operating system. Some of them are as follows:

1. It provides real-time operations.
2. Direct use of interrupts
3. Input/Output device flexibility
4. Reactive operation
5. Streamlined protection mechanisms
6. Configurability

## **Popular Embedded Operating Systems**

There are various popular embedded operating systems. Some of them are as follows:

### **eCos**

It stands for **'Embedded Configurable Operating System'**, and all of its components provide a wide range of configuration options. The eCos operating system may support a wide range of popular embedded CPUs.

### **mbed OS**

It is a free and open-source embedded operating system that offers a systematic and comprehensive environment for intelligent hardware development.

### **VxWorks**

**Wind River Company** firstly introduced it in **1983**. It is supported with task synchronization, memory efficiency management, and other features.

### **µC/OS-II:**

It is introduced based on the **μC/OS** principle. **μC/OS-II** may handle **64** tasks and provide various functionalities such as interrupt services, task scheduling, memory management, synchronization, and time management.

### **FreeRTOS**

It is a lightweight operating system that supports the priority scheduling algorithm. It provides various functionalities like memory management, message queue, task management, semaphore, time management, etc.

### **QNX**

QNX was created in **1980** and is a commercial embedded real operating system that requires the POSIX specification to compile.

### **µ Clinux**

It stands for **'Micro-Control Linux'**, and it is the latest version of embedded Linux. It is capable of grab all features of the Linux operating system.

### **Embedded Operating System Uses**

The embedded operating system is commonly used in various areas, including car navigation systems, multimedia players, airplane navigation systems, and medical equipment.

### **Car navigation system**

The car navigation system is a small computer system with a touch screen that enables the driver to navigate numerous menus such as audio playback, radio, GPS and route mapping, fuel level, hands-free calls, and tire pressure monitoring systems. All of these tasks are performed by the computer to improve the driving experience.

### **Parking Metering**

Smart city parking meters use the embedded system to manage the user input and track time and costs. Depending on the design, these devices contain a variety of built-in functions. For example, some sensors detect vehicle entry and exit, while others require the driver to enter the parking space or vehicle license. A user interface offers the driver options, including defining the expected return time and paying appropriately.

### **Medical Equipment**

Medical equipment automatically monitors bio constants, administers drugs. If the bio constants exceed or fall below a threshold value, it alerts the staff. As a result, it may help doctors treat the patients, monitor health issues, and save their lives.

### **The navigation system of a plane**

The navigation system of a plane is a good instance of a real-time operating system. The main computer of an airplane is connected to most control systems such as the wing, engine, pressure controls, and safety. As a result, it is specifically built to work inside a plane and help with takeoff, landing, and emergency operations.

## **Advantages and disadvantages of Embedded Operating System**

There are various advantages and disadvantages of an embedded operating system. Some of them are as follows:

### **Advantages**

There are various advantages of an embedded operating system. Some of them are as follows:

1. It is small in size and faster to load.
2. It is low cost.
3. It is easy to manage.
4. It provides better stability.
5. It provides higher reliability.
6. It provides some interconnections.
7. It has low power consumption.
8. It helps to increase the product quality.

### **Disadvantages**

There are various disadvantages of an embedded operating system. Some of them are as follows:

1. It isn't easy to maintain.
2. The troubleshooting is harder.
3. It has limited resources for memory.
4. It isn't easy to take a back of embedded files.
5. You can't change, improve, or upgrade an embedded system once it's been developed.
6. If any problem occurs, you need to reset the setting.
7. Its hardware is limited.

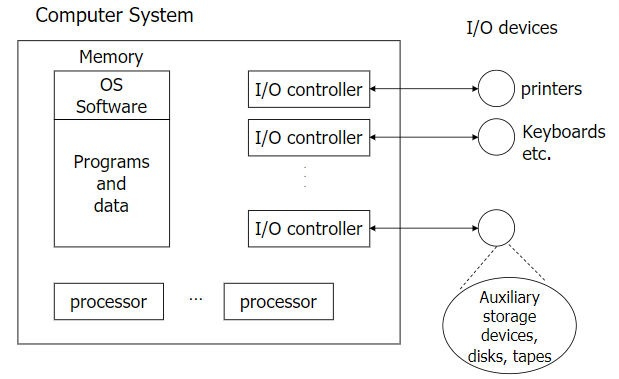
**Resource Manager view**

## **Operating System as Resource Manager**

Let us understand how the operating system works as a Resource Manager.

* Now-a-days all modern computers consist of processors, memories, timers, network interfaces, printers, and so many other devices.
* The operating system provides for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs in the bottom-up view.
* Operating system allows multiple programs to be in memory and run at the same time.
* Resource management includes multiplexing or sharing resources in two different ways: in time and in space.
* In time multiplexed, different programs take a chance of using CPU. First one tries to use the resource, then the next one that is ready in the queue and so on. For example: Sharing the printer one after another.
* In space multiplexing, Instead of the customers taking a chance, each one gets part of the resource. For example − Main memory is divided into several running programs, so each one can be resident at the same time.

The diagram given below shows the functioning of OS as a resource manager −



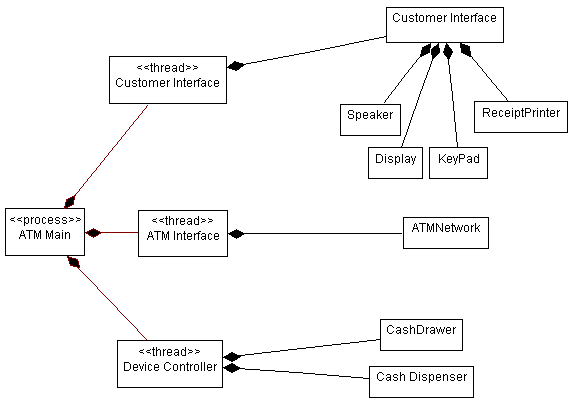
## Concepts:  Process View

To provide a basis for understanding the process organization of the system, an architectural view called the **process view** is used in the Analysis & Design discipline. There is only one process view of the system, which illustrates the process decomposition of the system, including the mapping of classes and subsystems on to [processes](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/process/glossary.htm#process) and [threads](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/process/glossary.htm#thread). The process view is refined during each iteration. As [[BOO98](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/process/referenc.htm#BOO98)] states: "With UML, the static and dynamic aspects of this view are captured in the same kinds of diagrams as for the design view - i.e. class diagrams, interaction diagrams, activity diagrams and statechart diagrams, but with a focus on the active classes that represent these threads and processes." Of concern when constructing and using the process view are, for example, issues of concurrency, response time, deadlock, throughput, fault tolerance, and scalability.

It is possible to design for concurrency without the use of direct underlying operating system support - for example using a specially written scheduler or other run-time support. In such cases, concurrency is simulated at the application infrastructure level, rather than in the operating system. If necessary, other stereotypes (in addition to the standard threads and processes) may be used to make this distinction (to guide implementation). For example, the Ada programming language contains its own model of concurrency, based on Ada tasks; the Ada run-time has to provide this, whether or not the operating system on which it runs has an appropriate equivalent - threads, say - which could be used to support Ada tasking.

In real-time systems, the Rational Unified Process recommends the use of [Capsule](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/addin_realtime/process/artifact/ar_cpsl.htm)s to represent active classes in the process view. Capsules have strong semantics to simplify the modeling of concurrency:

* they use asynchronous message-based communication through [Ports](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/addin_realtime/process/artifact/ar_cpsl.htm#Ports) using well-defined [Protocol](https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/addin_realtime/process/artifact/ar_prot.htm)s;
* they use run-to-completion semantics for message processing;
* they encapsulate passive objects (ensuring that thread interference cannot occur).



The process view shows the process organization of the system.

There are four additional views, the **Use-Case View**(handled in the Requirements discipline), and the **Logical View**, **Deployment View**, and **Implementation View**; these views are handled in the Analysis & Design and Implementation disciplines.

The architectural views are documented in a **Software Architecture Document**. You may add different views, such as a security view, to convey other specific aspects of the software architecture.

So in essence, architectural views can be seen as abstractions or simplifications of the models built, in which you make important characteristics more visible by leaving the details aside. The architecture is an important means for increasing the quality of any model built during system development.

# Hierarchical Architecture

Hierarchical architecture views the whole system as a hierarchy structure, in which the software system is decomposed into logical modules or subsystems at different levels in the hierarchy. This approach is typically used in designing system software such as network protocols and operating systems.

In system software hierarchy design, a low-level subsystem gives services to its adjacent upper level subsystems, which invoke the methods in the lower level. The lower layer provides more specific functionality such as I/O services, transaction, scheduling, security services, etc. The middle layer provides more domain dependent functions such as business logic and core processing services. And, the upper layer provides more abstract functionality in the form of user interface such as GUIs, shell programming facilities, etc.

It is also used in organization of the class libraries such as .NET class library in namespace hierarchy. All the design types can implement this hierarchical architecture and often combine with other architecture styles.

Hierarchical architectural styles is divided as −

* Main-subroutine
* Master-slave
* Virtual machine

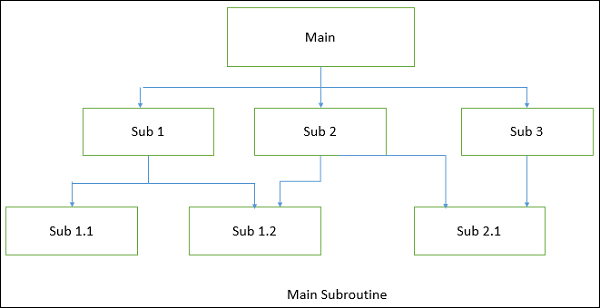
## **Main-subroutine**

The aim of this style is to reuse the modules and freely develop individual modules or subroutine. In this style, a software system is divided into subroutines by using top-down refinement according to desired functionality of the system.

These refinements lead vertically until the decomposed modules is simple enough to have its exclusive independent responsibility. Functionality may be reused and shared by multiple callers in the upper layers.

There are two ways by which data is passed as parameters to subroutines, namely −

* **Pass by Value** − Subroutines only use the past data, but can’t modify it.
* **Pass by Reference** − Subroutines use as well as change the value of the data referenced by the parameter.



### **Advantages**

* Easy to decompose the system based on hierarchy refinement.
* Can be used in a subsystem of object oriented design.

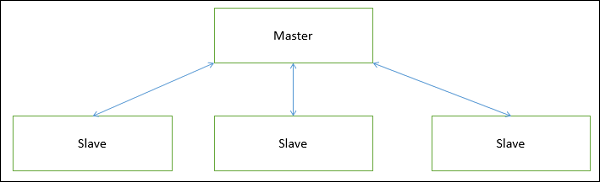
### **Disadvantages**

* Vulnerable as it contains globally shared data.
* Tight coupling may cause more ripple effects of changes.

## **Master-Slave**

This approach applies the 'divide and conquer' principle and supports fault computation and computational accuracy. It is a modification of the main-subroutine architecture that provides reliability of system and fault tolerance.

In this architecture, slaves provide duplicate services to the master, and the master chooses a particular result among slaves by a certain selection strategy. The slaves may perform the same functional task by different algorithms and methods or totally different functionality. It includes parallel computing in which all the slaves can be executed in parallel.



The implementation of the Master-Slave pattern follows five steps −

* Specify how the computation of the task can be divided into a set of equal sub-tasks and identify the sub-services that are needed to process a sub-task.
* Specify how the final result of the whole service can be computed with the help of the results obtained from processing individual sub-tasks.
* Define an interface for the sub-service identified in step 1. It will be implemented by the slave and used by the master to delegate the processing of individual sub-tasks.
* Implement the slave components according to the specifications developed in the previous step.
* Implement the master according to the specifications developed in step 1 to 3.

### **Applications**

* Suitable for applications where reliability of software is critical issue.
* Widely applied in the areas of parallel and distributed computing.

### **Advantages**

* Faster computation and easy scalability.
* Provides robustness as slaves can be duplicated.
* Slave can be implemented differently to minimize semantic errors.

### **Disadvantages**

* Communication overhead.
* Not all problems can be divided.
* Hard to implement and portability issue.

## **Virtual Machine Architecture**

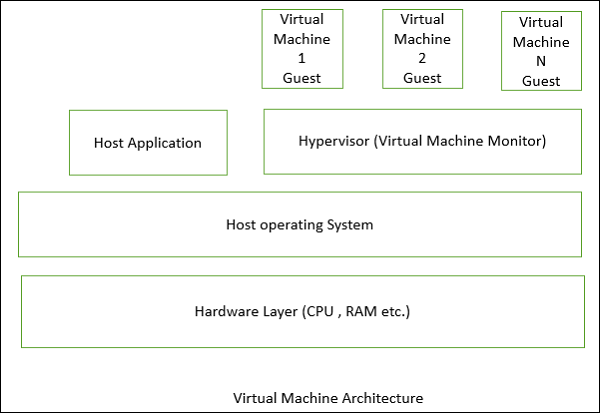
Virtual Machine architecture pretends some functionality, which is not native to the hardware and/or software on which it is implemented. A virtual machine is built upon an existing system and provides a virtual abstraction, a set of attributes, and operations.

In virtual machine architecture, the master uses the ‘same’ subservice’ from the slave and performs functions such as split work, call slaves, and combine results. It allows developers to simulate and test platforms, which have not yet been built, and simulate "disaster'' modes that would be too complex, costly, or dangerous to test with the real system.

In most cases, a virtual machine splits a programming language or application environment from an execution platform. The main objective is to provide **portability**. Interpretation of a particular module via a Virtual Machine may be perceived as −

* The interpretation engine chooses an instruction from the module being interpreted.
* Based on the instruction, the engine updates the virtual machine’s internal state and the above process is repeated.

The following figure shows the architecture of a standard VM infrastructure on a single physical machine.



The **hypervisor,** also called the **virtual machine monitor**, runs on the host OS and allocates matched resources to each guest OS. When the guest makes a system-call, the hypervisor intercepts and translates it into the corresponding system-call supported by the host OS. The hypervisor controls each virtual machine access to the CPU, memory, persistent storage, I/O devices, and the network.

### **Applications**

Virtual machine architecture is suitable in the following domains −

* Suitable for solving a problem by simulation or translation if there is no direct solution.
* Sample applications include interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Smalltalk and Java interpreter typed programming language.
* Common examples of virtual machines are interpreters, rule-based systems, syntactic shells, and command language processors.

### **Advantages**

* Portability and machine platform independency.
* Simplicity of software development.
* Provides flexibility through the ability to interrupt and query the program.
* Simulation for disaster working model.
* Introduce modifications at runtime.

### **Disadvantages**

* Slow execution of the interpreter due to the interpreter nature.
* There is a performance cost because of the additional computation involved in execution.

## **Layered Style**

In this approach, the system is decomposed into a number of higher and lower layers in a hierarchy, and each layer has its own sole responsibility in the system.

* Each layer consists of a group of related classes that are encapsulated in a package, in a deployed component, or as a group of subroutines in the format of method library or header file.
* Each layer provides service to the layer above it and serves as a client to the layer below i.e. request to layer i +1 invokes the services provided by the layer i via the interface of layer i. The response may go back to the layer i +1 if the task is completed; otherwise layer i continually invokes services from layer i -1 below.

### **Applications**

Layered style is suitable in the following areas −

* Applications that involve distinct classes of services that can be organized hierarchically.
* Any application that can be decomposed into application-specific and platform-specific portions.
* Applications that have clear divisions between core services, critical services, and user interface services, etc.

### **Advantages**

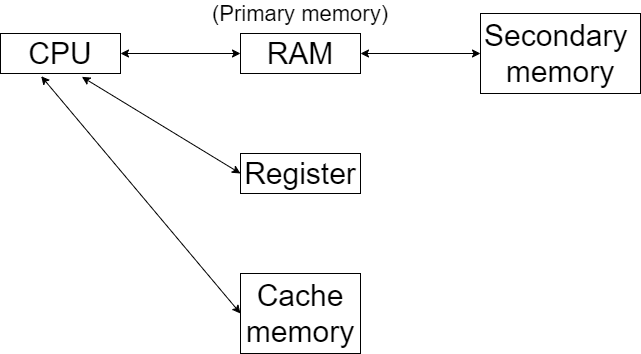
* Design based on incremental levels of abstraction.
* Provides enhancement independence as changes to the function of one layer affects at most two other layers.
* Separation of the standard interface and its implementation.
* Implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.
* Each layer can be an abstract machine deployed independently which support portability.
* Easy to decompose the system based on the definition of the tasks in a top-down refinement manner
* Different implementations (with identical interfaces) of the same layer can be used interchangeably

### **Disadvantages**

* Many applications or systems are not easily structured in a layered fashion.
* Lower runtime performance since a client’s request or a response to client must go through potentially several layers.
* There are also performance concerns on overhead on the data marshaling and buffering by each layer.
* Opening of interlayer communication may cause deadlocks and “bridging” may cause tight coupling.
* Exceptions and error handling is an issue in the layered architecture, since faults in one layer must spread upwards to all calling layers

### **Memory management**

* Memory management in operating system means that it is a method/ functionality to manage different types of memory.
* Mainly memory management is method of managing primary memory or RAM.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--xJqx009F--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/cdn.hashnode.com/res/hashnode/image/upload/v1607996534213/Vd3gtzd6a.png)

### Degree of multiprogramming

* CPU is not connected with secondary memory directly because the speed of secondary memory is very low in comparison of CPU.
* All the process/ programs are present in secondary memory, in order to execute them we have to load them in RAM
* From RAM they will be sent to CPU and get executed.
* This is called degree of multiprogramming.
* Multiprogramming means, try to place more and more processes from secondary memory to primary memory. Because then only the utilization of CPU will be at satisfactory level.
* If we increase the number of process in RAM than CPU utilization will be more and degree of multiprogramming will be also high.

# Memory Management in Operating System

The term Memory can be defined as a collection of data in a specific format. It is used to store instructions and processed data. The memory comprises a large array or group of words or bytes, each with its own location. The primary motive of a computer system is to execute programs. These programs, along with the information they access, should be in the main memory during execution. The CPU fetches instructions from memory according to the value of the program counter.

To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

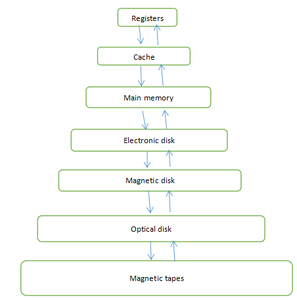
*Here, we will cover the following memory management topics:*

* *What is Main Memory*
* *What is Memory Management*
* *Why memory Management is required*
* *Logical address space and Physical address space*
* *Static and dynamic loading*
* *Static and dynamic linking*
* *Swapping*
* *Contiguous Memory allocation*
  + *Memory Allocation*
    - *First Fit*
    - *Best Fit*
    - *Worst Fit*
  + *Fragmentation*
    - *Internal Fragmentation*
    - *External Fragmentation*
  + *Paging*

**Now before, We start memory management let us known about what is main memory.**

### What is Main Memory:

The main memory is central to the operation of a modern computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs and information are kept when the processor is effectively utilizing them.  Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.  Main memory is also known as RAM(Random Access Memory). This memory is a volatile memory.RAM lost its data when a power interruption occurs.



*Figure 1: Memory hierarchy*

### What is Memory Management :

In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

### Why Memory Management is required:

* Allocate and de-allocate memory before and after process execution.
* To keep track of used memory space by processes.
* To minimize fragmentation issues.
* To proper utilization of main memory.
* To maintain data integrity while executing of process.

**Now we are discussing the concept of logical address space and Physical address space:**

### Logical and Physical Address Space:

**Logical Address space:**An address generated by the CPU is known as “Logical Address”. It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.

**Physical Address space:**An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”. A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

### Static and Dynamic Loading:

To load a process into the main memory is done by a loader. There are two different types of loading :

* **Static loading**:- loading the entire program into a fixed address. It requires more memory space.
* **Dynamic loading**:- The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

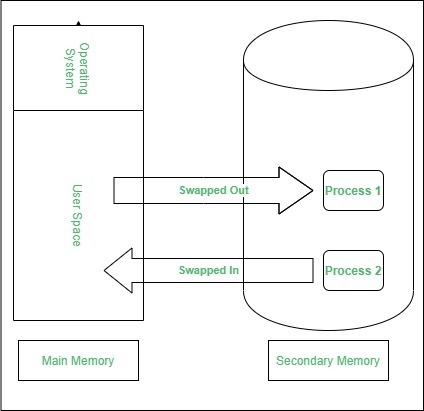
### Static and Dynamic linking:

To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

* **Static linking:**In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
* **Dynamic linking:** The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

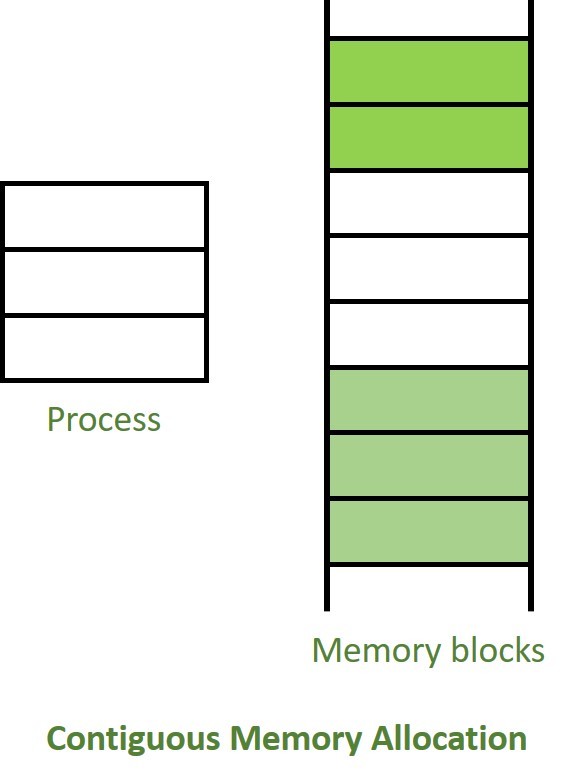
### Swapping :

When a process is executed it must have resided in memory. Swapping is a process of swap a process temporarily into a secondary memory from the main memory, which is fast as compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time directly proportional to the amount of memory swapped. Swapping is also known as roll-out, roll in, because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.



### Contiguous Memory Allocation :

The main memory should oblige both the operating system and the different client processes.  Therefore, the allocation of memory becomes an important task in the operating system.  The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.



### Memory allocation:

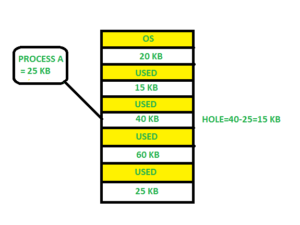
To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.

**Multiple partition allocation**: In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

**Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as “Hole”. When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement fulfills then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:

**First fit:-**

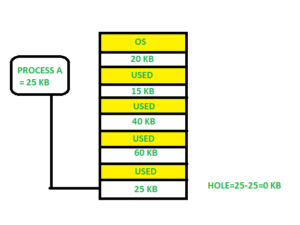
In the first fit, the first available free hole fulfills the requirement of the process allocated.



Here, in this diagram 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

**Best fit:-**

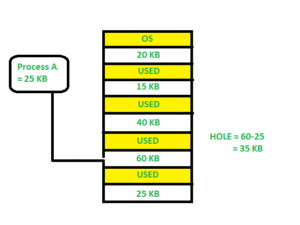
In the best fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB).

In this method memory utilization is maximum as compared to other memory allocation techniques.

**Worst fit:-**In the worst fit, allocate the largest available hole to process. This method produces the largest leftover hole.



Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB. Inefficient memory utilization is a major issue in the worst fit.

### Fragmentation:

A Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process.  To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problem. In operating system two types of fragmentation:

**Internal fragmentation:**

Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is leftover and creates an internal fragmentation problem.

 Example: Suppose there is a fixed partitioning is used for memory allocation and the different size of block 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demand for the block of memory. It gets a memory block of 3MB but 1MB block memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

**External fragmentation:**

In external fragmentation, we have a free memory block, but we can not assign it to process because blocks are not contiguous.

Example: Suppose (consider above example) three process p1, p2, p3 comes with size 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating process p1 process and p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous.  This is called external fragmentation.

Both the first fit and best-fit systems for memory allocation affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

Another possible solution to the external fragmentation is to allow the logical address space of the processes to be noncontiguous, thus permit a process to be allocated physical memory where ever the latter is available.

### Paging:

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
* Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program
* Physical Address (represented in bits): An address actually available on a memory unit
* Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

**Example:**

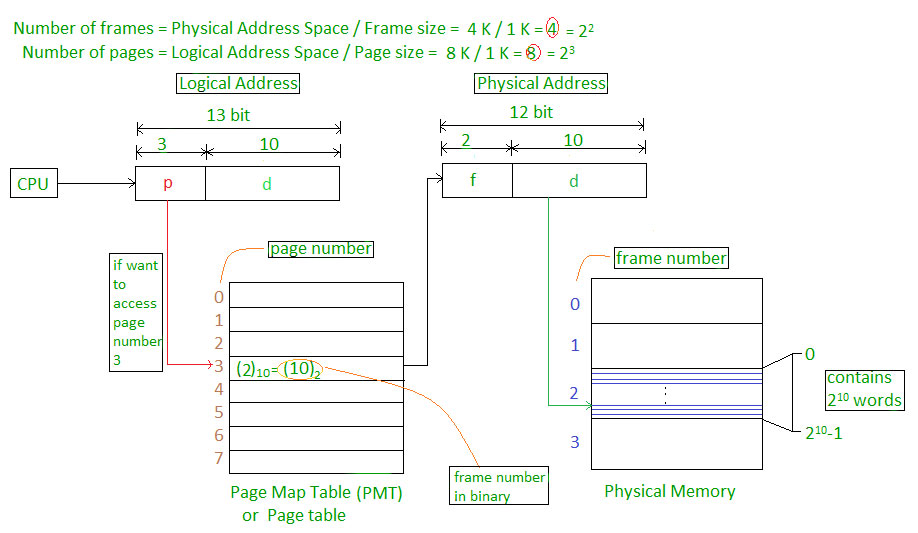
* If Logical Address = 31 bits, then Logical Address Space = 231 words = 2 G words (1 G = 230)
* If Logical Address Space = 128 M words = 27 \* 220 words, then Logical Address = log2 227 = 27 bits
* If Physical Address = 22 bits, then Physical Address Space = 222 words = 4 M words (1 M = 220)
* If Physical Address Space = 16 M words = 24 \* 220 words, then Physical Address = log2 224 = 24 bits

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 the paging technique.

* The Physical Address Space is conceptually divided into several fixed-size blocks, called **frames**.
* The Logical Address Space is also split into fixed-size blocks, called **pages**.
* Page Size = Frame Size

Let us consider an example:

* Physical Address = 12 bits, then Physical Address Space = 4 K words
* Logical Address = 13 bits, then Logical Address Space = 8 K words
* Page size = frame size = 1 K words (assumption)



The address generated by the CPU is divided into

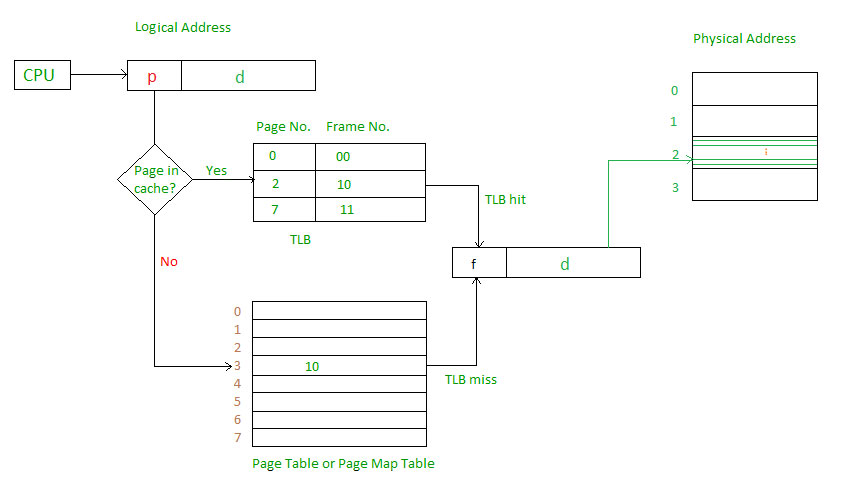
* **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
* **Page offset(d):** Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

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

The hardware implementation of the page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.

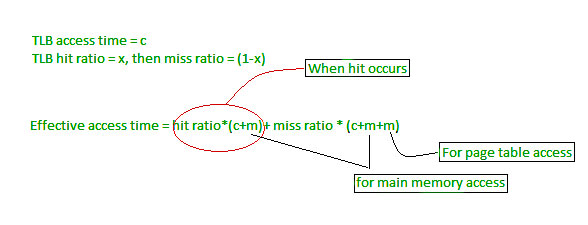
* The TLB is an associative, high-speed memory.
* Each entry in TLB consists of two parts: a tag and a value.
* When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



Main memory access time = m

If page table are kept in main memory,

Effective access time = m(for page table) + m(for particular page in page table)



**Memory management techniques**

### **1. Swapping**

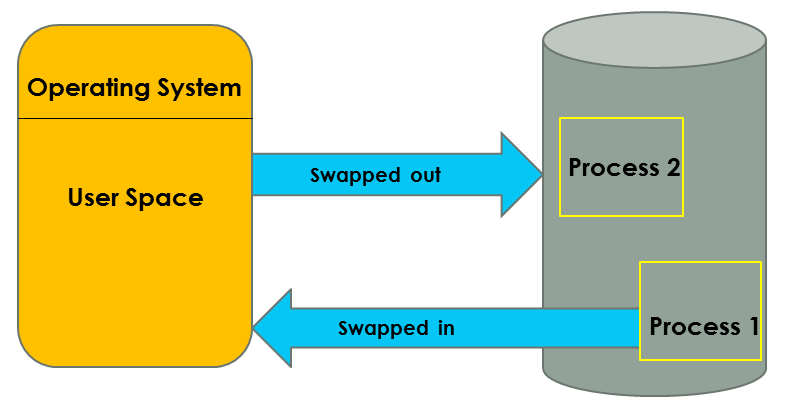
When process is to be executed then that process is taken from secondary memory to stored in RAM.But RAM have limited space so we have to take out and take in the process from RAM time to time. This process is called **swapping**. The purpose is to make a free space for other processes. And later on, that process is swapped back to the main memory.

**The situations in which swapping takes place**

1. The Round Robin algorithm is executing in which quantum process is supposed to preempt after running for some time. In that case, that process is swapped out, and the new process is swapped in.
2. When there is a priority assigned to each process, the process with low priority is swapped out, and the higher priority process is swapped in. After its execution, the lower priority process is again swapped in, and this process is so fast that users will not know anything about it.
3. In shortest time remaining first algorithm when the next process(which arrive in ready queue) is having less burst time,then executing process is preempted.
4. When process have to do I/O operations,then that process temporaily swapped out.

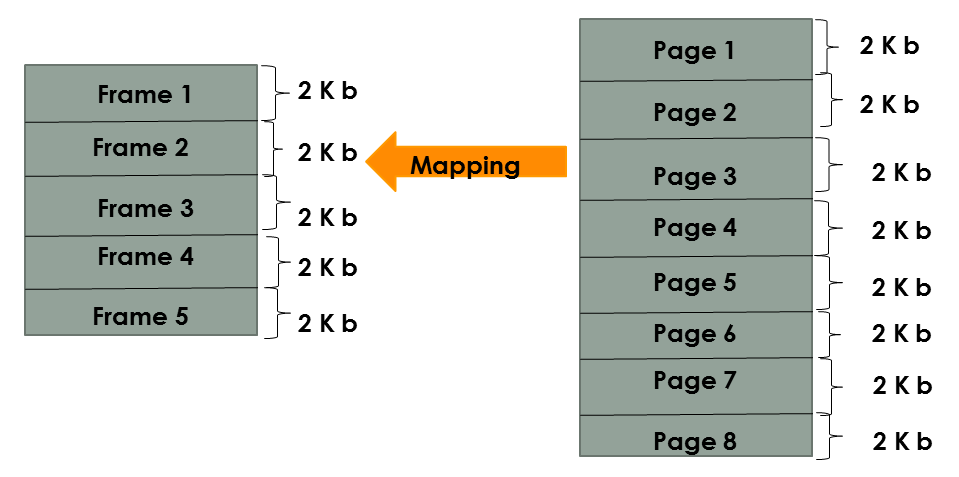
**It is further divided into two types:**

1. **Swap-in:** Swap-in means removing a program from the hard disk and putting it back in the RAM.
2. **Swap-out:** Swap-out means removing a program from the RAM and putting it into the hard disk.



### **2. Paging**

Paging is the memory management technique in which secondary memory is divided into fixed-size blocks called pages, and main memory is divided into fixed-size blocks called frames. The Frame has the same size as that of a Page. The processes are initially in secondary memory, from where the processes are shifted to main memory(RAM) when there is a requirement. Each process is mainly divided into parts where the size of each part is the same as the page size. One page of a process is mainly stored in one of the memory frames. Paging follows no contiguous memory allocation. That means pages in the main memory can be stored at different locations in the memory.



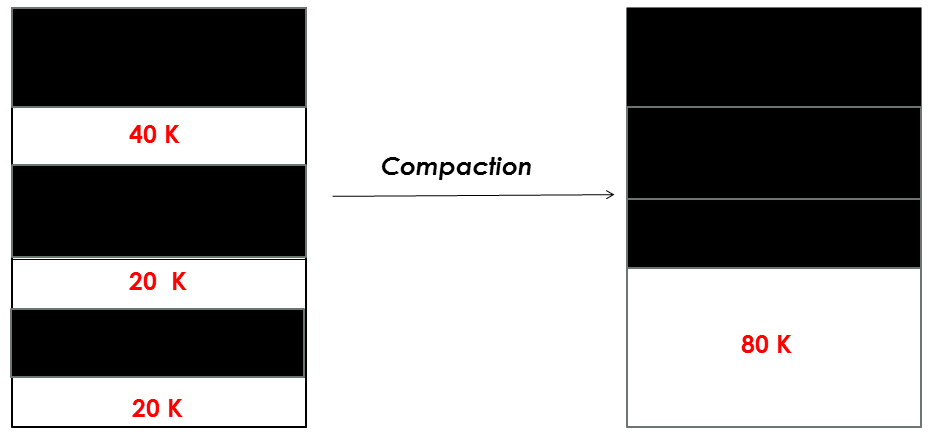
### **3. Compaction**

Compaction is a memory management technique in which the free space of a running system is compacted, to reduce fragmentation problem and improve memory allocation efficiency. Compaction is used by many modern operating systems, such as Windows, Linux, and Mac OS X. As in the fig we have some used memory(black color) and some unused memory(white color).The used memory is combined.All the empty spaces are combined together.This process is called compaction.This is done to prevent to solve the problem of fragmentation, but it requires too much of CPU time.

By compacting memory, the operating system can reduce or eliminate fragmentation and make it easier for programs to allocate and use memory.

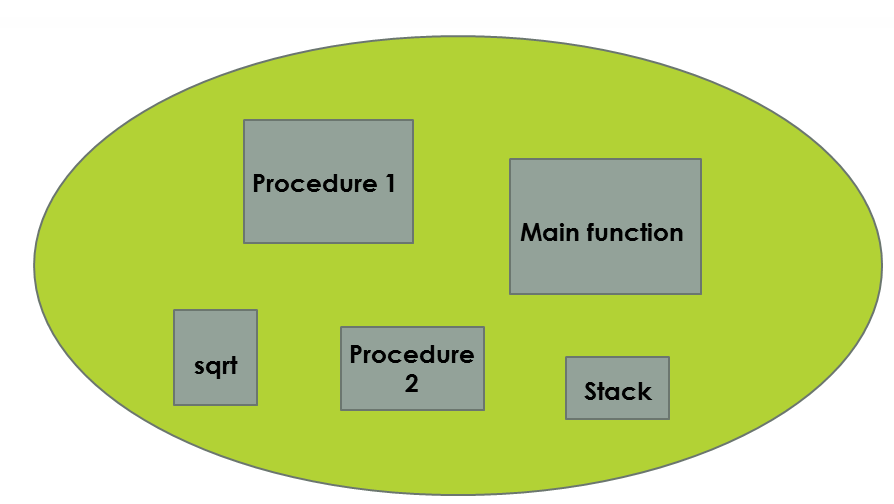
The compaction process usually consists of two steps:

1. Copying all pages that are not in use to one large contiguous area.
2. Then write the pages that are in use into the newly freed space.



### **4. Segmentation**

Segmentation is another memory management technique used by operating systems. The process is divided into segments of different sizes and then put in the main memory. The program/process is divided into modules, unlike paging, in which the process was divided into fixed-size pages or frames. The corresponding segments are loaded into the main memory when the process is executed. Segments contain the program’s utility functions, main function, and so on.



# Need for Paging

## **Disadvantage of Dynamic Partitioning**

The main disadvantage of Dynamic Partitioning is External fragmentation. Although, this can be removed by Compaction but as we have discussed earlier, the compaction makes the system inefficient.

We need to find out a mechanism which can load the processes in the partitions in a more optimal way. Let us discuss a dynamic and flexible mechanism called paging.

## **Need for Paging**

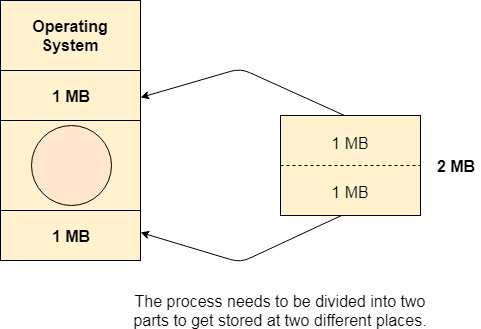
Lets consider a process P1 of size 2 MB and the main memory which is divided into three partitions. Out of the three partitions, two partitions are holes of size 1 MB each.

P1 needs 2 MB space in the main memory to be loaded. We have two holes of 1 MB each but they are not contiguous.

Although, there is 2 MB space available in the main memory in the form of those holes but that remains useless until it become contiguous. This is a serious problem to address.

We need to have some kind of mechanism which can store one process at different locations of the memory.

The Idea behind paging is to divide the process in pages so that, we can store them in the memory at different holes. We will discuss paging with the examples in the next sections.



Page Table

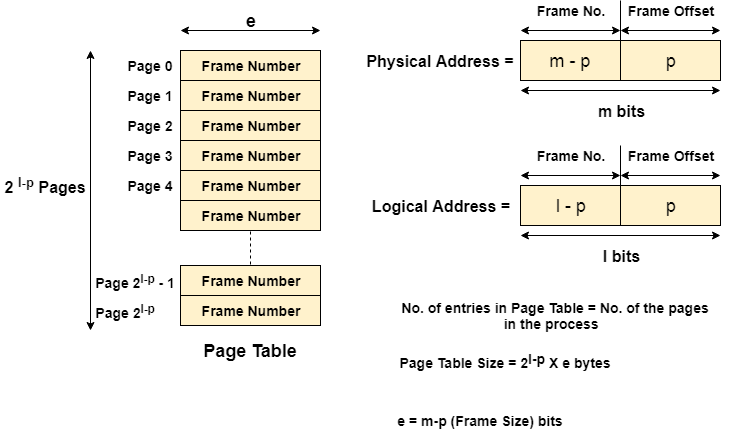
Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.

Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.

Physical addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

The image given below considers,

Physical Address Space = M words  
Logical Address Space = L words  
Page Size = P words  
  
Physical Address = log 2 M = m bits  
Logical Address = log 2 L = l bits  
page offset = log 2 P = p bits

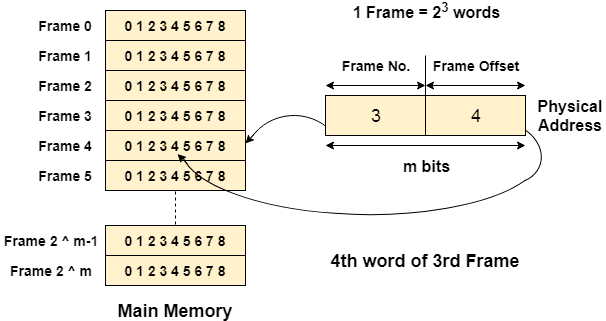
  
The CPU always accesses the processes through their logical addresses. However, the main memory recognizes physical address only.

In this situation, a unit named as Memory Management Unit comes into the picture. It converts the page number of the logical address to the frame number of the physical address. The offset remains same in both the addresses.

To perform this task, Memory Management unit needs a special kind of mapping which is done by page table. The page table stores all the Frame numbers corresponding to the page numbers of the page table.

In other words, the page table maps the page number to its actual location (frame number) in the memory.

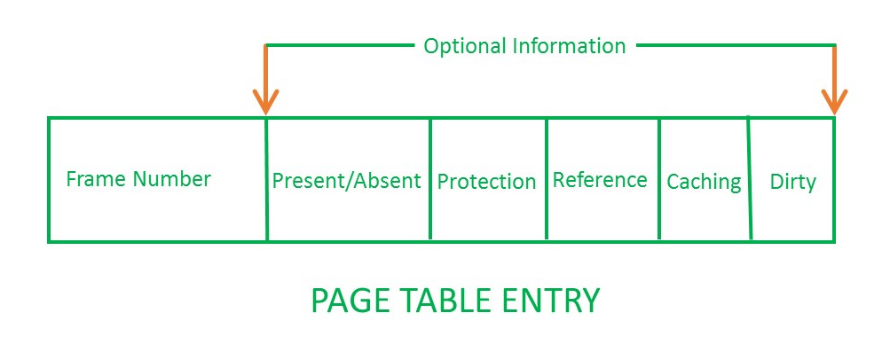
In the image given below shows, how the required word of the frame is accessed with the help of offset.



# Page Table Entries in Page Table

# Page table has page table entries where each page table entry stores a frame number and optional status (like protection) bits. Many of status bits used in the virtual memory system. The most important thing in PTE is frame Number.

**Page table entry has the following information –**



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 –** Some times 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

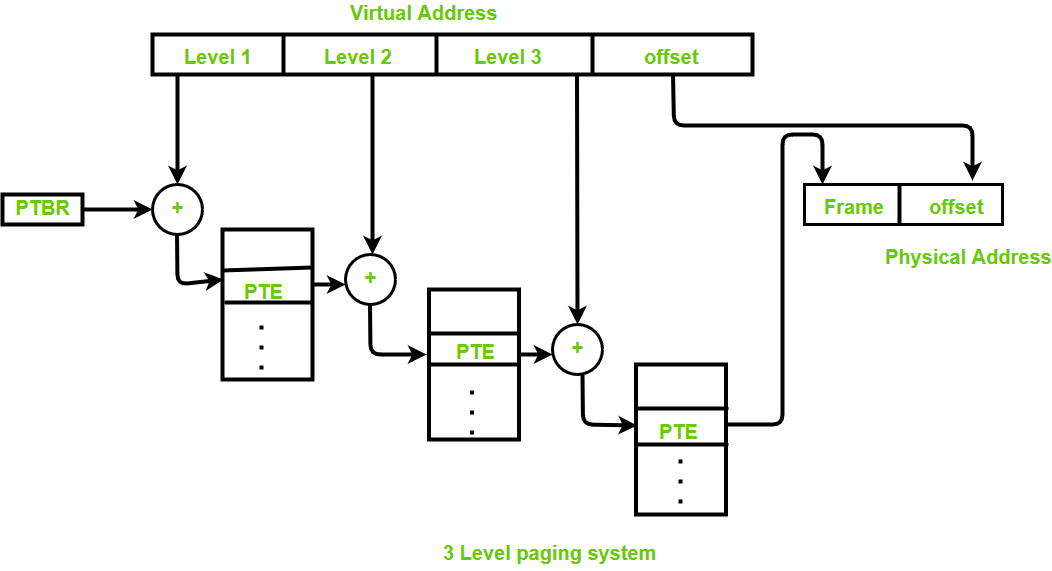
**Multilevel Paging** is a paging scheme that consists of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page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

**Why it is required?**

If the frame size of the main memory is smaller than the page size and the process cannot fit that way then we divide the pages into further pages, and this concept is known as multilevel paging.

**Virtual address:**

  
In multilevel paging whatever may be levels of paging, all the page tables will be stored in the main memory. So it requires more than one memory access to get the physical address of the page frame. One access for each level is needed. Each page table entry **except** the last level page table entry contains the base address of the next level page table.



Reference to actual page frame:

* Reference to PTE in level 1 page table = PTBR value + Level 1 offset present in virtual address.
* Reference to PTE in level 2 page table = Base address (present in Level 1 PTE) + Level 2 offset (present in VA).
* Reference to PTE in level 3 page table= Base address (present in Level 2 PTE) + Level 3 offset (present in VA).
* Actual page frame address = PTE (present in level 3).

Generally, the page table size will be equal to the size of the page.

**Assumptions:**   
Byte addressable memory and n is the number of bits used to represent virtual address.

**Important formula:** 

Number of entries in page table:

= (virtual address space size) / (page size)

= Number of pages

Virtual address space size:

= 2n B

Size of page table:

<>= (number of entries in page table)\*(size of PTE)

If page table size > desired size then create 1 more level.

**Disadvantage:**   
Extra memory references to access address translation tables can slow programs down by a factor of two or more. Use translation look aside buffer (TLB) to speed up address translation by storing page table entries.

**Example:**   
Q.Consider a virtual memory system with physical memory of 8GB, a page size of 8KB, and 46-bit virtual address. *Assume every page table exactly fits into a single page*. If page table entry size is 4B then how many levels of page tables would be required.

**Explanation:**

Page size = 8KB = 213 B

Virtual address space size = 246 B

PTE = 4B = 22 B

Number of pages or number of entries in page table,

= (virtual address space size) / (page size)

= 246B/213 B

= 233

Size of page table,

= (number of entries in page table)\*(size of PTE)

= 233\*22 B

= 235 B

To create one more level,

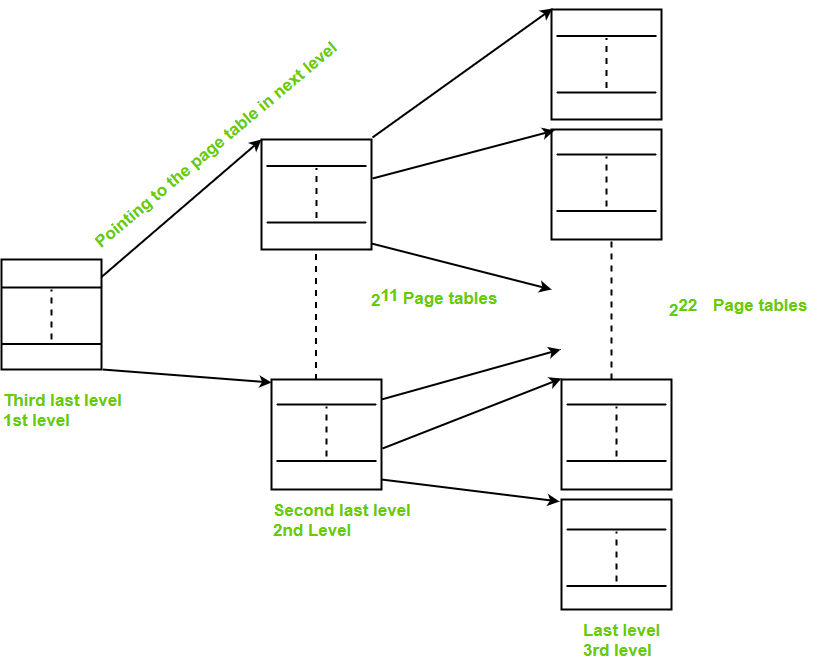
Size of page table > page size

Number of page tables in last level,

= 235 B / 213 B

= 222

The base address of these tables is stored in page table [second last level].



Size of page table [second last level]

= 222\*22B

= 224B

To create one more level,   
Size of page table [second last level] > page size

Number of page tables in second last level

= 224B/213 B

= 211

The base address of these tables are stored in page table [third last level]

Size of page table [third last level]

= 211\*22 B

= 213 B

= page size

**∴ 3 levels are required.**

# Inverted Page Table in Operating System

Most Operating Systems implement a separate page table for each process, i.e. for ‘n’ number of processes running on a Multiprocessing/ Timesharing operating system, there is ‘n’ number of page tables stored in the memory. Sometimes when a process is very large in size and it occupies virtual memory then with the size of the process, its page table size also increases substantially.

**Example:** A process of size 2 GB with:

Page size = 512 Bytes

Size of page table entry = 4 Bytes, then

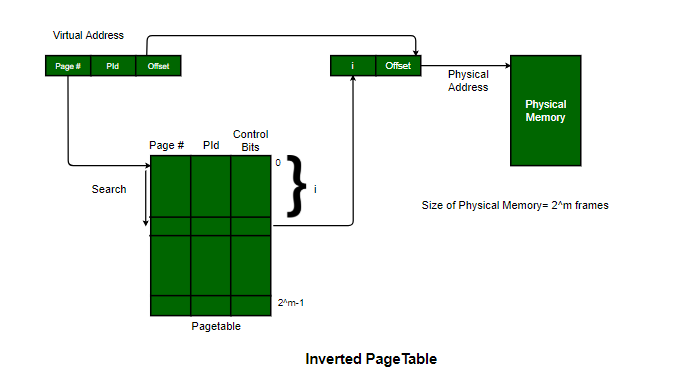
Number of pages in the process = 2 GB / 512 B = 222

PageTable Size = 222 \* 22 = 224 bytes

Through this example, it can be concluded that for multiple processes running simultaneously in an OS, a considerable part of memory is occupied by page tables only. Operating Systems also incorporate **multilevel paging schemes** which further increase the space required for storing the page tables and a large amount of memory is invested in storing them. The amount of memory occupied by the page tables can turn out to be a huge overhead and is always unacceptable as main memory is always a scarce resource. Various efforts are made to utilize the memory efficiently and to maintain a good balance in the level of multiprogramming and efficient CPU utilization.

**Inverted Page Table –** An alternate approach is to use the **Inverted Page Table** structure that consists of a 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. Through the inverted page table, the overhead of storing an individual page table for every process gets eliminated and only a fixed portion of memory is required to store the paging information of all the processes together. This technique is called inverted paging as the indexing is done with respect to the frame number instead of the logical page number. Each entry in the page table contains the following fields.

* **Page number –** It specifies the page number range of the logical address.
* **Process id –** An inverted page table contains the address space information of all the processes in execution. Since two different processes can have a similar set of virtual addresses, it becomes necessary in the Inverted Page Table to store a process Id of each process to identify its address space uniquely. This is done by using the combination of PId and Page Number. So this Process Id acts as an address space identifier and ensures that a virtual page for a particular process is mapped correctly to the corresponding physical frame.
* **Control bits –** These bits are used to store extra paging-related information. These include the valid bit, dirty bit, reference bits, protection, and locking information bits.
* **Chained pointer –** It may be possible sometimes that two or more processes share a part of the main memory. In this case, two or more logical pages map to the same Page Table Entry then a chaining pointer is used to map the details of these logical pages to the root page table.

**Working –** The operation of an inverted page table is shownbelow. 

 The virtual address generated by the CPU contains the fields and each page table entry contains the other relevant information required in paging related mechanism. When a memory reference takes place, this virtual address is matched by the Memory-Mapping unit(MMU), the Inverted Page table is searched and the corresponding frame number is obtained. If the match is found at the ith entry then the physical address of the process *is sent as the real address otherwise if no match is found then Segmentation Fault is generated.* **Note: Number of Entries in Inverted page table = Number of frames in Physical address Space(PAS).**

**Examples –** The Inverted Page table and its variations are implemented in various systems like PowerPC, UltraSPARC, and the IA-64 architecture. An implementation of the Mach operating system on the RT-PC also uses this technique.

**Advantages and Disadvantages:**

* **Reduced memory space –** Inverted Pagetables typically reduce 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.
* **Difficult shared memory implementation –** As the Inverted Page Table stores a single entry for each frame, it becomes difficult to implement the shared memory in the page tables. Chaining techniques are used to map more than one virtual address to the entry specified in the order of frame number.

# What is Thrash?

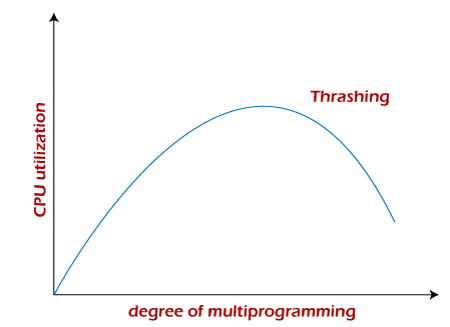
In computer science, **thrash** is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory. Depending on the configuration and algorithm, the actual throughput of a system can degrade by multiple orders of magnitude.

In computer science, **thrashing** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

To know more clearly about thrashing, first, we need to know about page fault and swapping.

* **Page fault:** We know every program is divided into some pages. A page fault occurs when a program attempts to access data or code in its address space but is not currently located in the system RAM.
* **Swapping:** Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This process is called swapping.

**Thrashing** is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages. This state in the operating system is known as thrashing. Because of thrashing, the CPU utilization is going to be reduced or negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no valuable work would be done by the CPU, and the CPU utilization would fall drastically.

The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory, thereby increasing the degree of multiprogramming. Unfortunately, this would result in a further decrease in the CPU utilization, triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called thrashing.

### **Algorithms during Thrashing**

Whenever thrashing starts, the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

**1. Global Page Replacement**

Since global page replacement can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more. Therefore, the global page replacement algorithm is not suitable when thrashing happens.

**2. Local Page Replacement**

Unlike the global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. Therefore, local page replacement is just an alternative to global page replacement in a thrashing scenario.

### **Causes of Thrashing**

Programs or workloads may cause thrashing, and it results in severe performance problems, such as:

* 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.
* So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

### **How to Eliminate Thrashing**

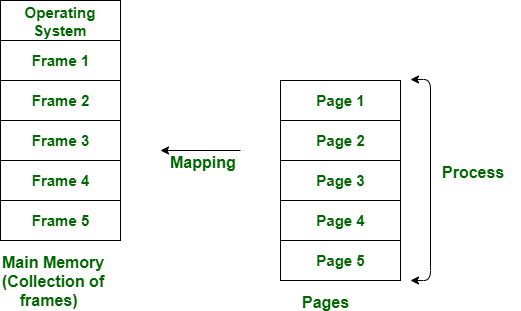
Thrashing has some negative impacts on hard drive health and system performance. Therefore, it is necessary to take some actions to avoid it. To resolve the problem of thrashing, here are the following methods, such as:

* **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.

# Difference Between Paging and Segmentation

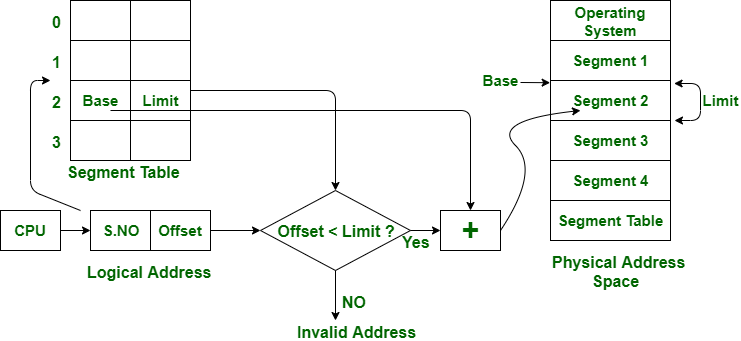
[**Paging**](https://www.geeksforgeeks.org/paging-in-operating-system/)**:**   
Paging is a method or technique which is used for non-contiguous memory allocation. It is a fixed-size partitioning theme (scheme). In paging, both main memory and secondary memory are divided into equal fixed-size partitions. The partitions of the secondary memory area unit and main memory area unit are known as pages and frames respectively.

Paging is a memory management method accustomed fetch processes from the secondary memory into the main memory in the form of pages. in paging, each process is split into parts wherever the size of every part is the same as the page size. The size of the last half could also be but the page size. The pages of the process area unit hold on within the frames of main memory relying upon their accessibility.



[**Segmentation**](https://www.geeksforgeeks.org/segmentation-in-operating-system/)**:**   
Segmentation is another non-contiguous memory allocation scheme like paging. like paging, in segmentation, the process isn’t divided indiscriminately into mounted(fixed) size pages. It is a variable-size partitioning theme. like paging, in segmentation, secondary and main memory are not divided into partitions of equal size. The partitions of secondary memory area units are known as segments. The details concerning every segment are hold in a table known as segmentation table. Segment table contains two main data concerning segment, one is Base, which is the bottom address of the segment and another is Limit, which is the length of the segment.

In segmentation, the CPU generates a logical address that contains the Segment number and segment offset. If the segment offset is a smaller amount than the limit then the address called valid address otherwise it throws miscalculation because the address is invalid.



The above figure shows the translation of a logical address to a physical address.

| S.NO | Paging | Segmentation |
| --- | --- | --- |
| 1. | In paging, the program is divided into fixed or mounted size pages. | In segmentation, the program is divided into variable size sections. |
| 2. | For the paging operating system is accountable. | For segmentation compiler is accountable. |
| 3. | Page size is determined by hardware. | Here, the section size is given by the user. |
| 4. | It is faster in comparison to segmentation. | Segmentation is slow. |
| 5. | Paging could result in internal fragmentation. | Segmentation could result in external fragmentation. |
| 6. | In paging, the logical address is split into a page number and page offset. | Here, the logical address is split into section number and section offset. |
| 7. | Paging comprises a page table that encloses the base address of every page. | While segmentation also comprises the segment table which encloses segment number and segment offset. |
| 8. | The page table is employed to keep up the page data. | Section Table maintains the section data. |
| 9. | In paging, the operating system must maintain a free frame list. | In segmentation, the operating system maintains a list of holes in the main memory. |
| 10. | Paging is invisible to the user. | Segmentation is visible to the user. |
| 11. | In paging, the processor needs the page number, and offset to calculate the absolute address. | In segmentation, the processor uses segment number, and offset to calculate the full address. |
| 12. | It is hard to allow sharing of procedures between processes. | Facilitates sharing of procedures between the processes. |
| 13 | In paging, a programmer cannot efficiently handle data structure. | It can efficiently handle data structures. |
| 14. | This protection is hard to apply. | Easy to apply for protection in segmentation. |
| 15. | The size of the page needs always be equal to the size of frames. | There is no constraint on the size of segments. |
| 16. | A page is referred to as a physical unit of information. | A segment is referred to as a logical unit of information. |
| 17. | Paging results in a less efficient system. | Segmentation results in a more efficient system. |

# 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, earlier people have 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.   
Formally,   
“The process of **transferring a block** of program code or other data into internal memory, replacing what is already stored”.   
Sometimes it happens that compare to the size of the biggest partition, the size of the program will be even more, then, in that case, you should go with overlays.

So overlay is a technique to run a program that is bigger than the size of the physical memory by keeping only those instructions and data that are needed at any given time.Divide the program into modules in such a way that not all modules need to be in the memory at the same time.

**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

**Example –**   
The best example of overlays is assembler. Consider the assembler has 2 passes, 2 pass means at any time it will be doing only one thing, either the 1st pass or the 2nd pass. This means it will finish 1st pass first and then 2nd pass.Let assume that available main memory size is 150KB and total code size is 200KB

Pass 1.......................70KB

Pass 2.......................80KB

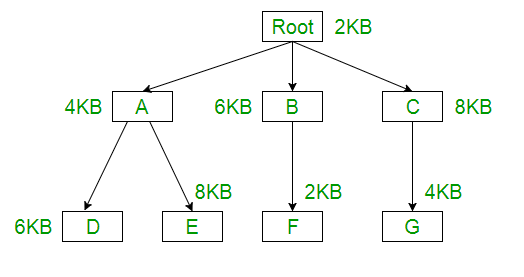
Symbol table.................30KB

Common routine...............20KB

As the total code size is 200KB and main memory size is 150KB, it is not possible to use 2 passes together.So, in this case, we should go with the overlays technique.According to the overlays concept at any time only one pass will be used and both the passes always need symbol table and common routine.Now the question is if overlays-driver\* is 10KB, then what is the minimum partition size required?For pass 1 total memory needed is = (70KB + 30KB + 20KB + 10KB) = 130KB and for pass 2 total memory needed is = (80KB + 30KB + 20KB + 10KB) = 140KB.So if we have minimum 140KB size partition then we can run this code very easily.

\*Overlays driver:-It is the user responsibility to take care of overlaying, the operating system will not provide anything.Which means the user should write even what part is required in the 1st pass and once the 1st pass is over, the user should write the code to pull out the pass 1 and load the pass 2.That is what is the responsibility of the user, that is known as the Overlays driver.Overlays driver will just help us to move out and move in the various part of the code.

**Question –**   
The overlay tree for a program is as shown below: 



What will be the size of the partition (in physical memory) required to load (and   
run) this program?   
(a) 12 KB (b) 14 KB (c) 10 KB (d) 8 KB

**Explanation –**   
Using the overlay concept we need not actually have the entire program inside the main memory.Only we need to have the part which are required at that instance of time, either we need Root-A-D or Root-A-E or Root-B-F or Root-C-G part.

Root+A+D = 2KB + 4KB + 6KB = 12KB

Root+A+E = 2KB + 4KB + 8KB = 14KB

Root+B+F = 2KB + 6KB + 2KB = 10KB

Root+C+G = 2KB + 8KB + 4KB = 14KB

So if we have 14KB size of partition then we can run any of them.   
Answer -(b) 14KB

# Virtual Memory in Operating System

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage locations.

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.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that 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.
2. 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.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. 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.   
The process includes the following steps :

virtual_mem

1. 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.
2. 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.
3. The OS will search for the required page in the logical address space.
4. 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.
5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

**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 may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. 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.

**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 six steps.

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is : p

Then, Effective memory access time = (p\*s) + (1-p)\*m

**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. When a process is busy swapping pages in and out then this situation is called thrashing.

swaping

**Thrashing :**

virtual_mem_2

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. However, the OS must be clever about how it manages this scheme. 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 system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

In the given diagram, the initial degree of multiprogramming up to some extent of point(lambda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multiprogramming the CPU utilization will drastically fall down and the system will spend more time only on the page replacement and the time is taken to complete the execution of the process will increase. This situation in the system is called thrashing.

**Causes of Thrashing :**

1. **High degree of multiprogramming**: If the number of processes keeps on increasing in the memory then the number of frames allocated to each process will be decreased. So, fewer frames will be available for each process. Due to this, a page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.

For example:   
Let free frames = 400   
**Case 1**: Number of process = 100   
Then, each process will get 4 frames.

**Case 2**: Number of processes = 400   
Each process will get 1 frame.   
Case 2 is a condition of thrashing, as the number of processes is increased, frames per process are decreased. Hence CPU time will be consumed in just swapping pages. 

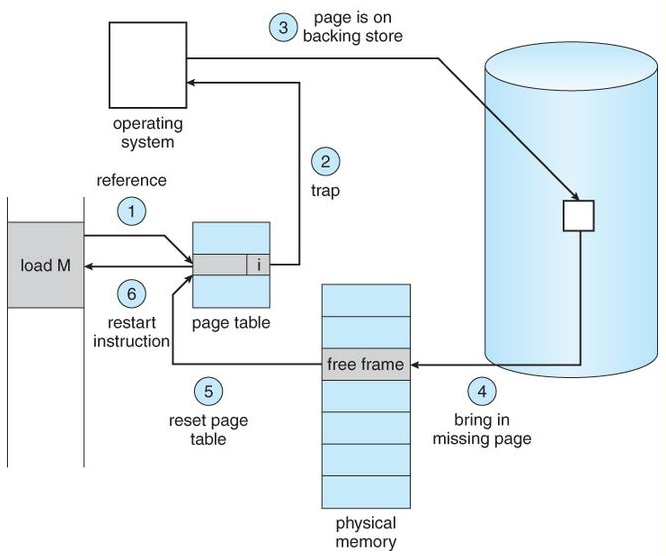
1. **Lacks of Frames**: If a process has fewer frames then fewer pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

**Recovery of Thrashing :**

* Do not allow the system to go into thrashing by instructing the long-term scheduler not to bring the processes into memory after the threshold.
* If the system is already thrashing then instruct the mid-term scheduler to suspend some of the processes so that we can recover the system from thrashing.

# Page Fault Handling in Operating System

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :



* The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
* An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
* Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Some times hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
* Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
* If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
* If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
* As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
* When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
* Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
* Assembly Routine reloads register and other state information, returns to user space to continue execution.

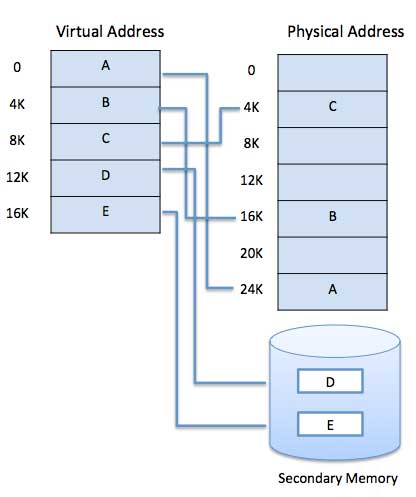
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

* User written error handling routines are used only when an error occurred in the data or computation.
* Certain options and features of a program may be used rarely.
* Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
* The ability to execute a program that is only partially in memory would counter many benefits.
* Less number of I/O would be needed to load or swap each user program into memory.
* A program would no longer be constrained by the amount of physical memory that is available.
* Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

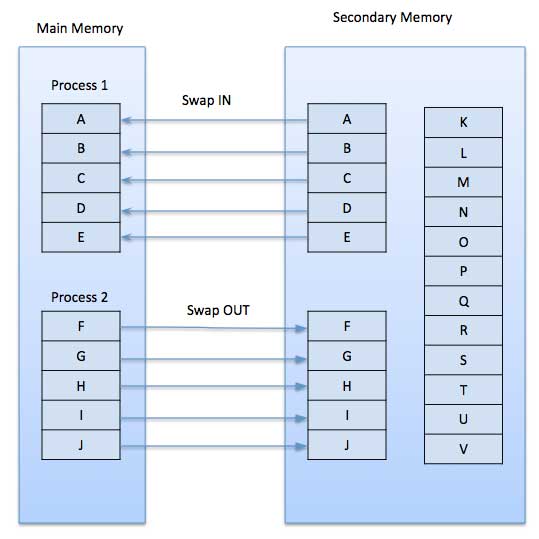
Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

## **Demand Paging**

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program’s pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

### **Advantages**

Following are the advantages of Demand Paging −

* Large virtual memory.
* More efficient use of memory.
* There is no limit on degree of multiprogramming.

### **Disadvantages**

* Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

# Translation Lookaside Buffer (TLB) in Paging

In Operating System (Memory Management Technique : [Paging](https://www.geeksforgeeks.org/operating-system-paging/)), 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.

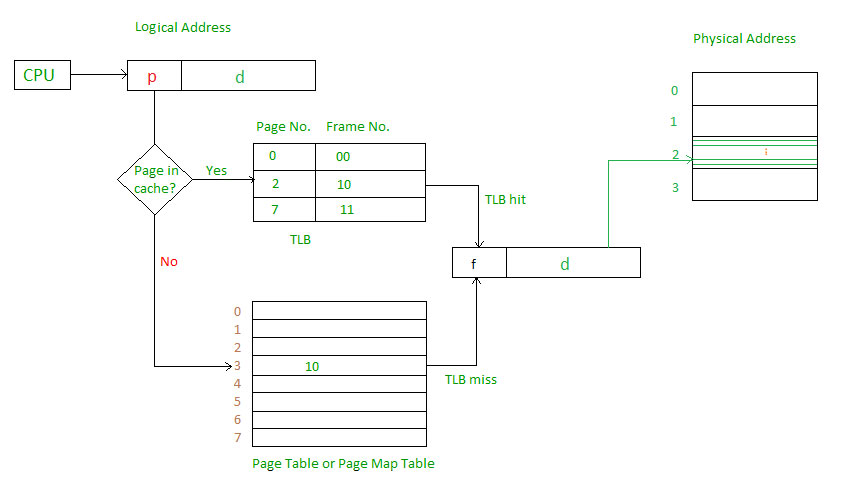
The problem initially was to fast access the main memory content based on address generated by CPU (i.e [logical/virtual address](https://www.geeksforgeeks.org/logical-vs-physical-address-in-operating-system/)). Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less.

The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (lets say this page table contains 1M entries), so registers may not hold all the PTE’s of Page table. So this is not a practical approach.

To overcome this size issue, the entire page table was kept in main memory. but the problem here is two main memory references are required: 

1. To find the frame number
2. To go to the address specified by frame number

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. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



**Steps in TLB hit:** 

1. CPU generates virtual (logical) address.
2. It is checked in TLB (present).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.

**Steps in TLB miss:** 

1. CPU generates virtual (logical) address.
2. It is checked in TLB (not present).
3. Now the page number is matched to page table residing in main memory (assuming page table contains all PTE).
4. Corresponding frame number is retrieved, which now tells where in the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e either FIFO, LRU or MFU etc).

**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 Algorithm**

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

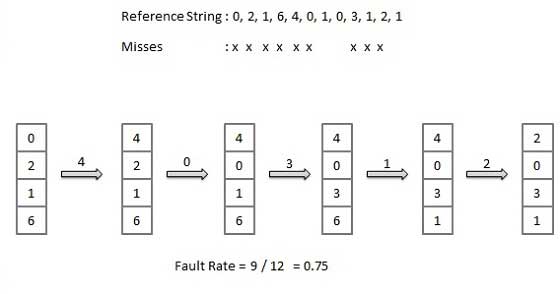
## **Reference String**

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

* For a given page size, we need to consider only the page number, not the entire address.
* If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
* For example, consider the following sequence of addresses − 123,215,600,1234,76,96
* If page size is 100, then the reference string is 1,2,6,12,0,0

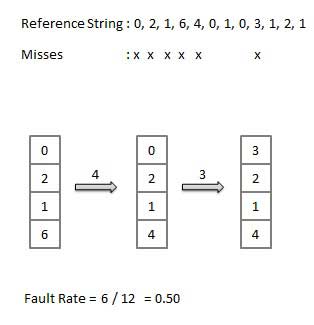
## **First In First Out (FIFO) algorithm**

* Oldest page in main memory is the one which will be selected for replacement.
* Easy to implement, keep a list, replace pages from the tail and add new pages at the head.



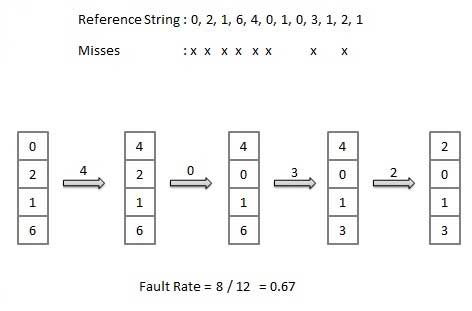
## **Optimal Page algorithm**

* An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
* Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



## **Least Recently Used (LRU) algorithm**

* Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
* Easy to implement, keep a list, replace pages by looking back into time.



## **Page Buffering algorithm**

* To get a process start quickly, keep a pool of free frames.
* On page fault, select a page to be replaced.
* Write the new page in the frame of free pool, mark the page table and restart the process.
* Now write the dirty page out of disk and place the frame holding replaced page in free pool.

## **Least frequently Used(LFU) algorithm**

* The page with the smallest count is the one which will be selected for replacement.
* This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

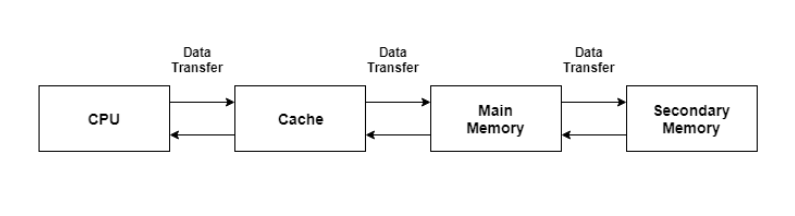
## **Most frequently Used(MFU) algorithm**

* This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

# Cache Management

Cache is a type of memory that is used to increase the speed of data access. Normally, the data required for any process resides in the main memory. However, it is transferred to the cache memory temporarily if it is used frequently enough.

A diagram to better understand the data transfer in cache management is as follows −



## **Cache Performance**

The cache performance can be explained using the following steps −

* If a process needs some data, it first searches in the cache memory. If the data is available in the cache, this is termed as a cache hit and the data is accessed as required.
* If the data is not in the cache then it is termed as a cache miss. Then the data is obtained from the main memory. After that the data is transferred to the cache memory under the assumption that it will be needed again.
* The performance of the cache is measured using the hit ratio. It is the number of cache hits divided by the total cache accesses. The formula for this is:



## **Types of Cache Memory**

There are mainly two types of cache memory i.e. primary cache and secondary cache. These are explained in detail as follows −

### **Primary Cache**

Primary cache is very fast and its access time is similar to the processor registers. This is because it is built onto the processor chip. However because of this reason, its size is quite small. It is also known as a level 1 cache and is build using static RAM (SRAM).

### **Secondary Cache**

The secondary cache or external cache is cache memory that is external to the primary cache. It is located between the primary cache and the main memory. It is also known as a level 2 cache and is often housed on the processor chip as well.

### **Advantages of Cache Memory**

Some of the advantages of cache memory are as follows −

* Cache memory is faster than main memory as it is located on the processor chip itself. Its speed is comparable to the processor registers and so frequently required data is stored in the cache memory.
* The memory access time is considerably less for cache memory as it is quite fast. This leads to faster execution of any process.
* The cache memory can store data temporarily as long as it is frequently required. After the use of any data has ended, it can be removed from the cache and replaced by new data from the main memory.

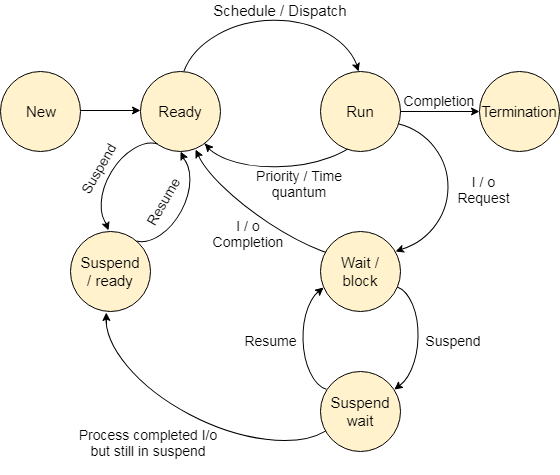
### **Disadvantages of Cache Memory**

Some of the disadvantages of cache memory are as follows −

* Since the cache memory is quite fast, it is extremely useful in any computer system. However, it is also quite expensive and so is used judiciously.
* The cache is memory expensive as observed from the previous point. Also, it is located directly on the processor chip. Because of these reasons, it has a limited capacity and is much smaller than main memory.

Process States

**State Diagram**



* **New (Create) –** In this step, the process is about to be created but not yet created, it is the program which 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 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 wait 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 is deleted.
* **Suspend ready –** Process that was initially in the ready state but was swapped out of main memory(refer Virtual Memory topic) and placed onto external storage by scheduler is said to be in suspend ready state. The process will transition back to 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.

**CPU and I/O Bound Processes:** If the process is intensive in terms of CPU operations then it is called CPU bound process. Similarly, If the process is intensive in terms of I/O operations then it is called I/O bound process. **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.

**Degree of multiprogramming –** The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming, e.g., if the degree of programming = 100, this means 100 processes can reside in the ready state at maximum.

## **Operations on the Process**

### **1. Creation**

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

### **2. Scheduling**

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

### **3. Execution**

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

### **4. Deletion/killing**

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system

# Process Scheduling

## **Definition**

The 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.

Process scheduling is an essential part of a Multiprogramming operating systems. 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.

## **Categories of Scheduling**

There are two categories of scheduling:

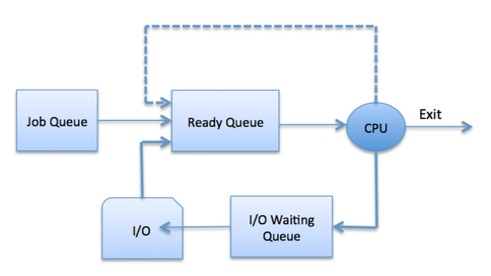
1. **Non-preemptive:** Here the resource can’t be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
2. **Preemptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

## **Process Scheduling Queues**

The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

* **Job queue** − This queue keeps all the processes in the system.
* **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
* **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

## **Two-State Process Model**

Two-state process model refers to running and non-running states which are described below −

|  |  |
| --- | --- |
| **S.N.** | **State & Description** |
| 1 | **Running**  When a new process is created, it enters into the system as in the running state. |
| 2 | **Not Running**  Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute. |

## **Schedulers**

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

* Long-Term Scheduler
* Short-Term Scheduler
* Medium-Term Scheduler

## **Long Term Scheduler**

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

## **Short Term Scheduler**

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

## **Medium Term Scheduler**

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

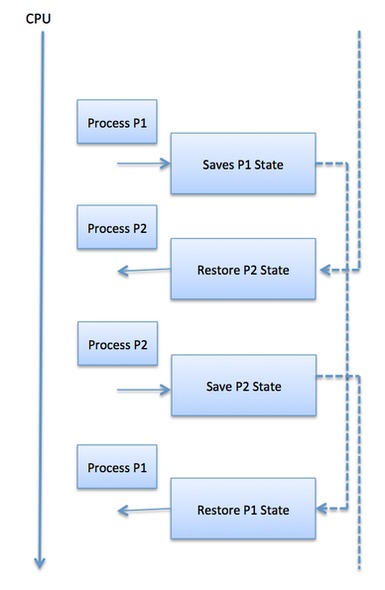
## **Comparison among Scheduler**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Long-Term Scheduler** | **Short-Term Scheduler** | **Medium-Term Scheduler** |
| 1 | It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler. |
| 2 | Speed is lesser than short term scheduler | Speed is fastest among other two | Speed is in between both short and long term scheduler. |
| 3 | It controls the degree of multiprogramming | It provides lesser control over degree of multiprogramming | It reduces the degree of multiprogramming. |
| 4 | It is almost absent or minimal in time sharing system | It is also minimal in time sharing system | It is a part of Time sharing systems. |
| 5 | It selects processes from pool and loads them into memory for execution | It selects those processes which are ready to execute | It can re-introduce the process into memory and execution can be continued. |

## **Context Switching**

A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

* Program Counter
* Scheduling information
* Base and limit register value
* Currently used register
* Changed State
* I/O State information
* Accounting information

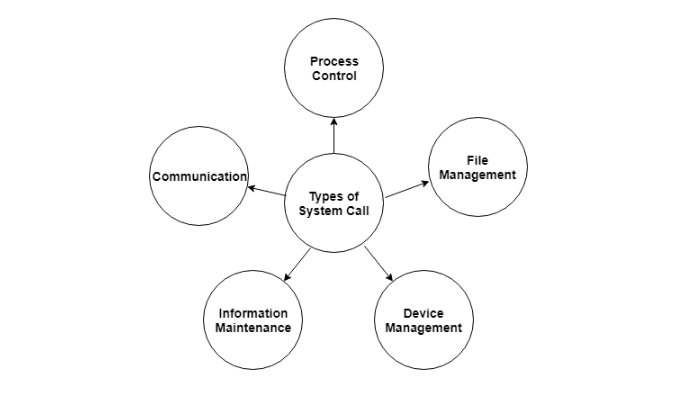
# Different types of system calls

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

## **Types of System Calls**

There are mainly five types of system calls. These are explained in detail as follows −



Here are the types of system calls −

### **Process Control**

These system calls deal with processes such as process creation, process termination etc.

### **File Management**

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

### **Device Management**

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

### **Information Maintenance**

These system calls handle information and its transfer between the operating system and the user program.

### **Communication**

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

## **Examples of Windows and Unix system calls**

There are various examples of Windows and Unix system calls. These are as listed below in the table:

|  |  |  |
| --- | --- | --- |
| **Process** | **Windows** | **Unix** |
| **Process Control** | CreateProcess() ExitProcess() WaitForSingleObject() | Fork() Exit() Wait() |
| **File Manipulation** | CreateFile() ReadFile() WriteFile() CloseHandle() | Open() Read() Write() Close() |
| **Device Management** | SetConsoleMode() ReadConsole() WriteConsole() | Ioctl() Read() Write() |
| **Information Maintenance** | GetCurrentProcessID() SetTimer() Sleep() | Getpid() Alarm() Sleep() |
| **Communication** | CreatePipe() CreateFileMapping() MapViewOfFile() | Pipe() Shmget() Mmap() |
| **Protection** | SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorgroup() | Chmod() Umask() Chown() |

There are many different system calls as shown above. Details of some of those system calls are as follows −

## **wait()**

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

## **exec()**

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.

## **fork()**

Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

## **exit()**

The exit() system call is used by a program to terminate its execution. In a multithreaded environment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.

## **kill()**

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit.However, kill system call does not necessary mean killing the process and can have various meanings.

### **open()**

The **open()** system call allows you to access a file on a file system. It allocates resources to the file and provides a handle that the process may refer to. Many processes can open a file at once or by a single process only. It's all based on the file system and structure.

### **read()**

It is used to obtain data from a file on the file system. It accepts three arguments in general:

* A file descriptor.
* A buffer to store read data.
* The number of bytes to read from the file.

The file descriptor of the file to be read could be used to identify it and open it using **open()** before reading.

### **wait()**

In some systems, a process may have to wait for another process to complete its execution before proceeding. When a parent process makes a child process, the parent process execution is suspended until the child process is finished. The **wait()** system call is used to suspend the parent process. Once the child process has completed its execution, control is returned to the parent process.

### **write()**

It is used to write data from a user buffer to a device like a file. This system call is one way for a program to generate data. It takes three arguments in general:

* A file descriptor.
* A pointer to the buffer in which data is saved.
* The number of bytes to be written from the buffer.

### **fork()**

Processes generate clones of themselves using the **fork()** system call. It is one of the most common ways to create processes in operating systems. When a parent process spawns a child process, execution of the parent process is interrupted until the child process completes. Once the child process has completed its execution, control is returned to the parent process.

### **close()**

It is used to end file system access. When this system call is invoked, it signifies that the program no longer requires the file, and the buffers are flushed, the file information is altered, and the file resources are de-allocated as a result.

### **exec()**

When an executable file replaces an earlier executable file in an already executing process, this system function is invoked. As a new process is not built, the old process identification stays, but the new process replaces data, stack, data, head, etc.

### **exit()**

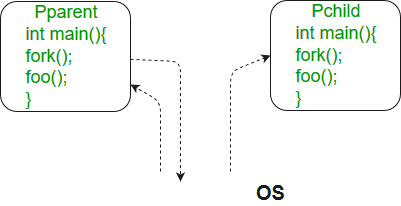
The **exit()** is a system call that is used to end program execution. This call indicates that the thread execution is complete, which is especially useful in multi-threaded environments. The operating system reclaims resources spent by the process following the use of the **exit()** system function.

# fork()

Fork system call is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

***Negative Value***: creation of a child process was unsuccessful.  
***Zero***: Returned to the newly created child process.  
***Positive value***: Returned to parent or caller. The value contains process ID of newly created child process.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/Fork_in_C.jpg)

*Note: fork() is threading based function, to get the correct output run the program on a local system.*

# Dual Mode Operations in Operating System

The dual-mode operations in the operating system protect the operating system from illegal users. We accomplish this defense by designating some of the system instructions as privileged instructions that can cause harm. The hardware only allows for the execution of privileged instructions in kernel mode. An example of a privileged instruction is the command to switch to user mode. Other examples include monitoring of I/O, controlling timers and handling interruptions.

To ensure proper operating system execution, we must differentiate between machine code execution and user-defined code. Most computer systems have embraced offering hardware support that helps distinguish between different execution modes. We have two modes of the operating system: **user mode** and **kernel mode**.

Mode bit is required to identify in which particular mode the current instruction is executing. If the mode bit is 1, it operates user mode, and if the mode bit is 0, it operates in kernel mode.

#### **NOTE: At the booting time of the system, it always starts with the kernel mode.**

### **Types of Dual Mode in Operating System**

The operating system has two modes of operation to ensure it works correctly: user mode and kernel mode.

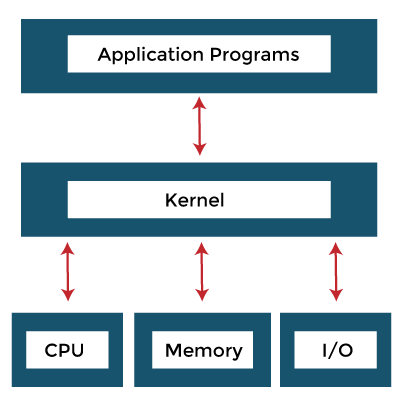
**1. User Mode**

When the computer system runs user applications like file creation or any other application program in the User Mode, this mode does not have direct access to the computer's hardware. For performing hardware related tasks, like when the user application requests for a service from the operating system or some interrupt occurs, in these cases, the system must switch to the Kernel Mode. The mode bit of the user mode is 1. This means that if the mode bit of the system's processor is 1, then the system will be in the User Mode.



**2. Kernel Mode**

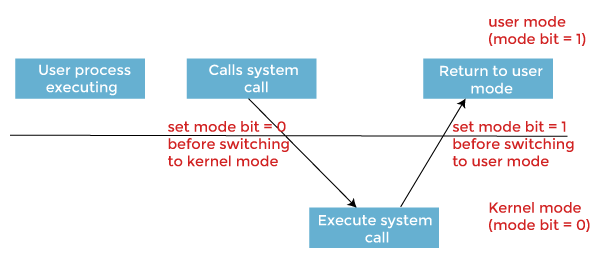
All the bottom level tasks of the Operating system are performed in the Kernel Mode. As the Kernel space has direct access to the hardware of the system, so the kernel-mode handles all the processes which require hardware support. Apart from this, the main functionality of the Kernel Mode is to execute privileged instructions.



These privileged instructions are not provided with user access, and that's why these instructions cannot be processed in the User mode. So, all the processes and instructions that the user is restricted to interfere with are executed in the Kernel Mode of the Operating System. The mode bit for the Kernel Mode is 0. So, for the system to function in the Kernel Mode, the Mode bit of the processor must be equal to 0.

### **Example**

With the mode bit, we can distinguish between a task executed on behalf of the operating system and one executed on behalf of the user.



* When the computer system executes on behalf of a user application, the system is in **user mode**.
* However, when a user application requests a service from the operating system via a system call, it must transition from **user** to **kernel mode** to fulfill the request. As we can say, this architectural enhancement is useful for many other aspects of system operation.
* At system boot time, the hardware starts in **kernel mode**.
* The operating system is then loaded and starts user applications in **user mode**.
* Whenever a trap or interrupt occurs, the hardware switches from **user mode** to **kernel mode**, changing the mode bit's state to 0.
* Thus, whenever the operating system gains control of the computer, it is in **kernel mode**.
* The system always **switches to user mode**by setting the mode bit to 1 before passing control to a user program.

### **Need for Dual-Mode Operations**

Certain types of processes are to be made hidden from the user, and certain tasks that do not require any type of hardware support. Using the **dual mode** of the OS, these tasks can be deal with separately.

Also, the Operating System needs to function in the **dual mode** because the Kernel Level programs perform all the bottom level functions of the OS like process management, Memory management, etc. If the user alters these, then this can cause an entire system failure. So, for specifying the access to the users only to the tasks of their use, Dual Mode is necessary for an Operating system.

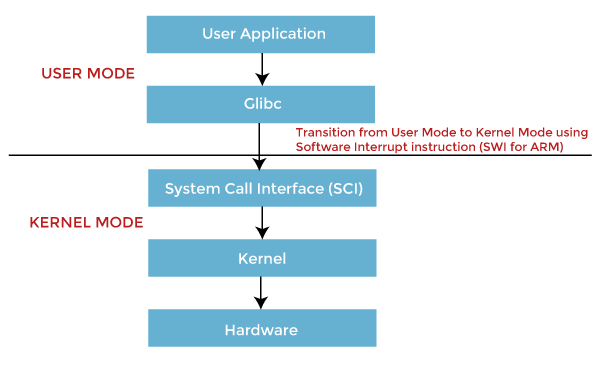
So, whenever the system works on the user applications, it is in the User mode. Whenever the user requests some hardware services, a transition from User mode to Kernel mode occurs, and this is done by changing the mode bit from 1 to 0. And for returning back into the User mode, the mode bit is again changed to 1.

### **User Mode and Kernel Mode Switching**

In its life span, a process executes in user mode and kernel mode. The user mode is a normal mode where the process has limited access. However, the kernel-mode is the privileged mode where the process has unrestricted access to system resources like hardware, memory, etc. A process can access services like hardware I/O by executing accessing kernel data in kernel mode. Anything related to process management, I/O hardware management, and memory management requires a process to execute in Kernel mode.

This is important to know that a process in Kernel mode get power to access any device and memory, and same time any crash in kernel mode brings down the whole system. But any crash in user mode brings down the faulty process only.

The kernel provides System Call Interface (SCI), which are entry points for user processes to enter kernel mode. System calls are the only way through which a process can go into kernel mode from user mode. The below diagram explains user mode to kernel mode switching in detail.



* When in user mode, the application process makes a call to **Glibc**, which is a library used by software programmers.
* Glibc library knows the proper way of calling System Call for different architectures. It set up passing arguments as per architecture's Application Binary Interface (ABI) to prepare for System Call entry.
* Now Glibc calls Software Interrupt instruction for ARM, which puts the processor into Supervisor mode by updating Mode bits of CPSR register and jumps to vector address 0x08.
* Till now, process execution was in User mode. After SWI instruction execution, the process is allowed to execute kernel code. Memory Management Unit (MMU) will now allow kernel Virtual memory access and execution for this process.
* From Vector address 0x08, process execution loads and jumps to SW Interrupt handler routine, **vector\_swi()**for ARM.
* In **vector\_swi()**, System Call Number (SCNO) is extracted from SWI instruction, and execution jumps to system call function using SCNO as an index in system call table **sys\_call\_table**.
* After System Call execution, in the return path, userspace registers are restored before starting execution in User Mode.

### **Why do we need Switching?**

There are two main reasons behind the switching between User mode and kernel mode, such as:

1. If everything were to run in a single-mode, we would end up with Microsoft's issue in the earlier versions of Windows. If a process were able to exploit a vulnerability, that process then could control the system.
2. Certain conditions are known as a trap, an exception or a system fault typically caused by an exceptional condition such as division by zero, invalid memory access, etc. If the process is running in kernel mode, such a trap situation can crash the entire operating system. A process in user mode that encounters a trap situation only crashes the user-mode process.

So, the overhead of switching is acceptable to ensure a more stable, secure system.

### **Difference between User Mode and Kernel Mode**

A computer operates either in user mode or kernel mode. The difference between User Mode and Kernel Mode is that user mode is the restricted mode in which the applications are running, and kernel-mode is the privileged mode the computer enters when accessing hardware resources.

The computer is switching between these two modes. Frequent context switching can slow down the speed, but it is impossible to execute all processes in the kernel mode. That is because; if one process fails, the whole operating system might fail. Below are some more differences between User mode and kernel mode, such as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Terms** | **User Mode** | **Kernel Mode** |  | |
| Definition | User Mode is a restricted mode, which the application programs are executing and starts. | Kernel Mode is the privileged mode, which the computer enters when accessing hardware resources. | |  |
| Modes | User Mode is considered as the slave mode or the restricted mode. | Kernel mode is the system mode, master mode or the privileged mode. | |
| Address Space | In User mode, a process gets its own address space. | In Kernel Mode, processes get a single address space. | |  |
| Interruptions | In User Mode, if an interrupt occurs, only one process fails. | In Kernel Mode, if an interrupt occurs, the whole operating system might fail. | |  |
| Restrictions | In user mode, there are restrictions to access kernel programs. Cannot access them directly. | In kernel mode, both user programs and kernel programs can access. | |  |

# Process Vs. Thread | Difference Between Process and Thread

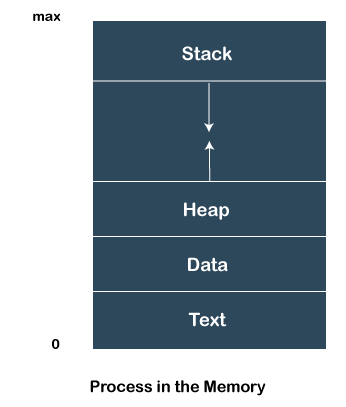
"Difference between process and thread" is one of the widely asked questions of technical interviews. Both processes and threads are related to each other and very much similar, hence create confusion to understand the differences between both of them. The process and thread are an independent sequence of execution, but both are differentiated in a way that processes execute in different memory spaces, whereas threads of the same process execute in shared memory space.

In this topic, we will understand the brief introduction of processes and threads and what are other differences between both of them.

## **What is Process?**

**A process is an instance of a program that is being executed.** When we run a program, it does not execute directly. It takes some time to follow all the steps required to execute the program, and following these execution steps is known as a process.

A process can create other processes to perform multiple tasks at a time; the created processes are known as **clone or child process**, and the main process is known as the **parent process**. Each process contains its own memory space and does not share it with the other processes. It is known as the active entity. A typical process remains in the below form in memory.



A process in OS can remain in any of the following states:

* **NEW**: A new process is being created.
* **READY**: A process is ready and waiting to be allocated to a processor.
* **RUNNING**: The program is being executed.
* **WAITING**: Waiting for some event to happen or occur.
* **TERMINATED**: Execution finished.

### **How do Processes work?**

When we start executing the program, the processor begins to process it. It takes the following steps:

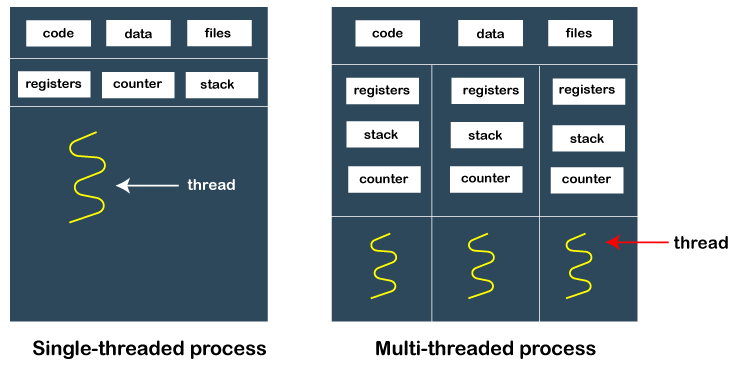
* Firstly, the program is loaded into the computer's memory in binary code after translation.
* A program requires memory and other OS resources to run it. The resources such that registers, program counter, and a stack, and these resources are provided by the OS.
* A register can have an instruction, a storage address, or other data that is required by the process.
* The program counter maintains the track of the program sequence.
* The stack has information on the active subroutines of a computer program.
* A program may have different instances of it, and each instance of the running program is knowns as the individual process.

### **Features of Process**

* Each time we create a process, we need to make a separate system call for each process to the OS. The **fork**() function creates the process.
* Each process exists within its own address or memory space.
* Each process is independent and treated as an isolated process by the OS.
* Processes need IPC (Inter-process Communication) in order to communicate with each other.
* A proper synchronization between processes is not required.

## **What is Thread?**

A thread is the subset of a process and is also known as the lightweight process. A process can have more than one thread, and these threads are managed independently by the scheduler. All the threads within one process are interrelated to each other. Threads have some common information, such as **data segment, code segment, files, etc.,** that is shared to their peer threads. But contains its own registers, stack, and counter.



### **How does thread work?**

As we have discussed that a thread is a subprocess or an execution unit within a process. A process can contain a single thread to multiple threads. A thread works as follows:

* When a process starts, OS assigns the memory and resources to it. Each thread within a process shares the memory and resources of that process only.
* Threads are mainly used to improve the processing of an application. In reality, only a single thread is executed at a time, but due to fast context switching between threads gives an illusion that threads are running parallelly.
* If a single thread executes in a process, it is known as **a single-threaded** And if multiple threads execute simultaneously, then it is known as **multithreading.**

### **Types of Threads**

There are two types of threads, which are:

**1. User Level Thread**

As the name suggests, the user-level threads are only managed by users, and the kernel does not have its information.

These are faster, easy to create and manage.

The kernel takes all these threads as a single process and handles them as one process only.

The user-level threads are implemented by user-level libraries, not by the system calls.

**2. Kernel-Level Thread**

The kernel-level threads are handled by the Operating system and managed by its kernel. These threads are slower than user-level threads because context information is managed by the kernel. To create and implement a kernel-level thread, we need to make a system call.

### **Features of Thread**

* Threads share data, memory, resources, files, etc., with their peer threads within a process.
* One system call is capable of creating more than one thread.
* Each thread has its own stack and register.
* Threads can directly communicate with each other as they share the same address space.
* Threads need to be synchronized in order to avoid unexpected scenarios.

## **Key Differences Between Process and Thread**

* A process is independent and does not contained within another process, whereas all threads are logically contained within a process.
* Processes are heavily weighted, whereas threads are light-weighted.
* A process can exist individually as it contains its own memory and other resources, whereas a thread cannot have its individual existence.
* A proper synchronization between processes is not required. In contrast, threads need to be synchronized in order to avoid unexpected scenarios.
* Processes can communicate with each other using inter-process communication only; in contrast, threads can directly communicate with each other as they share the same address space.

## **Difference Table Between Process and Thread**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Process** | **Thread** |  | | | | | |
| A process is an instance of a program that is being executed or processed. | Thread is a segment of a process or a lightweight process that is managed by the scheduler independently. | | | | | |  |
| Processes are independent of each other and hence don't share a memory or other resources. | Threads are interdependent and share memory. | | | | |  | |
| Each process is treated as a new process by the operating system. | The operating system takes all the user-level threads as a single process. | | | |  | | |
| If one process gets blocked by the operating system, then the other process can continue the execution. | If any user-level thread gets blocked, all of its peer threads also get blocked because OS takes all of them as a single process. | | |  | | | |
| Context switching between two processes takes much time as they are heavy compared to thread. | Context switching between the threads is fast because they are very lightweight. | |  | | | | |
| The data segment and code segment of each process are independent of the other. | Threads share data segment and code segment with their peer threads; hence are the same for other threads also. | |  | | | | |
| The operating system takes more time to terminate a process. | Threads can be terminated in very little time. | |  | | | | |
| New process creation is more time taking as each new process takes all the resources. | A thread needs less time for creation. | |  | | | | |

A process is an active program i.e. a program that is under execution. It is more than the program code as it includes the program counter, process stack, registers, program code etc. Compared to this, the program code is only the text section.

A thread is a lightweight process that can be managed independently by a scheduler. It improves the application performance using parallelism. A thread shares information like data segment, code segment, files etc. with its peer threads while it contains its own registers, stack, counter etc.

The major differences between a process and a thread are given as follows −

| **Comparison Basis** | **Process** | **Thread** |
| --- | --- | --- |
| Definition | A process is a program under execution i.e an active program. | A thread is a lightweight process that can be managed independently by a scheduler. |
| Context switching time | Processes require more time for context switching as they are more heavy. | Threads require less time for context switching as they are lighter than processes. |
| Memory Sharing | Processes are totally independent and don’t share memory. | A thread may share some memory with its peer threads. |
| Communication | Communication between processes requires more time than between threads. | Communication between threads requires less time than between processes . |
| Blocked | If a process gets blocked, remaining processes can continue execution. | If a user level thread gets blocked, all of its peer threads also get blocked. |
| Resource Consumption | Processes require more resources than threads. | Threads generally need less resources than processes. |
| Dependency | Individual processes are independent of each other. | Threads are parts of a process and so are dependent. |
| Data and Code sharing | Processes have independent data and code segments. | A thread shares the data segment, code segment, files etc. with its peer threads. |
| Treatment by OS | All the different processes are treated separately by the operating system. | All user level peer threads are treated as a single task by the operating system. |
| Time for creation | Processes require more time for creation. | Threads require less time for creation. |
| Time for termination | Processes require more time for termination. | Threads require less time for termination. |

# Difference between User level and Kernel level threads in Operating System

In this article, you will learn about the difference between the **user** and **Kernel level threads**. But before discussing the differences, you must know about the user and Kernel level threads and their advantages and disadvantages.

## **What is User Level Thread?**

The **User-level threads** are small and faster as compared to kernel-level threads, and the OS directly supports user-level threads. Users implement the user-level threads, and the kernel is unaware of their existence and handles them as though they are single-threaded processes. These threads are represented by registers, the **program counter (PC)**, stack, and some small process control. Furthermore, there is no kernel interaction in user-level thread synchronization.

It is also known as the many-to-one mapping thread, as the OS assigns every thread in a multithreaded program to an execution context. Every multithreaded process is treated as a single execution unit by the OS.

### **Advantages and Disadvantages of User Level Threads**

There are various advantages and disadvantages of User-level Threads. Some of the advantages and disadvantages of User-level Threads are as follows:

**Advantages**

1. User level threads are simpler and faster to generate. They are also easier to manage.
2. Thread switching in user-level threads doesn't need kernel mode privileges.
3. These are more portable.
4. These threads may be run on any OS.

**Disadvantages**

1. The complete process is blocked if a user-level thread runs a blocking operation.
2. User-level threads don't support system-wide scheduling priorities.
3. It is not appropriate for a multiprocessor system.

## **What are Kernel Level Threads?**

In **Kernel Level Thread**, the kernel handles all thread management. Every process doesn't have a thread table, but the kernel has one that maintains track of all of the threads in the system. If a thread wishes to make a new thread or stop an existing one, it initiates a kernel call that performs the work.

The kernel-level threads table contains each thread's registers, status, and other information. The data is identical to that of user-level threads, except it is now in kernel space rather than user space.

### **Advantages and Disadvantages of Kernel Level Threads**

There are various advantages and disadvantages of kernel-level Threads. Some of the advantages and disadvantages of Kernel-level Threads are as follows:

**Advantages**

1. If a thread in the kernel is blocked, it does not block all other threads in the same process.
2. Several threads of the same process might be scheduled on different CPUs in kernel-level threading.
3. Kernel routines can be multithreaded as well.

**Disadvantages**

1. Compared to user-level threads, kernel-level threads take longer to create and maintain.
2. A mode switch to kernel mode is important to transfer control from one thread in a process to another.

## **Main differences between the User Level Threads and Kernel Level Threads**

Here, you will learn the main differences between the **User Level** and **Kernel Level Threads** in the operating system. The operating system has various differences between the User Level and Kernel Level Threads are as follows:

1. Users implement the user-level threads. On the other hand, the OS implements kernel-level threads.
2. User-level threads may be created and handled much faster. In contrast, kernel-level threads take longer to create and maintain.
3. The entire process is halted if a single user-level thread carries out a blocking operation. On the other hand, if a kernel thread carries out a blocking operation, another thread may continue to run.
4. The user-level thread library includes the source code for thread creation, data transfer, thread destruction, message passing, and thread scheduling. On the other hand, the application code on kernel-level threads does not include thread management code. It is simply an API to the kernel mode.
5. User-level threads do not invoke system calls for scheduling. On the other hand, system calls are used to generate and manage threads at the kernel level.
6. The user-level thread is also referred to as the many-to-one mapping thread, as the OS assigns each thread in a multithreaded process to an execution context. On the other hand, One-to-one thread mapping is supported at the kernel level. Each user thread must be assigned to a kernel thread. This mapping is handled by the OS.
7. Context switch time is less in the user-level threads. On the other hand, context switch time is more in kernel-level threads.
8. User-level threads may operate on any OS. In contrast, kernel-level threads are specific to the OS.
9. Multithread applications are unable to employ multiprocessing in user-level threads. In contrast, Kernel-level threads may be multithreaded.
10. Some instances of user-level threads are Java threads and POSIX threads. On the other hand, some instances of Kernel-level threads are Windows and Solaris.

## **Head-to-head comparison between the User level threads and Kernel level threads**

Here, you will learn the head-to-head comparison between the User Level and Kernel Level Threads in the operating system. The operating system has various differences between the User Level and Kernel Level Threads are as follows:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Features** | **User Level Threads** | **Kernel Level Threads** | |  | | | | | | |
| **Implemented by** | It is implemented by the users. It is implemented by the OS. |  | | | | | | | |  |
| **Context switch time** | Its time is less. | Its time is more. | | | | | | |  | |
| **Multithreading** | Multithread applications are unable to employ multiprocessing in user-level threads. | It may be multithreaded. | | | | | |  | | |
| **Implementation** | It is easy to implement. | It is complicated to implement. | | | | |  | | | |
| **Blocking Operation** | If a thread in the kernel is blocked, it blocks all other threads in the same process. | If a thread in the kernel is blocked, it does not block all other threads in the same process. | | | |  | | | | |
| **Recognize** | OS doesn't recognize it. | It is recognized by OS. | | |  | | | | | |
| **Thread Management** | Its library includes the source code for thread creation, data transfer, thread destruction, message passing, and thread scheduling. | The application code on kernel-level threads does not include thread management code, and it is simply an API to the kernel mode. |  | | | | | | | |
| **Hardware Support** | It doesn't need hardware support. | It requires hardware support. |  | | | | | | | |
| **Creation and Management** | It may be created and managed much faster. | It takes much time to create and handle. |  | | | | | | | |
| **Examples** | Some instances of user-level threads are Java threads and POSIX threads. | Some instances of Kernel-level threads are Windows and Solaris. |  | | | | | | | |
| **Operating System** | Any OS may support it. | The specific OS may support it. |  | | | | | | | |

## **Conclusion**

In summary, the primary distinction between User Level Threads and Kernel Level Threads is that the user maintains User Level Threads. In contrast, the kernel-level threads are handled by the OS. All modern OS supports the threading model, and the implementation of a thread will vary depending on the OS.

# CPU Scheduling in Operating Systems

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.

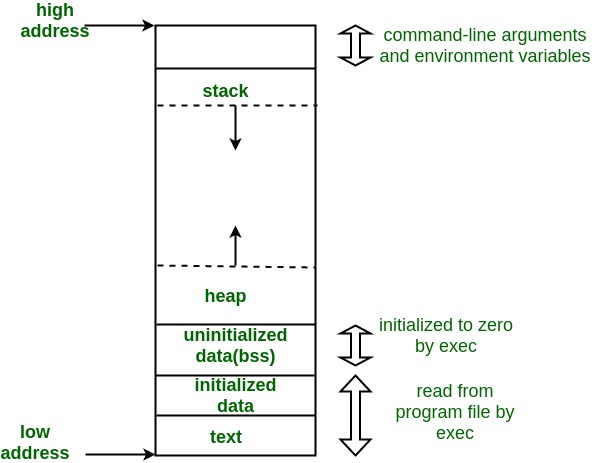
## What is a process?

In computing, 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. Depending on the[operating system](https://www.geeksforgeeks.org/introduction-of-operating-system-set-1/) (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.

## How is process memory used for efficient operation?

The [process memory](https://www.geeksforgeeks.org/memory-layout-of-c-program/) is divided into four sections for efficient operation:

* The**text category**is composed of integrated program code, which is read from fixed storage when the program is launched.
* The**data class**is made up of global and static variables, distributed and executed before the main action.
* [Heap](http://www.geeksforgeeks.org/binary-heap/) is used for flexible, or dynamic memory allocation and is managed by calls to new, delete, malloc, free, etc.
* The[stack](https://www.geeksforgeeks.org/stack-data-structure/) is used for local variables. The space in the stack is reserved for local variables when it is announced.



To know further, you can refer to our detailed article on [**States of a Process in Operating system**](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/)**.**

## What is 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.

Process Scheduling 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.

There are three types of[process schedulers](https://www.geeksforgeeks.org/process-schedulers-in-operating-system/):

* Long term or Job Scheduler
* Short term or CPU Scheduler
* Medium-term Scheduler

## Why do we need to schedule processes?

* **Scheduling** 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 Operating System (OS) of the computer and there are many different ways in which we can choose to configure programs.
* **Process Scheduling**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.
* Considering that there may be hundreds of programs that need to work, the OS must launch the program, stop it, switch to another program, etc. The way the OS configures the system to run another in the CPU is called “[context switching](https://www.geeksforgeeks.org/difference-between-swapping-and-context-switching/)”. If the OS keeps context-switching programs in and out of the provided CPUs, it can give the user a tricky idea that he or she can run any programs he or she wants to run, all at once.
* So now that we know we can run 1 program at a given CPU, and we know we can change the operating system and remove another one using the context switch, how do we choose which programs we need. run, and with what program?
* That’s where**scheduling** comes in! First, you determine the metrics, saying something like “the amount of time until the end”. We will define this metric as “the time interval between which a function enters the system until it is completed”. Second, you decide on a metrics that reduces metrics. We want our tasks to end as soon as possible.

## What is the 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 of the CPU scheduling 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. The function of an effective program is to 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. So 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**.
* **Throughput should be Maximum**. i.e. Number of processes that complete their execution per time unit should be maximized.
* **Minimum turnaround time**, i.e. time taken by a process to finish execution should be the least.
* There should be a **minimum waiting time** and the process should not starve in the ready queue.
* **Minimum** **response time.** It means that the time when a process produces the first response should be as less as possible.

## What are the different terminologies to take care of in any CPU Scheduling algorithm?

* **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*

## Things to take care while designing a CPU Scheduling algorithm?

Different **CPU Scheduling algorithms**have different structures and the choice of a particular algorithm depends on a variety of factors. Many conditions have been raised to compare CPU scheduling algorithms.

The criteria include the following:

* **CPU utilization:**The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, 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:**In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore 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.

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

There are mainly two types of scheduling methods:

* [Preemptive Scheduling](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/): Preemptive scheduling 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/): Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.



*Different types of CPU Scheduling Algorithms*

Let us now learn about these CPU scheduling algorithms in operating systems one by one:

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

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

**Characteristics of FCFS:**

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

**Advantages of FCFS:**

* Easy to implement
* First come, first serve method

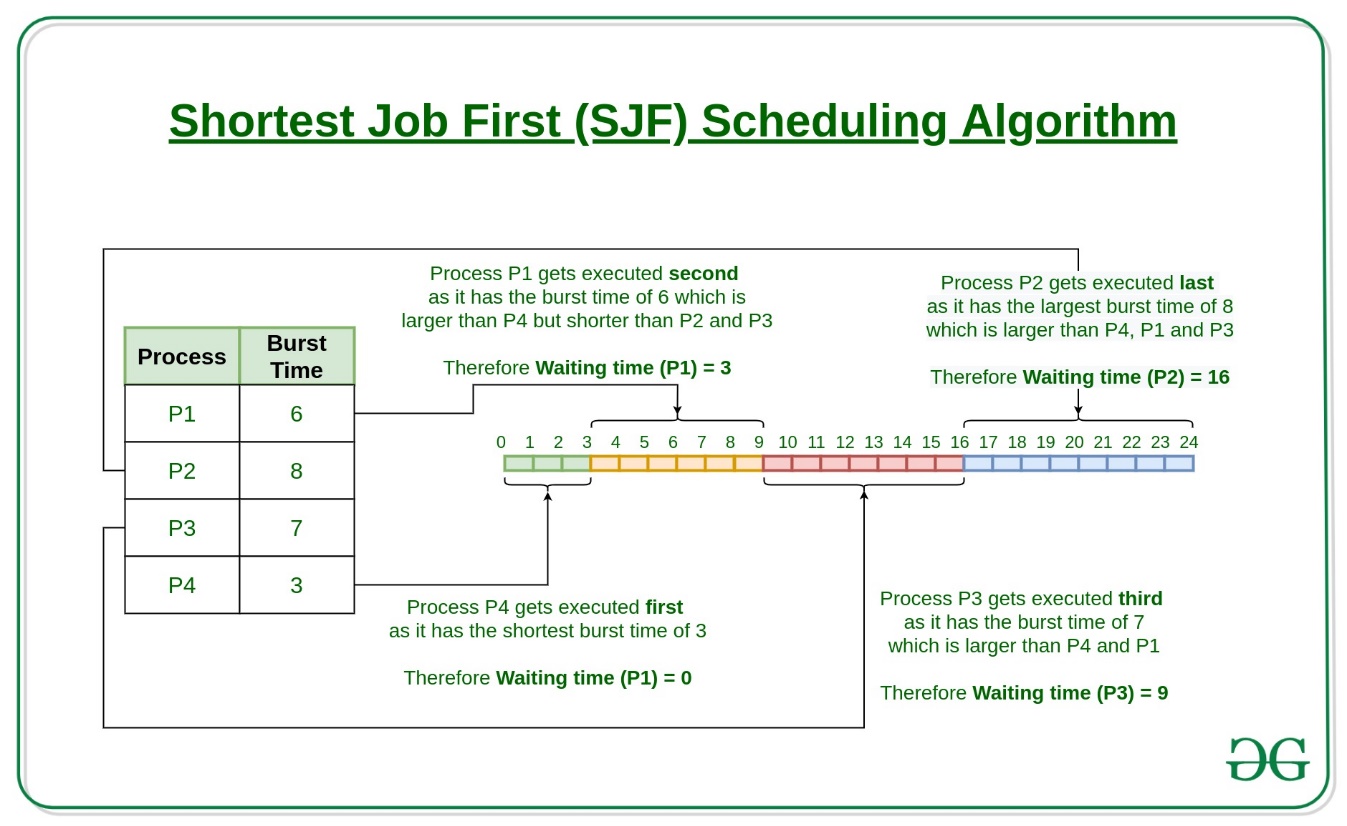
**Disadvantages of FCFS:**

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

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

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

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



**Characteristics of SJF:**

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

**Advantages of Shortest Job first:**

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

**Disadvantages of SJF:**

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

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

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

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

**Characteristics of LJF:**

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

**Advantages of LJF:**

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

**Disadvantages of LJF:**

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

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

### 4. Priority Scheduling:

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

* 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 of Priority Scheduling:**

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

**Disadvantages of Priority Scheduling:**

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

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

### 5. Round robin:

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

**Characteristics of Round robin:**

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

**Advantages of Round robin:**

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

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

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

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

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

**Advantages of SRTF:**

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

**Disadvantages of SRTF:**

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

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

### 7. Longest Remaining Time First:

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

**Characteristics of longest remaining time first:**

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

**Advantages of LRTF:**

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

**Disadvantages of LRTF:**

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

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

### 8. Highest Response Ratio Next:

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

**Characteristics of Highest Response Ratio Next:**

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

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

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

**Advantages of HRRN:**

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

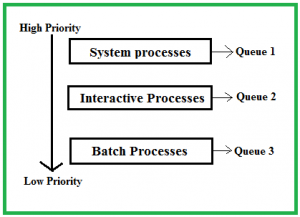
**Disadvantages of HRRN:**

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

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

### 9. Multiple Queue Scheduling:

Processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a **foreground (interactive)** process and a **background (batch)** process. These two classes have different scheduling needs. For this kind of situation **Multilevel Queue Scheduling** is used.



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

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

**Advantages of multilevel queue scheduling:**

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

**Disadvantages of multilevel queue scheduling:**

* Starvation problem
* It is inflexible in nature

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

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

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

**Characteristics of Multilevel Feedback Queue Scheduling:**

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

**Advantages of Multilevel feedback queue scheduling:**

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

**Disadvantages of Multilevel feedback queue scheduling:**

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

## Comparison between various CPU Scheduling algorithms

Here is a brief comparison between different CPU scheduling algorithms:

| Algorithm | Allocation is | Complexity | Average waiting time (AWT) | Preemption | Starvation | Performance |
| --- | --- | --- | --- | --- | --- | --- |
| FCFS | According to the arrival time of the processes, the CPU is allocated. | Simple and easy to implement | Large. | No | No | Slow performance |
| SJF | Based on the lowest CPU burst time  (BT). | More complex than FCFS | Smaller than FCFS | No | Yes | Minimum Average Waiting Time |
| LJFS | Based on the highest CPU burst time (BT) | More complex than FCFS | Depending on some measures e.g., arrival time, process size, etc. | No | Yes | Big turn-around time |
| LRTF | Same as LJFS the allocation of the CPU is based on the highest CPU  burst time (BT). But it is preemptive | More complex than FCFS | Depending on some measures e.g., arrival time, process size, etc. | Yes | Yes | The preference is given to the longer jobs |
| SRTF | Same as SJF the allocation of the CPU is based on the lowest CPU burst time (BT). But it is preemptive. | More complex than FCFS | Depending on some measures e.g., arrival time, process size, etc | Yes | Yes | The preference is given to the short jobs |
| RR | According to the order of the process arrives with fixed time quantum (TQ) | The complexity depends on Time Quantum size | Large as compared to SJF and Priority scheduling. | Yes | No | Each process has given a fairly fixed time |
| Priority Pre-emptive | According to the priority. The bigger priority task executes first | This type is less complex | Smaller than FCFS | Yes | Yes | Well performance but contain a starvation problem |
| Priority non-preemptive | According to the priority with monitoring the new incoming higher priority jobs | This type is less complex than Priority preemptive | Preemptive Smaller than FCFS | No | Yes | Most beneficial with batch systems |
| MLQ | According to the process that resides in the bigger queue priority | More complex than the priority scheduling algorithms | Smaller than FCFS | No | Yes | Good performance but contain a starvation problem |
| MFLQ | According to the process of a bigger priority queue. | It is the most Complex but its complexity rate depends on the TQ size | Smaller than all scheduling types in many cases | No | No | Good performance |

# Multilevel Queue Scheduling in Operating System

In this article, you will learn about multilevel queue scheduling and the several types of processes in the multilevel queue in the OS.

## **Multilevel Queue Scheduling**

Each algorithm supports a different process, but in a general system, some processes require scheduling using a priority algorithm. While some processes want to stay in the system **(interactive processes),** others are **background processes** whose execution can be delayed.

The number of ready queue algorithms between queues and within queues may differ between systems. A round-robin method with various time quantum is typically utilized for such maintenance. Several types of scheduling algorithms are designed for circumstances where the processes can be readily separated into groups. There are two sorts of processes that require different scheduling algorithms because they have varying response times and resource requirements. **The foreground (interactive)** and **background processes** (batch process) are distinguished. Background processes take priority over foreground processes.

The ready queue has been partitioned into seven different queues using the multilevel queue scheduling technique. These processes are assigned to one queue based on their priority, such as memory size, process priority, or type. The method for scheduling each queue is different. Some queues are utilized for the foreground process, while others are used for the background process. The **foreground queue may** be scheduled using a **round-robin method,** and the **background queue can** be scheduled using an **FCFS** strategy.

### **Advantages and Disadvantages of Multilevel Queue Scheduling**

There are various advantages and disadvantages of multilevel queue scheduling. Some of the advantages and disadvantages of the multilevel queue scheduling are as follows:

**Advantages**

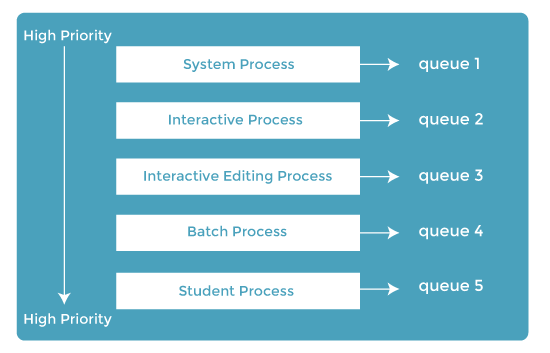
1. You can use multilevel queue scheduling to apply different scheduling methods to distinct processes.
2. It will have low overhead in terms of scheduling.

**Disadvantages**

1. There is a risk of starvation for lower priority processes.
2. It is rigid in nature.

### **Example**

Let's take an example of a multilevel queue-scheduling algorithm with five queues to understand how this scheduling works:



1. **System process**
2. **Interactive processes**
3. **Interactive editing processes**
4. **Batch processes**
5. **Student processes**

Every queue would have an absolute priority over the low-priority queues. No process may execute until the high-priority queues are empty. In the above instance, no other process may execute until and unless the queues for system, interactive, and editing processes are empty. If an interactive editing process enters the ready queue while a batch process is underway, the batch process will be preempted.

There are the descriptions of the processes that are used in the above example:

**System Process**

The OS has its process to execute, which is referred to as the System Process.

**Interactive Process**

It is a process in which the same type of interaction should occur.

**Batch Process**

Batch processing is an operating system feature that collects programs and data into a batch before processing starts.

**Student Process**

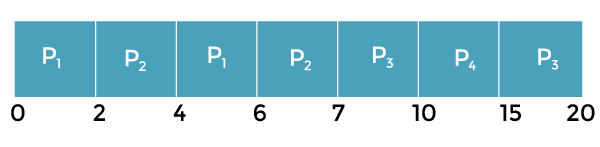
The system process is always given the highest priority, whereas the student processes are always given the lowest.

**Example Problem**

Let's take an example of a multilevel queue-scheduling (MQS) algorithm that shows how the multilevel queue scheduling work. Consider the four processes listed in the table below under multilevel queue scheduling. The queue number denotes the process's queue.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival Time** | **CPU Burst Time** | **Queue Number** |
| P1 | 0 | 4 | 1 |
| P2 | 0 | 3 | 1 |
| P3 | 0 | 8 | 2 |
| P4 | 10 | 5 | 4 |

Queue 1 has a higher priority than queue **2.** Round Robin is used in queue **1 (Time Quantum = 2),** while FCFS is used in queue **2.**



**Working:**

1. Both queues have been processed at the start. Therefore, **queue 1 (P1, P2)** runs first (due to greater priority) in a round-robin way and finishes after 7 units.
2. The process in **queue 2 (Process P3)** starts running (since there is no process in queue 1), but while it is executing, P4 enters queue 1 and interrupts P3, and then P3 takes the CPU and finishes its execution.

## **Multilevel Feedback Scheduling**

Each algorithm supports a different process, but some processes require scheduling using a priority algorithm in a general system. There is a different queue for foreground or background operations, but they do not switch between queues or change their foreground or background nature; this type of organization benefits from low scheduling but is inflexible.

This strategy prioritizes operations that require I/O and are interactive. It is a distinct process with a distinct CPU burst time. It enables a process to switch between queues. If a process consumes too much processor time, it will be switched to the lowest priority queue. A process waiting in a lower priority queue for too long may be shifted to a higher priority queue. This type of aging prevents starvation.

The parameters of the multilevel feedback queue scheduler are as follows:

1. The scheduling algorithm for every queue in the system.
2. The queues number in the system.
3. The method for determining when a queue should be demoted to a lower-priority queue.
4. When a process is upgraded to a higher-priority queue, this process determines when it gets upgraded.
5. The method for determining which processes will enter the queue and when those processes will require service

# Introduction of Process Synchronization

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

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

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

### Race Condition:

 When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition. Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place. A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in the critical section differs according to the order in which the threads execute. Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

### 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.**

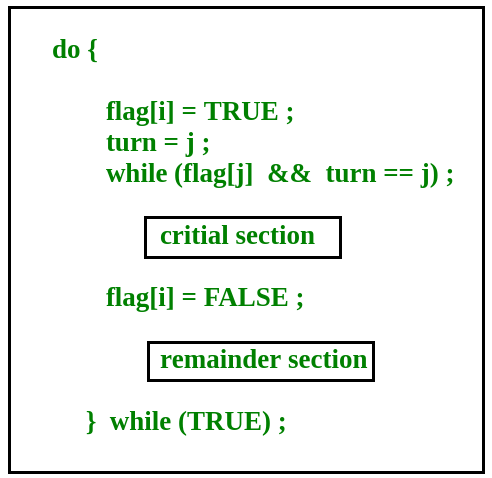
Any solution to the critical section problem must satisfy three requirements:

* **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 of Peterson’s solution:**

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

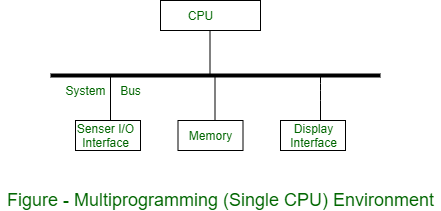
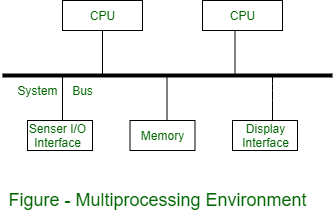
### **Semaphores:A** semaphore is a signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that is called the wait function.

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().  
There are two 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.

# Concurrent Processes in Operating System

**Concurrent processing** is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means, which occurs when something else happens. The tasks are broken into subtypes, which are then assigned to different processors to perform simultaneously, sequentially instead, as they would have to be performed by one processor. Concurrent processing is sometimes synonymous with parallel processing. The term real and virtual concurrency in concurrent processing:

1. **Multiprogramming Environment:** In a multiprogramming environment, there are multiple tasks shared by one processor. While a virtual concept can be achieved by the operating system, if the processor is allocated for each individual task, the virtual concept is visible if each task has a dedicated processor. The multilayer environment is shown in figure. 
2. **Multiprocessing Environment :** In multiprocessing environment two or more processors are used with shared memory. Only one virtual address space is used, which is common for all processors. All tasks reside in shared memory. In this environment, concurrency is supported in the form of concurrently executing processors. The tasks executed on different processors are performed with each other through shared memory. The multiprocessing environment is shown in figure. 
3. **Distributed Processing Environment :** In a distributed processing environment, two or more computers are connected to each other by a communication network or high speed bus. There is no shared memory between the processors and each computer has its own local memory. Hence a distributed application consisting of concurrent tasks, which are distributed over network communication via messages. The distributed processing environment is shown in figure. 

# Precedence Graph in Operating System

**Precedence Graph** is a directed acyclic graph which is used to show the execution level of several processes in operating system. It consists of nodes and edges. Nodes represent the processes and the edges represent the flow of execution. **Properties of Precedence Graph :** Following are the properties of Precedence Graph:

* It is a directed graph.
* It is an acyclic graph.
* Nodes of graph correspond to individual statements of program code.
* Edge between two nodes represents the execution order.
* A directed edge from node A to node B shows that statement A executes first and then Statement B executes.

Consider the following code:

S1 : a = x + y;

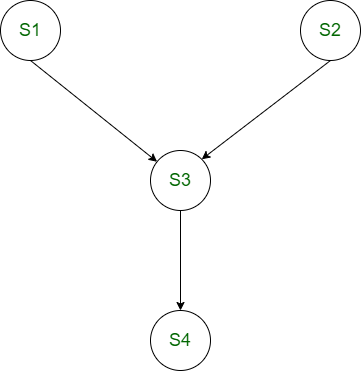
S2 : b = z + 1;

S3 : c = a - b;

S4 : w = c + 1;

If above code is executed concurrently, the following precedence relations exist:

* c = a – b cannot be executed before both a and b have been assigned values.
* w = c + 1 cannot be executed before the new values of c has been computed.
* The statements a = x + y and b = z + 1 could be executed concurrently.



**Example:** Consider the following precedence relations of a program:

1. S2 and S3 can be executed after S1 completes.
2. S4 can be executed after S2 completes.
3. S5 and S6 can be executed after S4 completes.
4. S7 can be executed after S5, S6 and S3 complete.

**Solution:** 

# Producer Consumer Problem 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.

wait(S){

while(S<=0); // busy waiting

S--;

}

signal(S){

S++;

}

Semaphores are of two types:

1. **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.
2. **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 –**

do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

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 –**

do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

signal(empty);

// consumes item

}while(true)

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.

# What is a Print Spooler?

A printer spooler is a simple program that manages all print jobs that are submitted to the print server or a computer printer. It allows the users to store multiple print jobs within a print queue or a buffer without affecting the application or the underlying system performance, and it also allows to delete a print job being processed. It enables a user to manage the print jobs currently waiting to be printed.

A printer spooler helps to understand some software components, which establish the printing process when you are trying to remove problems with a printer that is connected to a Windows computer. It eliminates the requirement to keep an application running until the printing process is finished. A print spooler's most significant purpose is to control the sequence in which documents are printed. A printer spooler may control printing in the background while you continue to use your computer. When you give the order to print a document, nothing comes out of the printer. This is because the print spooler needs to be reset as it may be hanged up.

Some print jobs are stored in a print queue or a buffer because computer printers take time to print. When the printer has printed all documents and is available for the next job, it retrieves the pending document from the print queue and prints it. Furthermore, with the help of a printer spooler, users can also view current jobs within their size, status, time, print queue, and the ability to suspend and delete them. With current print spoolers, the printing process has a little influence on user productivity. In contrast, users had a need to wait until a document was printed before performing other actions in the early days of personal computers.

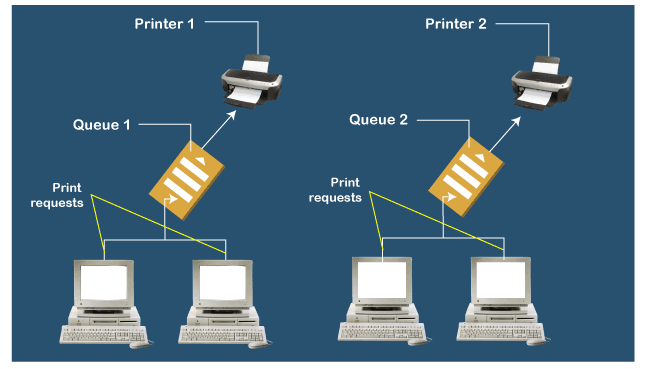
## **Stuck in the Print Queue**

A print queue is a collection of print jobs that are waiting to be printed from a list. All the print jobs are shown in a dialog, which can be seen after you describe the file for printing. When a printer has completed printing any document, it is removed from the queue of waiting jobs. When you are printing a document, and it stays in the queue without printing, in this case, you need to check the printer first for fixing problems like lack of ink or paper. Also, if there are no evident printer issues, there could be a problem with the files you're printing. Additionally, for the job that does not print from the queue, you can remove that job, and do not worry about other documents; they will print correctly.

## **Why do you need a printer spooler?**

For the printer, controlling all of the information about what you're printing is more difficult. This is due to the fact that it lacks sufficient memory and is too slow. For case, when your printer is connected in an office where many people give the command to print the document at the same time. Therefore, a program is required for the printer, which decides the order of the documents what document should be printed first (on the basis of who clicked print first). And, instead of sending it to all printers at once, it slowly passes the list of documents to be printed.

You will have a spooler built into your device if you use a Windows operating system. With your printer, a spooler functions for ordering the print jobs. Users had a need to wait until one job is finished before performing other actions in the early days of personal computers, but in modern times, a spooler fixes this problem, and you do not need to wait for one job to finish before loading up another. It simply puts all the jobs in a queue and prints with the order.



## **Common problems**

After you click print, there are different issues involved with the spooler with printers. In most cases, the spooler has added a document to the print queue or has sent a document to the printer if there is an issue with it. Then, it will stop all the prints jobs behind it in the queue. These include:

* When you click print, it may be stuck, and nothing comes out from the printer.
* In the spooler, the spooler cannot translate the data or document for the printer as the spooler becoming corrupt.
* There may be a spooler service failure. Through the transfer of information, the spooler or PC software has crashed partway.

All the problems given above mean that your printer will refuse to print anymore and stop in its tracks.

## **Steps to Clear Print Queue**

The print spooler can be the culprit if all the print jobs in the queue have got stuck and nothing comes out. The spooler may have encountered an error or may be hung. There is a symptom of a spooler problem when the presence of print jobs stays in the queue and appears to be stuck, and it cannot be deleted.

Microsoft recommends following these below steps if a print job will not delete in Windows 10.

* **Cycle printer power:** Cycling the printer power may be the solution to delete a stuck job in the queue; first, you need to turn off and unplug the printer, then wait for 30 seconds and turn it on the back again.
* **Check the printer connection:** If your printer is wireless, check the wireless network connection of the computer and run the printer's wireless diagnostics. If your printer is wired, check the cable is connected.
* **Remove and reinstall the printer:** With the help of using the Remove Device, removing and reinstalling the printer helps you out to delete the job stuck in the queue. And, in Window settings, add Device command in the scanners and printer function.
* **Updating printer driver:** The printer manufacturer's website allows the users to update the printer driver that may help remove errors. Therefore, you must to download the latest version of the printer driver.
* **Run the Microsoft printing troubleshooter:** Microsoft Support website allows the users to run the Microsoft printing troubleshooter feature, which may help them out to delete the job stuck in the queue.

After following these five steps, if you are unable to clear the print queue, nothing is printing; it is recommended by Microsoft to clear and restart the print spooler. This is done with the help of using the Service desktop app. The Microsoft Support website has instructions for using the program to reset the print spooler in Windows 10..

## **Check for Spooler Diagnostic Tools**

Some free software tools are offered by certain printer manufacturers to help solve issues with their printers. For instance, the company Hewlett Packard makes printers and scanners and provides HP Print and Scan Doctor for users on its website. Once you download the tool, issues and diagnose caused by hardware or software can be detected by this tool. This type of utility can address print spooler issues without requiring access to Windows Services by downloading and running them.

## **How to restart the print spooler service**

If a computer is having trouble sending a print job to a printer, it could be a problem with the print spooler service. You can follow some given steps to reset the print spooler service if it is not running or is hung.

* First, you need to access the Windows Task Manager.
* Then, click the Services tab in the Task Manager window.
* Now, to sort the list of services by description, click on the Description column header.
* Scroll down the page to find the Print Spooler service in the Description column. The service is called Spooler in the name column.
* Finally, you need to right-click on the Print Spooler service, and in the pop-up menu, select the Restart option.

# The Critical Section Problem

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

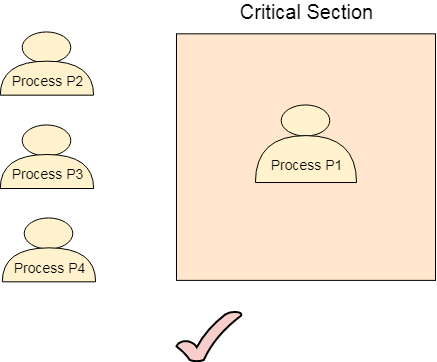
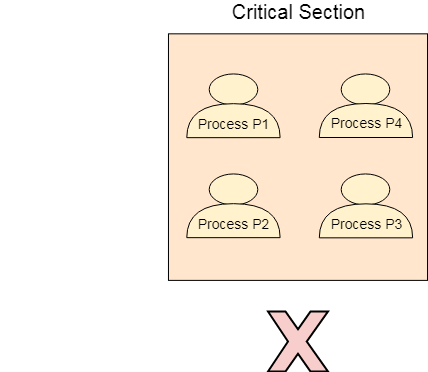
In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

## **Requirements of Synchronization mechanisms**

**Primary**

1. **Mutual Exclusion**

Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.

1. **Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

### **Secondary**

1. **Bounded Waiting**

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

1. **Architectural Neutrality**

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

# Lock Variable

This is the simplest synchronization mechanism. This is a Software Mechanism implemented in User mode. This is a busy waiting solution which can be used for more than two processes.

In this mechanism, a Lock variable **lock** is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.

A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

The pseudo code of the mechanism looks like following.

1. Entry Section →
2. While (lock! = 0);
3. Lock = 1;
4. //Critical Section
5. Exit Section →
6. Lock =0;

If we look at the Pseudo Code, we find that there are three sections in the code. Entry Section, Critical Section and the exit section.

Initially the value of **lock variable** is**0**. The process which needs to get into the **critical section**, enters into the entry section and checks the condition provided in the while loop.

The process will wait infinitely until the value of **lock** is 1 (that is implied by while loop). Since, at the very first time critical section is vacant hence the process will enter the critical section by setting the lock variable as 1.

When the process exits from the critical section, then in the exit section, it reassigns the value of**lock** as 0.

Every Synchronization mechanism is judged on the basis of four conditions.

1. Mutual Exclusion
2. Progress
3. Bounded Waiting
4. Portability

Out of the four parameters, Mutual Exclusion and Progress must be provided by any solution. Let?s analyze this mechanism on the basis of the above mentioned conditions.

## **Mutual Exclusion**

The lock variable mechanism doesn't provide Mutual Exclusion in some of the cases. This can be better described by looking at the pseudo code by the Operating System point of view I.E. Assembly code of the program. Let's convert the Code into the assembly language.

1. Load Lock, R0
2. CMP R0, #0
3. JNZ Step 1
4. Store #1, Lock
5. Store #0, Lock

Let us consider that we have two processes P1 and P2. The process P1 wants to execute its critical section. P1 gets into the entry section. Since the value of lock is 0 hence P1 changes its value from 0 to 1 and enters into the critical section.

Meanwhile, P1 is preempted by the CPU and P2 gets scheduled. Now there is no other process in the critical section and the value of lock variable is 0. P2 also wants to execute its critical section. It enters into the critical section by setting the lock variable to 1.

Now, CPU changes P1's state from waiting to running. P1 is yet to finish its critical section. P1 has already checked the value of lock variable and remembers that its value was 0 when it previously checked it. Hence, it also enters into the critical section without checking the updated value of lock variable.

Now, we got two processes in the critical section. According to the condition of mutual exclusion, morethan one process in the critical section must not be present at the same time. Hence, the lock variable mechanism doesn't guarantee the mutual exclusion.

The problem with the lock variable mechanism is that, at the same time, more than one process can see the vacant tag and more than one process can enter in the critical section. Hence, the lock variable doesn't provide the mutual exclusion that's why it cannot be used in general.

Since, this method is failed at the basic step; hence, there is no need to talk about the other conditions to be fulfilled.

# Test Set Lock Mechanism

## **Modification in the assembly code**

In lock variable mechanism, Sometimes Process reads the old value of lock variable and enters the critical section. Due to this reason, more than one process might get into critical section. However, the code shown in the part one of the following section can be replaced with the code shown in the part two. This doesn't affect the algorithm but, by doing this, we can manage to provide the mutual exclusion to some extent but not completely.

In the updated version of code, the value of Lock is loaded into the local register R0 and then value of lock is set to 1.

However, in step 3, the previous value of lock (that is now stored into R0) is compared with 0. if this is 0 then the process will simply enter into the critical section otherwise will wait by executing continuously in the loop.

The benefit of setting the lock immediately to 1 by the process itself is that, now the process which enters into the critical section carries the updated value of lock variable that is 1.

In the case when it gets preempted and scheduled again then also it will not enter the critical section regardless of the current value of the lock variable as it already knows what the updated value of lock variable is.

|  |  |
| --- | --- |
| **Section 1** | **Section 2** |
| 1. Load Lock, R0  2. CMP R0, #0  3. JNZ step1  4. store #1, Lock | 1. Load Lock, R0  2. Store #1, Lock  3. CMP R0, #0  4. JNZ step 1 |

## **TSL Instruction**

However, the solution provided in the above segment provides mutual exclusion to some extent but it doesn't make sure that the mutual exclusion will always be there. There is a possibility of having more than one process in the critical section.

What if the process gets preempted just after executing the first instruction of the assembly code written in section 2? In that case, it will carry the old value of lock variable with it and it will enter into the critical section regardless of knowing the current value of lock variable. This may make the two processes present in the critical section at the same time.

To get rid of this problem, we have to make sure that the preemption must not take place just after loading the previous value of lock variable and before setting it to 1. The problem can be solved if we can be able to merge the first two instructions.

In order to address the problem, the operating system provides a special instruction called **Test Set Lock (TSL)** instruction which simply loads the value of lock variable into the local register R0 and sets it to 1 simultaneously

The process which executes the TSL first will enter into the critical section and no other process after that can enter until the first process comes out. No process can execute the critical section even in the case of preemption of the first process.

The assembly code of the solution will look like following.

1. TSL Lock, R0
2. CMP R0, #0
3. JNZ step 1

Let's examine TSL on the basis of the four conditions.

* **Mutual Exclusion**

Mutual Exclusion is guaranteed in TSL mechanism since a process can never be preempted just before setting the lock variable. Only one process can see the lock variable as 0 at a particular time and that's why, the mutual exclusion is guaranteed.

* **Progress**

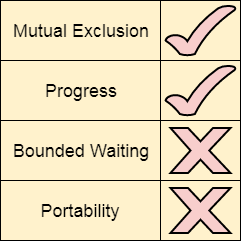
According to the definition of the progress, a process which doesn't want to enter in the critical section should not stop other processes to get into it. In TSL mechanism, a process will execute the TSL instruction only when it wants to get into the critical section. The value of the lock will always be 0 if no process doesn't want to enter into the critical section hence the progress is always guaranteed in TSL.

* **Bounded Waiting**

Bounded Waiting is not guaranteed in TSL. Some process might not get a chance for so long. We cannot predict for a process that it will definitely get a chance to enter in critical section after a certain time.

* **Architectural Neutrality**

TSL doesn't provide Architectural Neutrality. It depends on the hardware platform. The TSL instruction is provided by the operating system. Some platforms might not provide that. Hence it is not Architectural natural.



# Turn Variable or Strict Alternation Approach

Turn Variable or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A turn variable is used which is actually a lock.

This approach can only be used for only two processes. In general, let the two processes be Pi and Pj. They share a variable called turn variable. The pseudo code of the program can be given as following.

**For Process Pi**

1. Non - CS
2. while (turn ! = i);
3. Critical Section
4. turn = j;
5. Non - CS

**For Process Pj**

1. Non - CS
2. while (turn ! = j);
3. Critical Section
4. turn = i ;
5. Non - CS

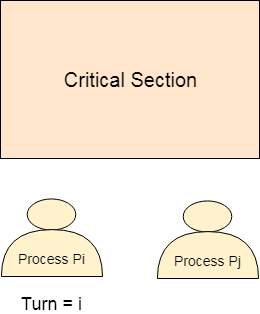
The actual problem of the lock variable approach was the fact that the process was entering in the critical section only when the lock variable is 1. More than one process could see the lock variable as 1 at the same time hence the mutual exclusion was not guaranteed there.

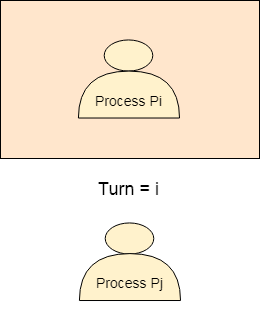
This problem is addressed in the turn variable approach. Now, A process can enter in the critical section only in the case when the value of the turn variable equal to the PID of the process.

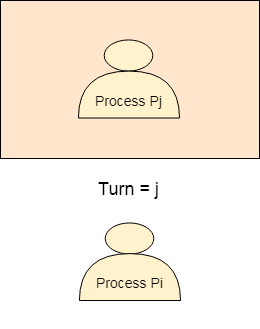
There are only two values possible for turn variable, i or j. if its value is not i then it will definitely be j or vice versa.

In the entry section, in general, the process Pi will not enter in the critical section until its value is j or the process Pj will not enter in the critical section until its value is i.

Initially, two processes Pi and Pj are available and want to execute into critical section.

  
The turn variable is equal to i hence Pi will get the chance to enter into the critical section. The value of Pi remains I until Pi finishes critical section.

  
Pi finishes its critical section and assigns j to turn variable. Pj will get the chance to enter into the critical section. The value of turn remains j until Pj finishes its critical section.



## **Analysis of Strict Alternation approach**

Let's analyse Strict Alternation approach on the basis of four requirements.

### **Mutual Exclusion**

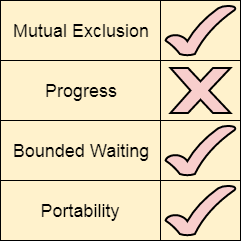
The strict alternation approach provides mutual exclusion in every case. This procedure works only for two processes. The pseudo code is different for both of the processes. The process will only enter when it sees that the turn variable is equal to its Process ID otherwise not Hence No process can enter in the critical section regardless of its turn.

### **Progress**

Progress is not guaranteed in this mechanism. If Pi doesn't want to get enter into the critical section on its turn then Pj got blocked for infinite time. Pj has to wait for so long for its turn since the turn variable will remain 0 until Pi assigns it to j.

### **Portability**

The solution provides portability. It is a pure software mechanism implemented at user mode and doesn't need any special instruction from the Operating System.



# Readers-Writers Problem | (Introduction and Readers Preference Solution)

Consider a situation where we have a file shared between many people. 

* If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
* However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the **readers-writers problem**

Problem parameters: 

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

**Solution when Reader has the Priority over Writer**

Here priority means, no reader should wait if the share is currently opened for reading.

Three variables are used: **mutex, wrt, readcnt** to implement solution 

1. **semaphore** mutex, wrt; // semaphore **mutex** is used to ensure mutual exclusion when **readcnt** is updated i.e. when any reader enters or exit from the critical section and semaphore **wrt** is used by both readers and writers
2. **int** readcnt;  //    **readcnt** tells the number of processes performing read in the critical section, initially 0

**Functions for semaphore :**

– wait() : decrements the semaphore value.

– signal() : increments the semaphore value.

**Writer process:** 

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

do {

// writer requests for critical section

wait(wrt);

// performs the write

// leaves the critical section

signal(wrt);

} while(true);

**Reader process:** 

1. Reader requests the entry to critical section.
2. If allowed:
   * it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wrt** semaphore to restrict the entry of writers if any reader is inside.
   * It then, signals mutex as any other reader is allowed to enter while others are already reading.
   * After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

do {

// Reader wants to enter the critical section

wait(mutex);

// The number of readers has now increased by 1

readcnt++;

// there is atleast one reader in the critical section

**// this ensure no writer can enter if there is even one reader**

**// thus we give preference to readers here**

if (readcnt==1)

wait(wrt);

// other readers can enter while this current reader is inside

// the critical section

signal(mutex);

// current reader performs reading here

wait(mutex);   // a reader wants to leave

readcnt--;

// that is, no reader is left in the critical section,

if (readcnt == 0)

signal(wrt);         // writers can enter

signal(mutex); // reader leaves

} while(true);

Thus, the semaphore ‘**wrt**‘ is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

# READERS WRITERS PROBLEM

The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets.

The readers-writers problem is used for managing synchronization among various reader and writer process so that there are no problems with the data sets, i.e. no inconsistency is generated.

Let's understand with an example - If two or more than two readers want to access the file at the same point in time there will be no problem. However, in other situations like when two writers or one reader and one writer wants to access the file at the same point of time, there may occur some problems, hence the task is to design the code in such a manner that if one reader is reading then no writer is allowed to update at the same point of time, similarly, if one writer is writing no reader is allowed to read the file at that point of time and if one writer is updating a file other writers should not be allowed to update the file at the same point of time. However, multiple readers can access the object at the same time.

Let us understand the possibility of reading and writing with the table given below:

**TABLE 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Process 1** | **Process 2** | **Allowed / Not Allowed** |
| Case 1 | Writing | Writing | Not Allowed |
| Case 2 | Reading | Writing | Not Allowed |
| Case 3 | Writing | Reading | Not Allowed |
| Case 4 | Reading | Reading | Allowed |

The solution of readers and writers can be implemented using binary semaphores.

We use two binary semaphores "write" and "mutex", where binary semaphore can be defined as:

Semaphore: A semaphore is an integer variable in S, that apart from initialization is accessed by only two standard atomic operations - wait and signal, whose definitions are as follows:

1. 1. wait( S )
2. {
3. **while**( S <= 0) ;
4. S--;
5. }
7. 2. signal( S )
8. {
9. S++;
10. }

From the above definitions of wait, it is clear that if the value of S <= 0 then it will enter into an infinite loop (because of the semicolon; after while loop). Whereas the job of the signal is to increment the value of S.

The below code will provide the solution of the reader-writer problem, reader and writer process codes are given as follows -

### **Code for Reader Process**

The code of the reader process is given below -

1. **static** **int** readcount = 0;
2. wait (mutex);
3. readcount ++; // on each entry of reader increment readcount
4. **if** (readcount == 1)
5. {
6. wait (write);
7. }
8. signal(mutex);
10. --READ THE **FILE**?
12. wait(mutex);
13. readcount --; // on every exit of reader decrement readcount
14. **if** (readcount == 0)
15. {
16. signal (write);
17. }
18. signal(mutex);

In the above code of reader, **mutex** and **write** are **semaphores** that have an initial value of 1, whereas **the readcount** variable has an initial value as 0. Both **mutex** and **write** are common in reader and writer process code, semaphore **mutex** ensures mutual exclusion and semaphore write handles the writing mechanism.

The **readcount** variable denotes the number of readers accessing the file concurrently. The moment variable **readcount** becomes 1, **wait** operation is used to write semaphore which decreases the value by one. This means that a writer is not allowed how to access the file anymore. On completion of the read operation, **readcount** is decremented by one. When **readcount** becomes 0, the signal operation which is used to **write** permits a writer to access the file.

### **Code for Writer Process**

The code that defines the writer process is given below:

1. wait(write);
2. WRITE INTO THE **FILE**
3. signal(wrt);

If a writer wishes to access the file, **wait** operation is performed on **write** semaphore, which decrements **write** to 0 and no other writer can access the file. On completion of the writing job by the writer who was accessing the file, the signal operation is performed on **write**.

**Let's see the proof of each case mentioned in Table 1**

**CASE 1:** WRITING - WRITING → NOT ALLOWED. That is when two or more than two processes are willing to write, then it is not allowed. Let us see that our code is working accordingly or not?

1. Explanation :
2. The initial value of semaphore write = 1
3. Suppose two processes P0 and P1 wants to write, let P0 enter first the writer code, The moment P0 enters
4. Wait( write ); will decrease semaphore write by one, now write = 0
5. And **continue** WRITE INTO THE **FILE**
6. Now suppose P1 wants to write at the same time (will it be allowed?) let's see.
7. P1 does Wait( write ), since the write value is already 0, therefore from the definition of wait, it will go into an infinite loop (i.e. Trap), hence P1 can never write anything, till P0 is writing.
8. Now suppose P0 has finished the task, it will
9. signal( write); will increase semaphore write by 1, now write = 1
10. **if** now P1 wants to write it since semaphore write > 0
11. This proofs that, **if** one process is writing, no other process is allowed to write.

**CASE 2:** READING - WRITING → NOT ALLOWED. That is when one or more than one process is reading the file, then writing by another process is not allowed. Let us see that our code is working accordingly or not?

1. Explanation:
2. Initial value of semaphore mutex = 1 and variable readcount = 0
3. Suppose two processes P0 and P1 are in a system, P0 wants to read **while** P1 wants to write, P0 enter first into the reader code, the moment P0 enters
4. Wait( mutex ); will decrease semaphore mutex by 1, now mutex = 0
5. Increment readcount by 1, now readcount = 1, next
6. **if** (readcount == 1)// evaluates to TRUE
7. {
8. wait (write); // decrement write by 1, i.e. write = 0(which
9. clearly proves that **if** one or more than one
10. reader is reading then no writer will be
11. allowed.
12. }
14. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to enter.
16. And reader continues to --READ THE **FILE**?
18. Suppose now any writer wants to enter into its code then:
20. As the first reader has executed wait (write); because of which write value is 0, therefore wait(writer); of the writer, code will go into an infinite loop and no writer will be allowed.
21. This proofs that, **if** one process is reading, no other process is allowed to write.
22. Now suppose P0 wants to stop the reading and wanted to exit then
23. Following sequence of instructions will take place:
24. wait(mutex); // decrease mutex by 1, i.e. mutex = 0
25. readcount --; // readcount = 0, i.e. no one is currently reading
26. **if** (readcount == 0) // evaluates TRUE
27. {
28. signal (write); // increase write by one, i.e. write = 1
29. }
30. signal(mutex);// increase mutex by one, i.e. mutex = 1
32. Now **if** again any writer wants to write, it can **do** it now, since write > 0

**CASE 3:** WRITING -- READING → NOT ALLOWED. That is when if one process is writing into the file, then reading by another process is not allowed. Let us see that our code is working accordingly or not?

1. Explanation:
2. The initial value of semaphore write = 1
3. Suppose two processes P0 and P1 are in a system, P0 wants to write **while** P1 wants to read, P0 enter first into the writer code, The moment P0 enters
4. Wait( write ); will decrease semaphore write by 1, now write = 0
5. And **continue** WRITE INTO THE **FILE**
6. Now suppose P1 wants to read the same time (will it be allowed?) let's see.
7. P1 enters reader's code
8. Initial value of semaphore mutex = 1 and variable readcount = 0
9. Wait( mutex ); will decrease semaphore mutex by 1, now mutex = 0
10. Increment readcount by 1, now readcount = 1, next
11. **if** (readcount == 1)// evaluates to TRUE
12. {
13. wait (write); // since value of write is already 0, hence it
14. will enter into an infinite loop and will not be
15. allowed to proceed further (which clearly
16. proves that **if** one writer is writing then no
17. reader will be allowed.
18. }
20. The moment writer stops writing and willing to exit then
21. This proofs that, **if** one process is writing, no other process is allowed to read.
23. The moment writer stops writing and willing to exit then it will execute:
24. signal( write); will increase semaphore write by 1, now write = 1
25. **if** now P1 wants to read it can since semaphore write > 0

**CASE 4:** READING - READING → ALLOWED. That is when one process is reading the file, and other process or processes is willing to read, then they all are allowed i.e. reading - reading is not mutually exclusive. Let us see that our code is working accordingly or not?

1. Explanation :
2. Initial value of semaphore mutex = 1 and variable readcount = 0
3. Suppose three processes P0, P1 and P2 are in a system, all the three processes P0, P1, and P2 want to read, let P0 enter first into the reader code, the moment P0 enters
4. Wait( mutex ); will decrease semaphore mutex by 1, now mutex = 0
5. Increment readcount by 1, now readcount = 1, next
6. **if** (readcount == 1)// evaluates to TRUE
7. {
8. wait (write); // decrement write by 1, i.e. write = 0(which
9. clearly proves that **if** one or more than one
10. reader is reading then no writer will be
11. allowed.
12. }
14. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to enter.
16. And P0 continues to --READ THE **FILE**?
18. →Now P1 wants to enter the reader code
19. current value of semaphore mutex = 1 and variable readcount = 1
20. let P1 enter into the reader code, the moment P1 enters
21. Wait( mutex ); will decrease semaphore mutex by 1, now mutex = 0
22. Increment readcount by 1, now readcount = 2, next
23. **if** (readcount == 1)// eval. to False, it will not enter if block
25. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to enter.
27. Now P0 and P1 continues to --READ THE **FILE**?
29. →Now P2 wants to enter the reader code
30. current value of semaphore mutex = 1 and variable readcount = 2
31. let P2 enter into the reader code, The moment P2 enters
32. Wait( mutex ); will decrease semaphore mutex by 1, now mutex = 0
33. Increment readcount by 1, now readcount = 3, next
34. **if** (readcount == 1)// eval. to False, it will not enter if block
36. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to enter.
38. Now P0, P1, and P2 continues to --READ THE **FILE**?

41. Suppose now any writer wants to enter into its code then:
43. As the first reader P0 has executed wait (write); because of which write value is 0, therefore wait(writer); of the writer, code will go into an infinite loop and no writer will be allowed.
45. Now suppose P0 wants to come out of system( stop reading) then
46. wait(mutex); //will decrease semaphore mutex by 1, now mutex = 0
47. readcount --; // on every exit of reader decrement readcount by
48. one i.e. readcount = 2
50. **if** (readcount == 0)// eval. to FALSE it will not enter if block
52. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to exit

55. → Now suppose P1 wants to come out of system (stop reading) then
56. wait(mutex); //will decrease semaphore mutex by 1, now mutex = 0
57. readcount --; // on every exit of reader decrement readcount by
58. one i.e. readcount = 1
60. **if** (readcount == 0)// eval. to FALSE it will not enter if block
62. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1 i.e. other readers are allowed to exit
64. →Now suppose P2 (last process) wants to come out of system (stop reading) then
65. wait(mutex); //will decrease semaphore mutex by 1, now mutex = 0
66. readcount --; // on every exit of reader decrement readcount by
67. one i.e. readcount = 0
69. **if** (readcount == 0)// eval. to TRUE it will enter into if block
70. {
71. signal (write); // will increment semaphore write by one, i.e.
72. now write = 1, since P2 was the last process
73. which was reading, since now it is going out,
74. so by making write = 1 it is allowing the writer
75. to write now.
76. }
78. signal(mutex); // will increase semaphore mutex by 1, now mutex = 1

81. The above explanation proves that **if** one or more than one processes are willing to read simultaneously

# Semaphores in Operating System

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows −

* **Wait**

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

wait(S)

{

   while (S<=0);

   S--;

}

* **Signal**

The signal operation increments the value of its argument S.

signal(S)

{

   S++;

}

## **Types of Semaphores**

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows −

* **Counting Semaphores**

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

* **Binary Semaphores**

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

## **Advantages of Semaphores**

Some of the advantages of semaphores are as follows −

* Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
* There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
* Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

## **Disadvantages of Semaphores**

Some of the disadvantages of semaphores are as follows −

* Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
* Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
* Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

Counting Semaphore

There are the scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in the critical section at the same time.

The programming code of semaphore implementation is shown below which includes the structure of semaphore and the logic using which the entry and the exit can be performed in the critical section.

1. struct Semaphore
2. {
3. int value; // processes that can enter in the critical section simultaneously.
4. queue type L; // L contains set of processes which get blocked
5. }
6. Down (Semaphore S)
7. {
8. SS.value = S.value - 1; //semaphore's value will get decreased when a new
9. //process enter in the critical section
10. if (S.value**<** **0**)
11. {
12. put\_process(PCB) in L; //if the value is negative then
13. //the process will get into the blocked state.
14. Sleep();
15. }
16. else
17. return;
18. }
19. up (Semaphore s)
20. {
21. SS.value = S.value+1; //semaphore value will get increased when
22. //it makes an exit from the critical section.
23. if(S.value**<**=0)
24. {
25. select a process from L; //if the value of semaphore is positive
26. //then wake one of the processes in the blocked queue.
27. wake-up();
28. }
29. }
30. }

In this mechanism, the entry and exit in the critical section are performed on the basis of the value of counting semaphore. The value of counting semaphore at any point of time indicates the maximum number of processes that can enter in the critical section at the same time.

A process which wants to enter in the critical section first decrease the semaphore value by 1 and then check whether it gets negative or not. If it gets negative then the process is pushed in the list of blocked processes (i.e. q) otherwise it gets enter in the critical section.

When a process exits from the critical section, it increases the counting semaphore by 1 and then checks whether it is negative or zero. If it is negative then that means that at least one process is waiting in the blocked state hence, to ensure bounded waiting, the first process among the list of blocked processes will wake up and gets enter in the critical section.

The processes in the blocked list will get waked in the order in which they slept. If the value of counting semaphore is negative then it states the number of processes in the blocked state while if it is positive then it states the number of slots available in the critical section.

Binary Semaphore or Mutex

In counting semaphore, Mutual exclusion was not provided because we has the set of processes which required to execute in the critical section simultaneously.

However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slots available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

Let's see the programming implementation of Binary Semaphore.

1. StructBsemaphore
2. {
3. enum Value(0,1); //value is enumerated data type which can only have two values 0 or 1.
4. Queue type L;
5. }
6. /\* L contains all PCBs corresponding to process
7. Blocked while processing down operation unsuccessfully.
8. \*/
9. Down (Bsemaphore S)
10. {
11. if (s.value == 1) // if a slot is available in the
12. //critical section then let the process enter in the queue.
13. {
14. S.value = 0; // initialize the value to 0 so that no other process can read it as 1.
15. }
16. else
17. {
18. put the process (PCB) in S.L; //if no slot is available
19. //then let the process wait in the blocked queue.
20. sleep();
21. }
22. }
23. Up (Bsemaphore S)
24. {
25. if (S.L is empty) //an empty blocked processes list implies that no process
26. //has ever tried to get enter in the critical section.
27. {
28. S.Value =1;
29. }
30. else
31. {
32. Select a process from S.L;
33. Wakeup(); // if it is not empty then wake the first process of the blocked queue.
34. }
35. }

# THE DINING PHILOSOPHERS PROBLEM

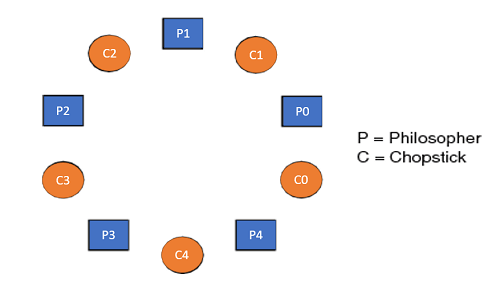
The dining philosopher's problem is the classical problem of synchronization which says that Five philosophers are sitting around a circular table and their job is to think and eat alternatively. A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers. To eat a philosopher needs both their right and a left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosopher is available. In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again.

The dining philosopher demonstrates a large class of concurrency control problems hence it's a classic synchronization problem.



**Five Philosophers sitting around the table**

**Dining Philosophers Problem**- Let's understand the Dining Philosophers Problem with the below code, we have used fig 1 as a reference to make you understand the problem exactly. The five Philosophers are represented as P0, P1, P2, P3, and P4 and five chopsticks by C0, C1, C2, C3, and C4.



1. Void Philosopher
2. {
3. **while**(1)
4. {
5. take\_chopstick[i];
6. take\_chopstick[ (i+1) % 5] ;
7. . .
8. . EATING THE NOODLE
9. .
10. put\_chopstick[i] );
11. put\_chopstick[ (i+1) % 5] ;
12. .
13. . THINKING
14. }
15. }

Let's discuss the above code:

Suppose Philosopher P0 wants to eat, it will enter in Philosopher() function, and execute **take\_chopstick[i];** by doing this it holds **C0 chopstick** after that it execute **take\_chopstick[ (i+1) % 5];** by doing this it holds **C1 chopstick**( since i =0, therefore (0 + 1) % 5 = 1)

Similarly suppose now Philosopher P1 wants to eat, it will enter in Philosopher() function, and execute **take\_chopstick[i];** by doing this it holds **C1 chopstick** after that it execute **take\_chopstick[ (i+1) % 5];** by doing this it holds **C2 chopstick**( since i =1, therefore (1 + 1) % 5 = 2)

But Practically Chopstick C1 is not available as it has already been taken by philosopher P0, hence the above code generates problems and produces race condition.

### **The solution of the Dining Philosophers Problem**

We use a semaphore to represent a chopstick and this truly acts as a solution of the Dining Philosophers Problem. Wait and Signal operations will be used for the solution of the Dining Philosophers Problem, for picking a chopstick wait operation can be executed while for releasing a chopstick signal semaphore can be executed.

Semaphore: A semaphore is an integer variable in S, that apart from initialization is accessed by only two standard atomic operations - wait and signal, whose definitions are as follows:

1. 1. wait( S )
2. {
3. **while**( S <= 0) ;
4. S--;
5. }
7. 2. signal( S )
8. {
9. S++;
10. }

From the above definitions of wait, it is clear that if the value of S <= 0 then it will enter into an infinite loop(because of the semicolon; after while loop). Whereas the job of the signal is to increment the value of S.

The structure of the chopstick is an array of a semaphore which is represented as shown below -

1. semaphore C[5];

Initially, each element of the semaphore C0, C1, C2, C3, and C4 are initialized to 1 as the chopsticks are on the table and not picked up by any of the philosophers.

Let's modify the above code of the Dining Philosopher Problem by using semaphore operations wait and signal, the desired code looks like

1. **void** Philosopher
2. {
3. **while**(1)
4. {
5. Wait( take\_chopstickC[i] );
6. Wait( take\_chopstickC[(i+1) % 5] ) ;
7. . .
8. . EATING THE NOODLE
9. .
10. Signal( put\_chopstickC[i] );
11. Signal( put\_chopstickC[ (i+1) % 5] ) ;
12. .
13. . THINKING
14. }
15. }

In the above code, first wait operation is performed on take\_chopstickC[i] and take\_chopstickC [ (i+1) % 5]. This shows philosopher i have picked up the chopsticks from its left and right. The eating function is performed after that.

On completion of eating by philosopher i the, signal operation is performed on take\_chopstickC[i] and take\_chopstickC [ (i+1) % 5]. This shows that the philosopher i have eaten and put down both the left and right chopsticks. Finally, the philosopher starts thinking again.

### **Let's understand how the above code is giving a solution to the dining philosopher problem?**

Let value of i = 0( initial value ), Suppose Philosopher P0 wants to eat, it will enter in Philosopher() function, and execute **Wait( take\_chopstickC[i] );** by doing this it holds **C0 chopstick** and reduces semaphore C0 to 0**,** after that it execute **Wait( take\_chopstickC[(i+1) % 5] );** by doing this it holds **C1 chopstick**( since i =0, therefore (0 + 1) % 5 = 1) and reduces semaphore C1 to 0

Similarly, suppose now Philosopher P1 wants to eat, it will enter in Philosopher() function, and execute **Wait( take\_chopstickC[i] );** by doing this it will try to hold **C1 chopstick** but will not be able to do that**,** since the value of semaphore C1 has already been set to 0 by philosopher P0, therefore it will enter into an infinite loop because of which philosopher P1 will not be able to pick chopstick C1 whereas if Philosopher P2 wants to eat, it will enter in Philosopher() function, and execute **Wait( take\_chopstickC[i] );** by doing this it holds **C2 chopstick** and reduces semaphore C2 to 0, after that, it executes **Wait( take\_chopstickC[(i+1) % 5] );** by doing this it holds **C3 chopstick**( since i =2, therefore (2 + 1) % 5 = 3) and reduces semaphore C3 to 0.

Hence the above code is providing a solution to the dining philosopher problem, A philosopher can only eat if both immediate left and right chopsticks of the philosopher are available else philosopher needs to wait. Also at one go two independent philosophers can eat simultaneously (i.e., philosopher **P0 and P2, P1 and P3 & P2 and P4** can eat simultaneously as all are the independent processes and they are following the above constraint of dining philosopher problem)

### **The drawback of the above solution of the dining philosopher problem**

From the above solution of the dining philosopher problem, we have proved that no two neighboring philosophers can eat at the same point in time. The drawback of the above solution is that this solution can lead to a deadlock condition. This situation happens if all the philosophers pick their left chopstick at the same time, which leads to the condition of deadlock and none of the philosophers can eat.

To avoid deadlock, some of the solutions are as follows -

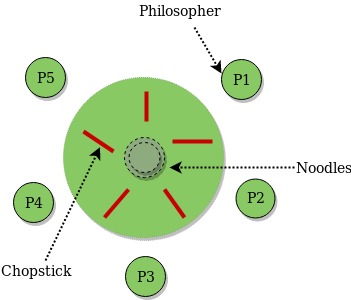
* Maximum number of philosophers on the table should not be more than four, in this case, chopstick C4 will be available for philosopher P3, so P3 will start eating and after the finish of his eating procedure, he will put down his both the chopstick C3 and C4, i.e. semaphore C3 and C4 will now be incremented to 1. Now philosopher P2 which was holding chopstick C2 will also have chopstick C3 available, hence similarly, he will put down his chopstick after eating and enable other philosophers to eat.
* A philosopher at an even position should pick the right chopstick and then the left chopstick while a philosopher at an odd position should pick the left chopstick and then the right chopstick.
* Only in case if both the chopsticks ( left and right ) are available at the same time, only then a philosopher should be allowed to pick their chopsticks
* All the four starting philosophers ( P0, P1, P2, and P3) should pick the left chopstick and then the right chopstick, whereas the last philosopher P4 should pick the right chopstick and then the left chopstick. This will force P4 to hold his right chopstick first since the right chopstick of P4 is C0, which is already held by philosopher P0 and its value is set to 0, i.e C0 is already 0, because of which P4 will get trapped into an infinite loop and chopstick C4 remains vacant. Hence philosopher P3 has both left C3 and right C4 chopstick available, therefore it will start eating and will put down its both chopsticks once finishes and let others eat which removes the problem of deadlock.

The design of the problem was to illustrate the challenges of avoiding deadlock, a deadlock state of a system is a state in which no progress of system is possible. Consider a proposal where each philosopher is instructed to behave as follows:

* The philosopher is instructed to think till the left fork is available, when it is available, hold it.
* The philosopher is instructed to think till the right fork is available, when it is available, hold it.
* The philosopher is instructed to eat when both forks are available.
* then, put the right fork down first
* then, put the left fork down next
* repeat from the beginning.

# Dining Philosopher Problem Using Semaphores

**The Dining Philosopher Problem –** The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. 



**Semaphore Solution to Dining Philosopher –**  
Each philosopher is represented by the following pseudocode: 

process P[i]

while true do

{ THINK;

PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);

EAT;

PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])

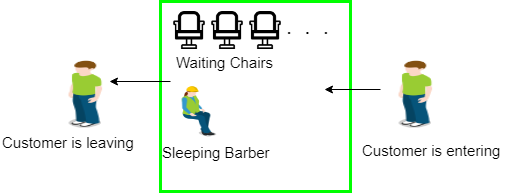
}

There are three states of the philosopher: **THINKING, HUNGRY, and EATING**. Here there are two semaphores: Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

# Sleeping Barber problem in Process Synchronization

**Problem :** The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.

* If there is no customer, then the barber sleeps in his own chair.
* When a customer arrives, he has to wake up the barber.
* If there are many customers and the barber is cutting a customer’s hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.



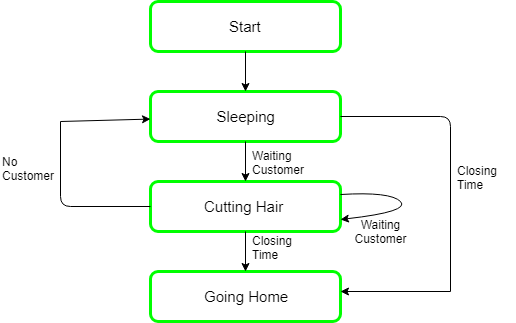
**Solution :** The solution to this problem includes three [semaphores](https://www.geeksforgeeks.org/semaphores-operating-system/).First is for the customer which counts the number of customers present in the waiting room (customer in the barber chair is not included because he is not waiting). Second, the barber 0 or 1 is used to tell whether the barber is idle or is working, And the third mutex is used to provide the mutual exclusion which is required for the process to execute. In the solution, the customer has the record of the number of customers waiting in the waiting room if the number of customers is equal to the number of chairs in the waiting room then the upcoming customer leaves the barbershop.

When the barber shows up in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. Then the barber goes to sleep until the first customer comes up.

When a customer arrives, he executes customer procedure the customer acquires the mutex for entering the critical region, if another customer enters thereafter, the second one will not be able to anything until the first one has released the mutex. The customer then checks the chairs in the waiting room if waiting customers are less then the number of chairs then he sits otherwise he leaves and releases the mutex.

If the chair is available then customer sits in the waiting room and increments the variable waiting value and also increases the customer’s semaphore this wakes up the barber if he is sleeping.

At this point, customer and barber are both awake and the barber is ready to give that person a haircut. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.



# What is Inter Process Communication?

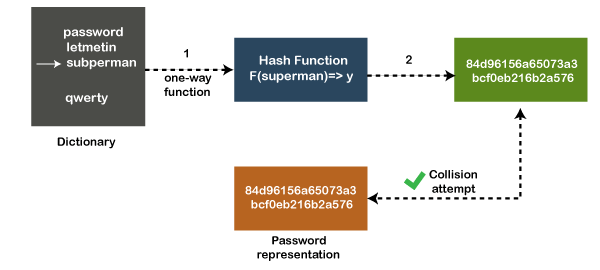
In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

Let us now look at the general definition of inter-process communication, which will explain the same thing that we have discussed above.

### **Definition**

"Inter-process communication is used for exchanging useful information between numerous threads in one or more processes (or programs)."

To understand inter process communication, you can consider the following given diagram that illustrates the importance of inter-process communication:



### **Role of Synchronization in Inter Process Communication**

It is one of the essential parts of inter process communication. Typically, this is provided by interprocess communication control mechanisms, but sometimes it can also be controlled by communication processes.

These are the following methods that used to provide the synchronization:

1. **Mutual Exclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

**Mutual Exclusion:-**

It is generally required that only one process thread can enter the critical section at a time. This also helps in synchronization and creates a stable state to avoid the race condition.

**Semaphore:-**

Semaphore is a type of variable that usually controls the access to the shared resources by several processes. Semaphore is further divided into two types which are as follows:

1. Binary Semaphore
2. Counting Semaphore

**Barrier:-**

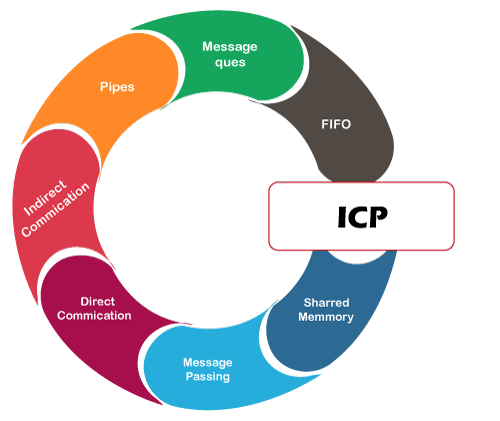
A barrier typically not allows an individual process to proceed unless all the processes does not reach it. It is used by many parallel languages, and collective routines impose barriers.

**Spinlock:-**

Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or stays in a loop while checking that the lock is available or not. It is known as busy waiting because even though the process active, the process does not perform any functional operation (or task).

### **Approaches to Interprocess Communication**

We will now discuss some different approaches to inter-process communication which are as follows:



These are a few different approaches for Inter- Process Communication:

1. **Pipes**
2. **Shared Memory**
3. **Message Queue**
4. **Direct Communication**
5. **Indirect communication**
6. **Message Passing**
7. **FIFO**

To understand them in more detail, we will discuss each of them individually.

**Pipe:-**

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

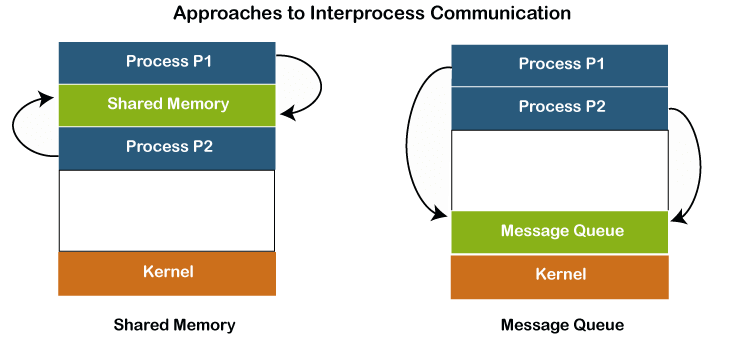
**Shared Memory:-**

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

**Message Queue:-**

In general, several different messages are allowed to read and write the data to the message queue. In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them. In short, we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

To understand the concept of Message queue and Shared memory in more detail, let's take a look at its diagram given below:



**Message Passing:-**

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the hared variables.

Usually, the inter-process communication mechanism provides two operations that are as follows:

* send (message)
* received (message)

#### **Note: The size of the message can be fixed or variable.**

**Direct Communication:-**

In this type of communication process, usually, a link is created or established between two communicating processes. However, in every pair of communicating processes, only one link can exist.

**Indirect Communication**

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

**FIFO:-**

It is a type of general communication between two unrelated processes. It can also be considered as full-duplex, which means that one process can communicate with another process and vice versa.

### **Some other different approaches**

* **Socket:-**

It acts as a type of endpoint for receiving or sending the data in a network. It is correct for data sent between processes on the same computer or data sent between different computers on the same network. Hence, it used by several types of operating systems.

* **File:-**

A file is a type of data record or a document stored on the disk and can be acquired on demand by the file server. Another most important thing is that several processes can access that file as required or needed.

* **Signal:-**

As its name implies, they are a type of signal used in inter process communication in a minimal way. Typically, they are the massages of systems that are sent by one process to another. Therefore, they are not used for sending data but for remote commands between multiple processes.Usually, they are not used to send the data but to remote commands in between several processes.

### **Why we need interprocess communication?**

There are numerous reasons to use inter-process communication for sharing the data. Here are some of the most important reasons that are given below:

* It helps to speedup modularity
* Computational
* Privilege separation
* Convenience
* Helps operating system to communicate with each other and synchronize their actions as well.

# What is Deadlock in Operating System (OS)?

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

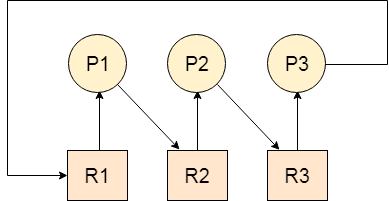
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

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

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

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



### **Difference between Starvation and Deadlock**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.** | **Deadlock** | **Starvation** |  | |
| 1 | Deadlock is a situation where no process got blocked and no process proceeds | Starvation is a situation where the low priority process got blocked and the high priority processes proceed. | |  |
| 2 | Deadlock is an infinite waiting. | Starvation is a long waiting but not infinite. | |  |
| 3 | Every Deadlock is always a starvation. | Every starvation need not be deadlock. | | |
| 4 | The requested resource is blocked by the other process. | The requested resource is continuously be used by the higher priority processes. | | |
| 5 | Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously. | It occurs due to the uncontrolled priority and resource management. | | |

## **Necessary conditions for Deadlocks**

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

1. **Hold and Wait**

A process waits for some resources while holding another resource at the same time.

1. **No preemption**

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

1. **Circular Wait**

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

# Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

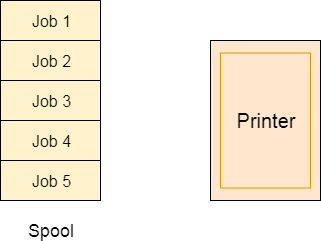
## **1. Mutual Exclusion**

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

### **Spooling**

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

## **2. Hold and Wait**

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

**!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)**

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

## **3. No Preemption**

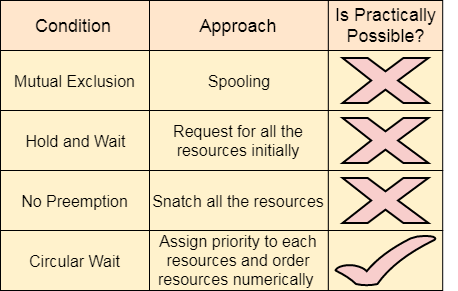
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

## **4. Circular Wait**

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.



Among all the methods, violating Circular wait is the only approach that can be implemented practically

# Deadlock avoidance

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

#### **Resources Assigned**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Type 1** | **Type 2** | **Type 3** | **Type 4** |
| A | 3 | 0 | 2 | 2 |
| B | 0 | 0 | 1 | 1 |
| C | 1 | 1 | 1 | 0 |
| D | 2 | 1 | 4 | 0 |

#### **Resources still needed**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Type 1** | **Type 2** | **Type 3** | **Type 4** |
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 1 | 2 |
| C | 1 | 2 | 1 | 0 |
| D | 2 | 1 | 1 | 2 |

1. E = (7 6 8 4)
2. P = (6 2 8 3)
3. A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

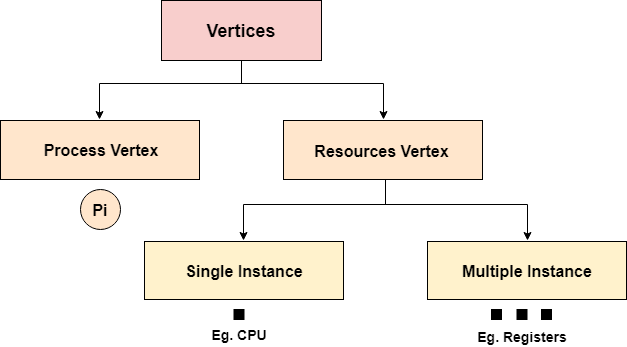
The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

# Resource Allocation Graph

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

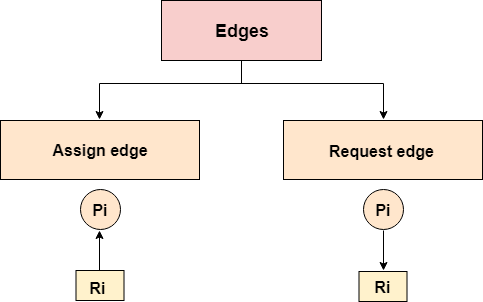
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

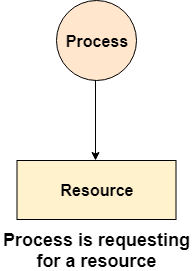
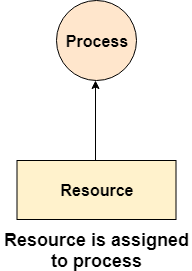
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

### **Example**

Let'sconsider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

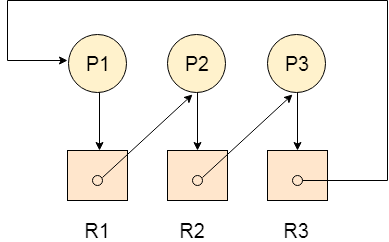
The graph is deadlock free since no cycle is being formed in the graph.

# OS Resource Allocation Graph 4Deadlock Detection using RAG

If a cycle is being formed in a Resource allocation graph where all the resources have the single instance then the system is deadlocked.

In Case of Resource allocation graph with multi-instanced resource types, Cycle is a necessary condition of deadlock but not the sufficient condition.

The following example contains three processes P1, P2, P3 and three resources R2, R2, R3. All the resources are having single instances each.



If we analyze the graph then we can find out that there is a cycle formed in the graph since the system is satisfying all the four conditions of deadlock.

### **Allocation Matrix**

Allocation matrix can be formed by using the Resource allocation graph of a system. In Allocation matrix, an entry will be made for each of the resource assigned. For Example, in the following matrix, en entry is being made in front of P1 and below R3 since R3 is assigned to P1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **R1** | **R2** | **R3** |
| P1 | 0 | 0 | 1 |
| P2 | 1 | 0 | 0 |
| P3 | 0 | 1 | 0 |

### **Request Matrix**

In request matrix, an entry will be made for each of the resource requested. As in the following example, P1 needs R1 therefore an entry is being made in front of P1 and below R1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **R1** | **R2** | **R3** |
| P1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 0 |
| P3 | 0 | 0 | 1 |

### **Avial = (0,0,0)**

Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

We cannot fulfill the demand of at least one process using the available resources therefore the system is deadlocked as determined earlier when we detected a cycle in the graph.

# Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes.

## **For Resource**

### **Preempt the resource**

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

### **Rollback to a safe state**

System passes through various states to get into the deadlock state. The operating system canrollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

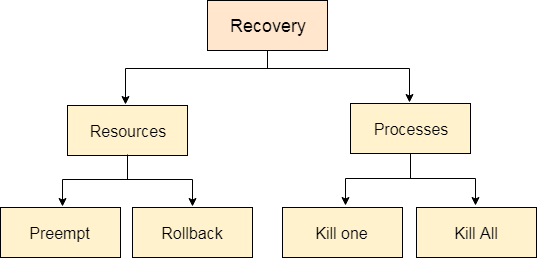
## **For Process**

### **Kill a process**

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

### **Kill all process**

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



**What is the hard disk architecture in operating systems?**

Khizar Hayat Saani

**Hard disks** are secondary storage devices we can use to store data. Most modern computers use hard disks to store large amounts of data.

The architecture of a hard disk consists of several physical components that include:

* Platters
* Spindle
* Read/write heads
* Tracks
* Sectors

### Platters

Hard disks are organized as a concentric stack of disks. An individual disk is referred to as a **platter**.

Each platter consists of two surfaces: a lower and an upper surface.

## Spindle

The platters within the hard disk are connected by a **spindle** that runs through the middle of the platters.

The spindle moves in a unidirectional manner along its axis (either clockwise or counterclockwise).

The movement of the spindle causes the platters to rotate as well.

## Read/write head

Each surface on a platter contains a **read/write head** that is used to read or write data onto the disk.

The read/write heads can move back and forth along the surface of a platter. Read/write heads are in turn connected to a single *actuator arm*.

## Tracks

Each surface of a platter consists of a fixed number of **tracks**. These are circular areas on the surface of a platter that decrease in circumference as we move towards the center of the platter.

Data is first written to the outermost track.

## Sectors

Each track is divided into a fixed number of **sectors**. Sectors divide track sections and store data.

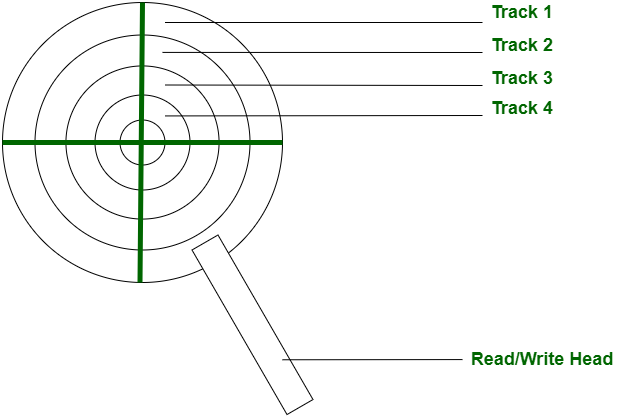
# Difference between Seek Time and Disk Access Time in Disk Scheduling

**Seek Time:**

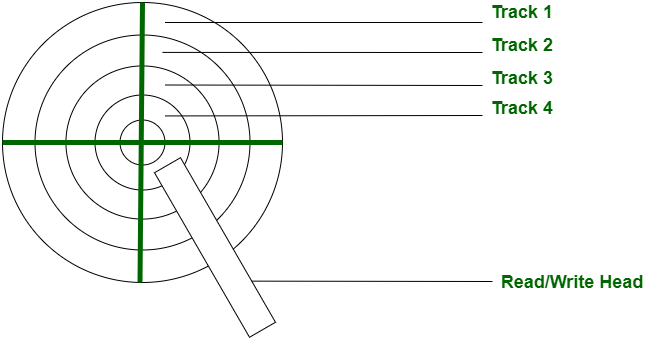
A [disk](https://www.geeksforgeeks.org/hard-disk-drive-hdd-secondary-memory/) is divided into many circular tracks. Seek Time is defined as the time required by the read/write head to move from one track to another.

Example,

Consider the following diagram, the read/write head is currently on track 1.



Now, on the next read/write request, we may want to read data from Track 4, in this case, our read/write head will move to track 4. The time it will take to reach track 4 is the **seek time**.



**Disk Access Time:**

Disk Access Time is defined as the total time required by the computer to process a read/write request and then retrieve the required data from the disk storage.

Disk Access Time is divided into 2 parts:

1. Access Time
2. Data Transfer Time

Disk Access Time = Access Time + Data Transfer Time

**1. Access Time:**

Access Time is defined as the setup time before the actual data transfer takes place.

For example, the read/write head is on track 1 but we need to read data from another track or segment. Thus, the read/write head will move to the data block location before the actual transfer can take place. This delay is called Access Time.

Access Time is calculated by summation of the following:

**(a).** Seek Time

**(b).** Rotational Latency

**(c).** Command Processing Time

**(d).** Settle Time

These are explained as following below in brief.

**(a). Seek Time –**It is the time required by the read/write head to move from the current track to the requested track.  
Seek Time

* = (Number of tracks/cylinders crossed) \* (Time to cross one track/cylinder)

**(b). Rotational Latency –**It is the time required by the read/write head to move from the current sector to the requested sector.  
Rotational Latency

* = (Angle by which disk is rotated) / (Angular Frequency)
* **(c). Command Processing Time –**It is the time required by the disk device to process the command and establish a connection between the various components of the disk device to read/write data. It is due to the internal circuitry.
* **(d). Settle Time –**Settle Time is the time required by read/write head to stop vibrating.

**Note:** Command Processing Time and Settle Time are not normally mentioned in numerical question. We take them as zero.

**2. Data Transfer Time:**

Data Transfer Time is defined as the time required to transfer data between the system and the disk.

Data Transfer Time is of two types:

**(a).** Internal Transfer Rate

**(b).** External Transfer Rate

These are explained as following below in brief.

* **(a). Internal Transfer Rate –**It is defined as the time required to move data between the disk surface and hard disk cache.
* **(b). External Transfer Rate –**It is defined as the time required to move data between the hard disk cache and the system.

Let’s see the difference between Seek Time and Disk Access Time:

|  |  |  |
| --- | --- | --- |
| S.NO. | Seek Time | Disk Access Time |
| 1 | It is the time required by the read/write head to move from one track to another. | It is the time required by the computer to process a read/write request and retrieve the required data. |
| 2 | It is always less than Disk Access Time. Since, it is a sub part of Disk Access time. | It is very large compared to Seek time. |
| 3 | It doesn’t consider transfer of data. | It considers the time required to transfer data.c |

# Disk Scheduling Algorithms

**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.

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 request 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.

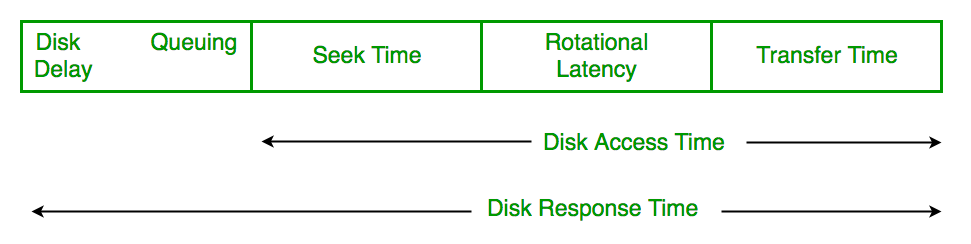
There are many Disk Scheduling Algorithms but before discussing them let’s have a quick look at some of the important terms:

* **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



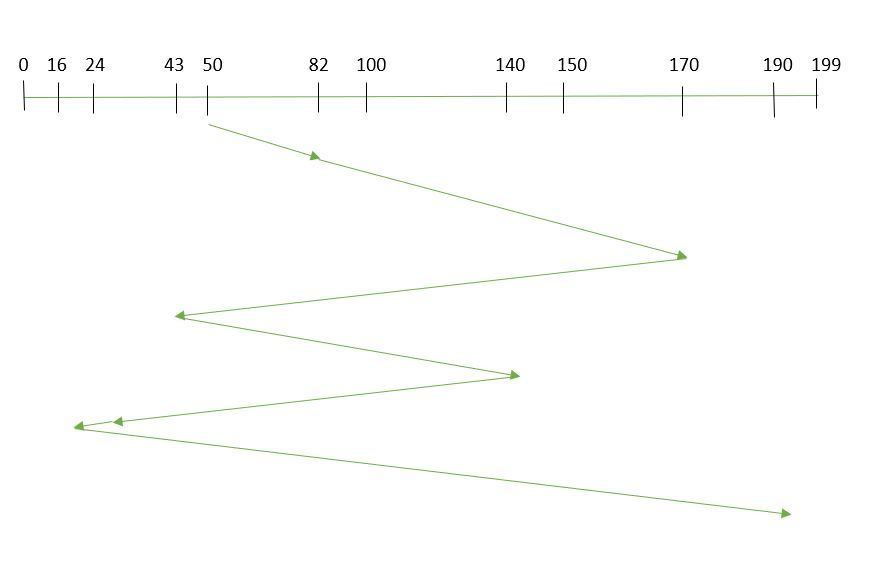
* **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

**Disk Scheduling Algorithms**

1. **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.Let us understand this with the help of an example.

#### Example:

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



1. 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

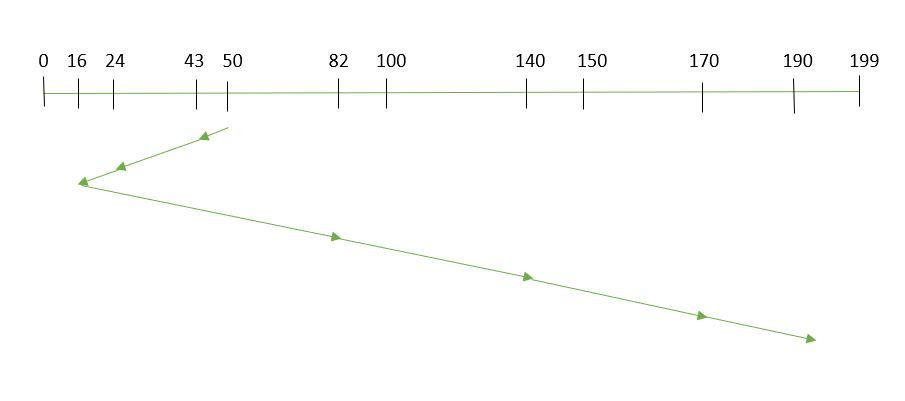
Disadvantages:

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

1. **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.Let us understand this with the help of an example.

#### 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 seek time:

1. =(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170)   
   =208

Advantages:

* Average Response Time decreases
* Throughput increases

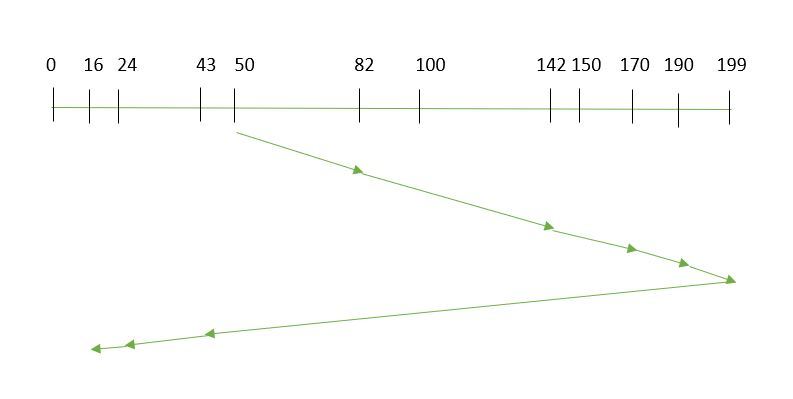
Disadvantages:

* 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

1. **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:

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

Advantages:

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

Disadvantages:

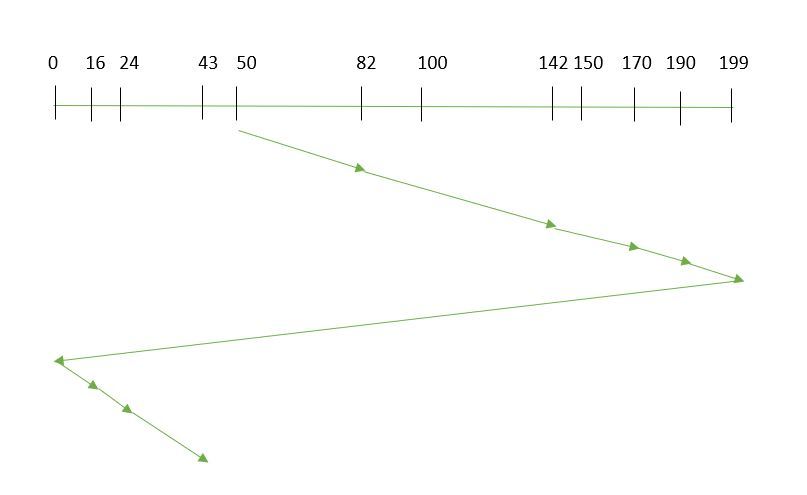
* Long waiting time for requests for locations just visited by disk arm

1. **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

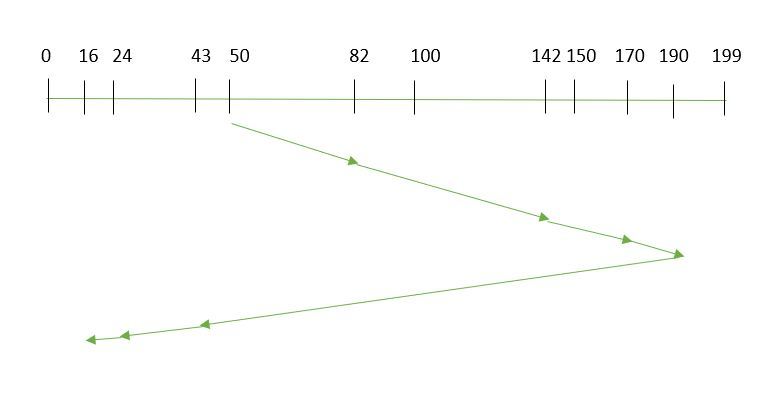
Advantages:

* 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”.**



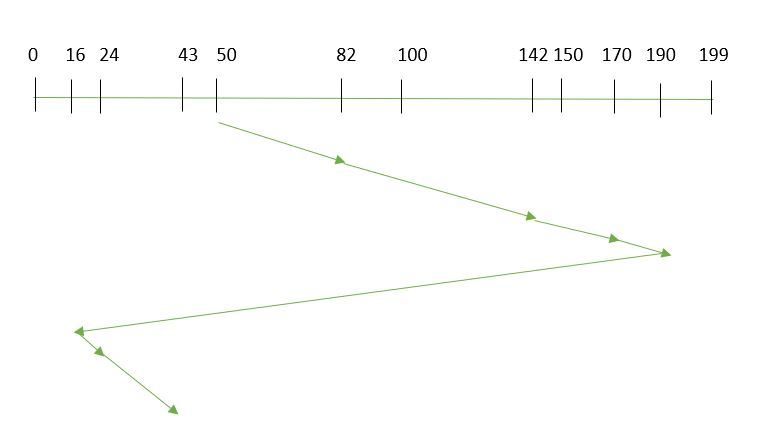


So, the seek time is calculated as:

1. =(190-50)+(190-16)   
   =314
2. **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:

1. =(190-50)+(190-16)+(43-16)   
   =341
2. **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 perfect. Which is why it is usually used for and analysis and simulation.
3. **LIFO**– In LIFO (Last In, First Out) algorithm, 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.
4. **N-STEP SCAN** – It is also known as 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 requests and this way all get requests get a guaranteed service   
   **Advantages**
   * It eliminates starvation of requests completely
5. **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).

Exercise

**1)** Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing \_\_\_\_\_\_\_\_\_\_\_\_ number of requests. (GATE CS 2014

(A) 1

(B) 2

(C) 3

(D) 4

See [this](https://www.geeksforgeeks.org/gate-gate-cs-2014-set-1-question-29/) for solution.

**2)** Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs? (GATE CS 2004)

(A) 50%

(B) 40%

(C) 25%

(D) 0%

See [this](https://www.geeksforgeeks.org/gate-gate-cs-2004-question-12/) for solution.

**3)** Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is \_\_\_\_\_\_\_\_\_ tracks

(A) 8

(B) 9

(C) 10

(D) 11

See [this](https://www.geeksforgeeks.org/gate-gate-cs-2015-set-1-question-40/) for solution.

**4)** Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50 × 10^6 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller’s transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is \_\_\_\_\_\_\_\_\_\_\_\_\_

# Disk Scheduling Algorithms in OS (Operating System)

As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss some important terms related to disk scheduling.

### **Seek Time**

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

### **Rotational Latency**

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

### **Transfer Time**

It is the time taken to transfer the data.

### **Disk Access Time**

Disk access time is given as,

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

### **Disk Response Time**

It is the average of time spent by each request waiting for the IO operation.

### **Purpose of Disk Scheduling**

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

### **Goal of Disk Scheduling Algorithm**

* Fairness
* High throughout
* Minimal traveling head time

### **Disk Scheduling Algorithms**

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

* FCFS scheduling algorithm
* SSTF (shortest seek time first) algorithm
* SCAN scheduling
* C-SCAN scheduling
* LOOK Scheduling
* C-LOOK scheduling

# FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

## **Disadvantages**

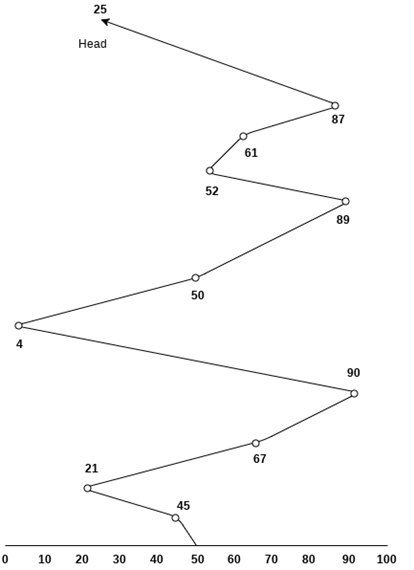
* The scheme does not optimize the seek time.
* The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

### **Example**

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

### **Solution**



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

# SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.

## **Disadvantages**

* It may cause starvation for some requests.
* Switching direction on the frequent basis slows the working of algorithm.
* It is not the most optimal algorithm.

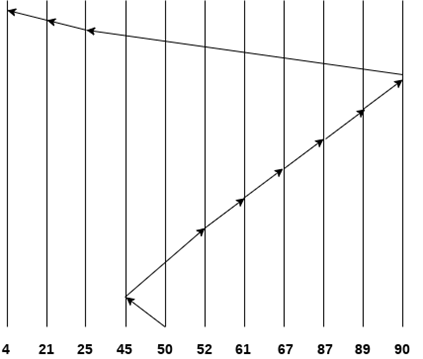
### **Example**

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

### **Solution:**



Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

# SCAN and C-SCAN algorithm

## **Scan Algorithm**

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.

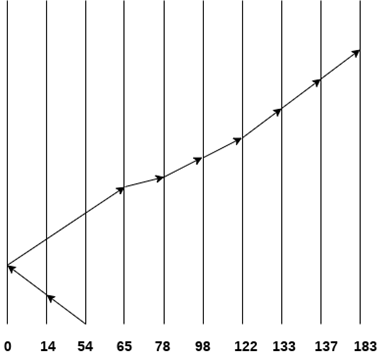
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

### **Example**

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

## **C-SCAN algorithm**

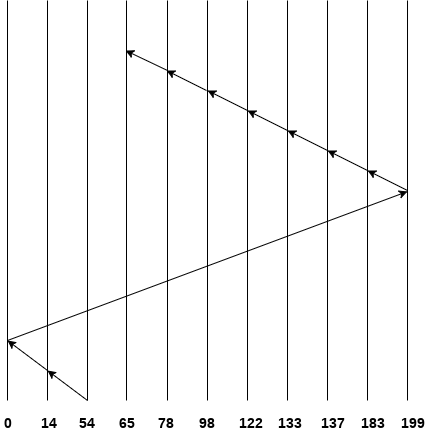
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

### **Example**

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

# Look Scheduling

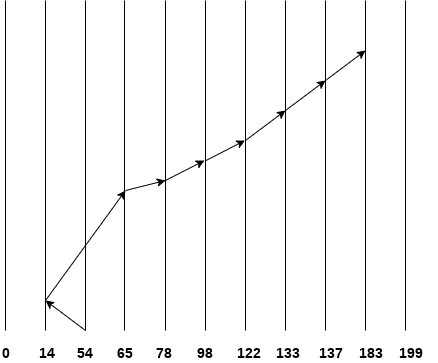
It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

### **Example**

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

## **C Look Scheduling**

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

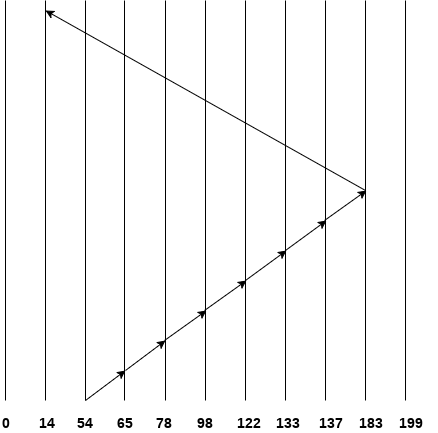
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

### **Example**

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

# Spooling in Operating System

In Operating System, we had to give the input to the CPU, and the CPU executes the instructions and finally gives the output. But there was a problem with this approach. In a normal situation, we have to deal with many processes, and we know that the time taken in the I/O operation is very large compared to the time taken by the CPU for the execution of the instructions. So, in the old approach, one process will give the input with the help of an input device, and during this time, the CPU is in an idle state.

Then the CPU executes the instruction, and the output is again given to some output device, and at this time, the CPU is also in an idle state. After showing the output, the next process starts its execution. So, most of the time, the CPU is idle, which is the worst condition that we can have in Operating Systems. Here, the concept of Spooling comes into play.

### **What is Spooling**

Spooling is a process in which data is temporarily held to be used and executed by a device, program, or system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.

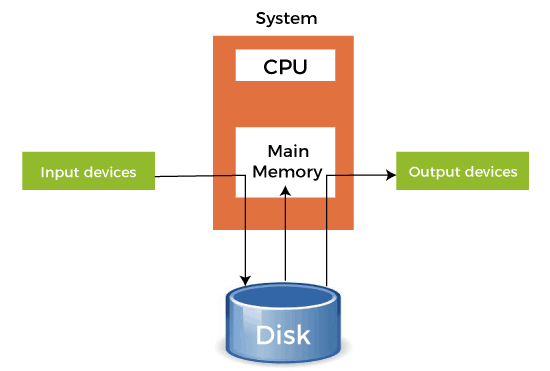
SPOOL is an acronym for **simultaneous peripheral operations online**. Generally, the spool is maintained on the computer's physical memory, buffers, or the I/O device-specific interrupts. The spool is processed in ascending order, working based on a FIFO (first-in, first-out) algorithm.

Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices. An operating system does the following activities related to the distributed environment:

* Handles I/O device data spooling as devices have different data access rates.
* Maintains the spooling buffer, which provides a waiting station where data can rest while the slower device catches up.
* Maintains parallel computation because of the spooling process as a computer can perform I/O in parallel order. It becomes possible to have the computer read data from a tape, write data to disk, and write out to a tape printer while it is doing its computing task.

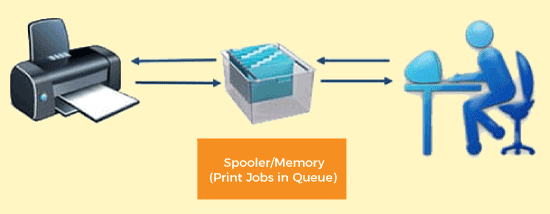
### **How Spooling Works in Operating System**

In an operating system, spooling works in the following steps, such as:

1. Spooling involves creating a buffer called SPOOL, which is used to hold off jobs and data till the device in which the SPOOL is created is ready to make use and execute that job or operate on the data.
2. When a faster device sends data to a slower device to perform some operation, it uses any secondary memory attached as a SPOOL buffer. This data is kept in the SPOOL until the slower device is ready to operate on this data. When the slower device is ready, then the data in the SPOOL is loaded onto the main memory for the required operations.  
   
3. Spooling considers the entire secondary memory as a huge buffer that can store many jobs and data for many operations. The advantage of Spooling is that it can create a queue of jobs that execute in FIFO order to execute the jobs one by one.
4. A device can connect to many input devices, which may require some operation on their data. So, all of these input devices may put their data onto the secondary memory (SPOOL), which can then be executed one by one by the device. This will make sure that the CPU is not idle at any time. So, we can say that Spooling is a combination of buffering and queuing.
5. After the CPU generates some output, this output is first saved in the main memory. This output is transferred to the secondary memory from the main memory, and from there, the output is sent to the respective output devices.

### **Example of Spooling**

The biggest example of Spooling is **printing**. The documents which are to be printed are stored in the SPOOL and then added to the queue for printing. During this time, many processes can perform their operations and use the CPU without waiting while the printer executes the printing process on the documents one-by-one.



Many features can also be added to the Spooling printing process, like setting priorities or notification when the printing process has been completed or selecting the different types of paper to print on according to the user's choice.

### **Advantages of Spooling**

Here are the following advantages of spooling in an operating system, such as:

* The number of I/O devices or operations does not matter. Many I/O devices can work together simultaneously without any interference or disruption to each other.
* In spooling, there is no interaction between the I/O devices and the CPU. That means there is no need for the CPU to wait for the I/O operations to take place. Such operations take a long time to finish executing, so the CPU will not wait for them to finish.
* CPU in the idle state is not considered very efficient. Most protocols are created to utilize the CPU efficiently in the minimum amount of time. In spooling, the CPU is kept busy most of the time and only goes to the idle state when the queue is exhausted. So, all the tasks are added to the queue, and the CPU will finish all those tasks and then go into the idle state.
* It allows applications to run at the speed of the CPU while operating the I/O devices at their respective full speeds.

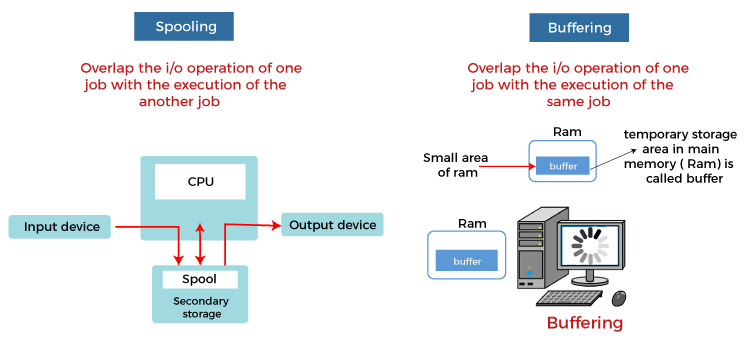
### **Disadvantages of Spooling**

In an operating system, spooling has the following disadvantages, such as:

* Spooling requires a large amount of storage depending on the number of requests made by the input and the number of input devices connected.
* Because the SPOOL is created in the secondary storage, having many input devices working simultaneously may take up a lot of space on the secondary storage and thus increase disk traffic. This results in the disk getting slower and slower as the traffic increases more and more.
* Spooling is used for copying and executing data from a slower device to a faster device. The slower device creates a SPOOL to store the data to be operated upon in a queue, and the CPU works on it. This process in itself makes Spooling futile to use in real-time environments where we need real-time results from the CPU. This is because the input device is slower and thus produces its data at a slower pace while the CPU can operate faster, so it moves on to the next process in the queue. This is why the final result or output is produced at a later time instead of in real-time.

### **Difference between Spooling and Buffering**

Spooling and buffering are the two ways by which I/O subsystems improve the performance and efficiency of the computer by using a storage space in the main memory or on the disk.



The basic difference between Spooling and Buffering is that Spooling overlaps the I/O of one job with the execution of another job. In comparison, the buffering overlaps the I/O of one job with the execution of the same job. Below are some more differences between Spooling and Buffering, such as:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Terms** | **Spooling** | **Buffering** |  | | | |
| Definition | Spooling, an acronym of Simultaneous Peripheral Operation Online (SPOOL), puts data into a temporary working area to be accessed and processed by another program or resource. | Buffering is an act of storing data temporarily in the buffer. It helps in matching the speed of the data stream between the sender and receiver. | | | |  |
| Resource requirement | Spooling requires less resource management as different resources manage the process for specific jobs. | Buffering requires more resource management as the same resource manages the process of the same divided job. | | |  | |
| Internal implementation | Spooling overlaps the input and output of one job with the computation of another job. | Buffering overlaps the input and output of one job with the computation of the same job. | |  | | |
| Efficient | Spooling is more efficient than buffering. | Buffering is less efficient than spooling. | |  | | |
| Processor | Spooling can also process data at remote sites. The spooler only has to notify when a process gets completed at the remote site to spool the next process to the remote side device. | Buffering does not support remote processing. | |  | | |
| Size on memory | It considers the disk as a huge spool or buffer. | Buffer is a limited area in the main memory. | |  | | |

File Systems

File system is the part of the operating system which is responsible for file management. It provides a mechanism to store the data and access to the file contents including data and programs. Some Operating systems treats everything as a file for example Ubuntu.

The File system takes care of the following issues

* **File Structure**

We have seen various data structures in which the file can be stored. The task of the file system is to maintain an optimal file structure.

* **Recovering Free space**

Whenever a file gets deleted from the hard disk, there is a free space created in the disk. There can be many such spaces which need to be recovered in order to reallocate them to other files.

* **disk space assignment to the files**

The major concern about the file is deciding where to store the files on the hard disk. There are various disks scheduling algorithm which will be covered later in this tutorial.

* **tracking data location**

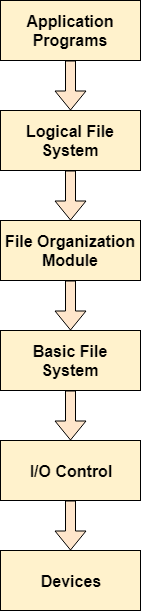
A File may or may not be stored within only one block. It can be stored in the non contiguous blocks on the disk. We need to keep track of all the blocks on which the part of the files reside.

File System Structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



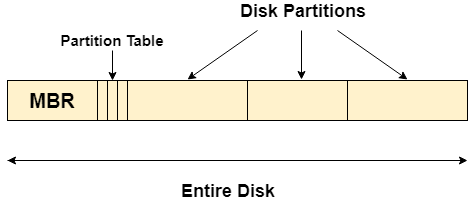
* When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
* Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
* Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
* I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

# Master Boot Record (MBR)

Master boot record is the information present in the first sector of any hard disk. It contains the information regarding how and where the Operating system is located in the hard disk so that it can be booted in the RAM.

MBR is sometimes called master partition table because it includes a partition table which locates every partition in the hard disk.

Master boot record (MBR) also includes a program which reads the boot sector record of the partition that contains operating system.



## **What happens when you turn on your computer?**

Due to the fact that the main memory is volatile, when we turn on our computer, CPU

cannot access the main memory directly. However, there is a special program called as BIOS stored in ROM is accessed for the first time by the CPU.

BIOS contains the code, by executing which, the CPU access the very first partition of hard disk that is MBR. It contains a partition table for all the partitions of the hard disk.

Since, MBR contains the information about where the operating system is being stored and it also contains a program which can read the boot sector record of the partition, hence the CPU fetches all this information and load the operating system into the main memory.

On Disk Data Structures

There are various on disk data structures that are used to implement a file system. This structure may vary depending upon the operating system.

1. **Boot Control Block**

Boot Control Block contains all the information which is needed to boot an operating system from that volume. It is called boot block in UNIX file system. In NTFS, it is called the partition boot sector.

1. **Volume Control Block**

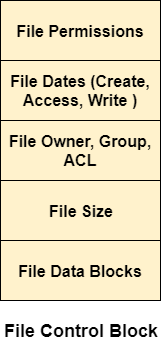
Volume control block all the information regarding that volume such as number of blocks, size of each block, partition table, pointers to free blocks and free FCB blocks. In UNIX file system, it is known as super block. In NTFS, this information is stored inside master file table.

1. **Directory Structure (per file system)**

A directory structure (per file system) contains file names and pointers to corresponding FCBs. In UNIX, it includes inode numbers associated to file names.

1. **File Control Block**

File Control block contains all the details about the file such as ownership details, permission details, file size,etc. In UFS, this detail is stored in inode. In NTFS, this information is stored inside master file table as a relational database structure. A typical file control block is shown in the image below.



In Memory Data Structure

Till now, we have discussed the data structures that are required to be present on the hard disk in order to implement file systems. Here, we will discuss the data structures required to be present in memory in order to implement the file system.

The in-memory data structures are used for file system management as well as performance improvement via caching. This information is loaded on the mount time and discarded on ejection.

1. **In-memory Mount Table**

In-memory mount table contains the list of all the devices which are being mounted to the system. Whenever the connection is maintained to a device, its entry will be done in the mount table.

1. **In-memory Directory structure cache**

This is the list of directory which is recently accessed by the CPU. The directories present in the list can also be accessed in the near future so it will be better to store them temporally in cache.

1. **System-wide open file table**

This is the list of all the open files in the system at a particular time. Whenever the user open any file for reading or writing, the entry will be made in this open file table.

1. **Per process Open file table**

It is the list of open files subjected to every process. Since there is already a list which is there for every open file in the system thereforeIt only contains Pointers to the appropriate entry in the system wide table.

# Directory Implementation

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

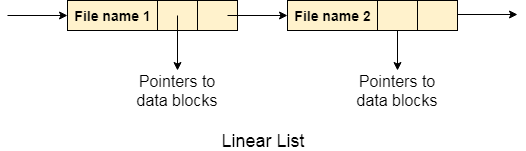
The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used in these days.

### **1. Linear List**

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

**Characteristics**

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



### **2. Hash Table**

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.

# os directory implementation hash tableAllocation Methods

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

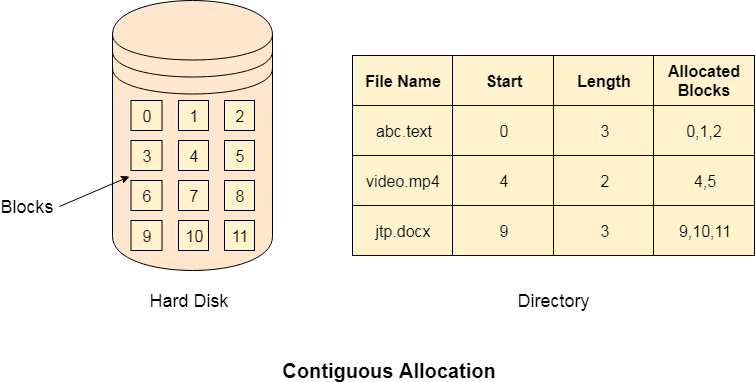
There are following methods which can be used for allocation.

1. Contiguous Allocation.
2. Extents
3. Linked Allocation
4. Clustering
5. FAT
6. Indexed Allocation
7. Linked Indexed Allocation
8. Multilevel Indexed Allocation
9. Inode

# Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



## **Advantages**

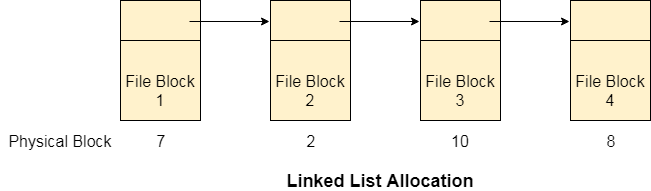
1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

## **Disadvantages**

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

# Linked List Allocation

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



## **Advantages**

1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

## **Disadvantages**

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

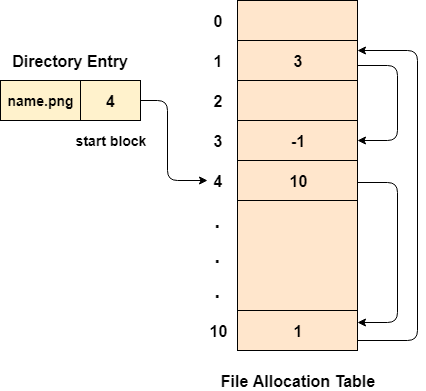
# File Allocation Table

The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.

File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.

File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.

It simply accesses the file allocation table, read the desired block entry from there and access that block. This is the way by which the random access is accomplished by using FAT. It is used by MS-DOS and pre-NT Windows versions.



## **Advantages**

1. Uses the whole disk block for data.
2. A bad disk block doesn't cause all successive blocks lost.
3. Random access is provided although its not too fast.
4. Only FAT needs to be traversed in each file operation.

## **Disadvantages**

1. Each Disk block needs a FAT entry.
2. FAT size may be very big depending upon the number of FAT entries.
3. Number of FAT entries can be reduced by increasing the block size but it will also increase Internal Fragmentation.

# Indexed Allocation

## **Limitation of FAT**

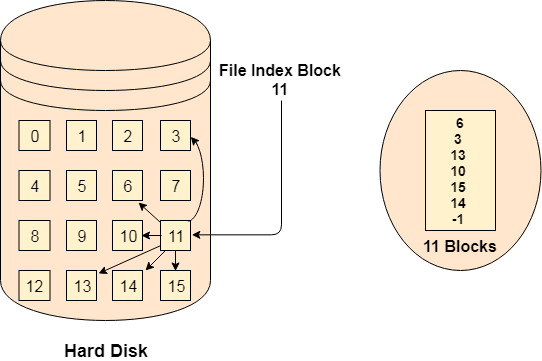
Limitation in the existing technology causes the evolution of a new technology. Till now, we have seen various allocation methods; each of them was carrying several advantages and disadvantages.

File allocation table tries to solve as many problems as possible but leads to a drawback. The more the number of blocks, the more will be the size of FAT.

Therefore, we need to allocate more space to a file allocation table. Since, file allocation table needs to be cached therefore it is impossible to have as many space in cache. Here we need a new technology which can solve such problems.

## **Indexed Allocation Scheme**

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



## **Advantages**

1. Supports direct access
2. A bad data block causes the lost of only that block.

## **Disadvantages**

1. A bad index block could cause the lost of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.
4. More pointer overhead

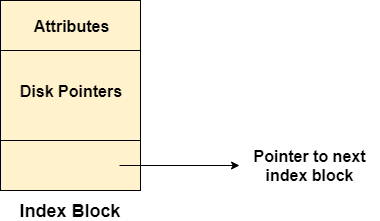
# Linked Index Allocation

## **Single level linked Index Allocation**

In index allocation, the file size depends on the size of a disk block. To allow large files, we have to link several index blocks together. In linked index allocation,

* Small header giving the name of the file
* Set of the first 100 block addresses
* Pointer to another index block

For the larger files, the last entry of the index block is a pointer which points to another index block. This is also called as linked schema.



**Advantage:** It removes file size limitations

**Disadvantage:** Random Access becomes a bit harder

## **Multilevel Index Allocation**

In Multilevel index allocation, we have various levels of indices. There are outer level index blocks which contain the pointers to the inner level index blocks and the inner level index blocks contain the pointers to the file data.

* The outer level index is used to find the inner level index.
* The inner level index is used to find the desired data block.

**Advantage:** Random Access becomes better and efficient.

**Disadvantage:** Access time for a file will be higher.

# os linked index allocation index blockInode

In UNIX based operating systems, each file is indexed by an Inode. Inode are the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of Inodes in the file system.

An Inode includes the following information

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.

# os InodeProtection in Operating System

Protection is especially important in a multiuser environment when multiple users use computer resources such as CPU, memory, etc. It is the operating system's responsibility to offer a mechanism that protects each process from other processes. In a multiuser environment, all assets that require protection are classified as objects, and those that wish to access these objects are referred to as subjects. The operating system grants different 'access rights' to different subjects.



In this article, you will learn the protection in the operating system with its needs, goals, and authentication.

## **What is Protection in Operating System?**

A mechanism that controls the access of programs, processes, or users to the resources defined by a computer system is referred to as protection. You may utilize protection as a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files.

It needs the protection of computer resources like the software, memory, processor, etc. Users should take protective measures as a helper to multiprogramming OS so that multiple users may safely use a common logical namespace like a directory or data. Protection may be achieved by maintaining confidentiality, honesty and availability in the OS. It is critical to secure the device from unauthorized access, viruses, worms, and other malware.

## **Need of Protection in Operating System**

Various needs of protection in the operating system are as follows:

1. There may be security risks like unauthorized reading, writing, modification, or preventing the system from working effectively for authorized users.
2. It helps to ensure data security, process security, and program security against unauthorized user access or program access.
3. It is important to ensure no access rights' breaches, no viruses, no unauthorized access to the existing data.
4. Its purpose is to ensure that only the systems' policies access programs, resources, and data.

## **Goals of Protection in Operating System**

Various goals of protection in the operating system are as follows:

1. The policies define how processes access the computer system's resources, such as the CPU, memory, software, and even the operating system. It is the responsibility of both the operating system designer and the app programmer. Although, these policies are modified at any time.
2. Protection is a technique for protecting data and processes from harmful or intentional infiltration. It contains protection policies either established by itself, set by management or imposed individually by programmers to ensure that their programs are protected to the greatest extent possible.
3. It also provides a multiprogramming OS with the security that its users expect when sharing common space such as files or directories.

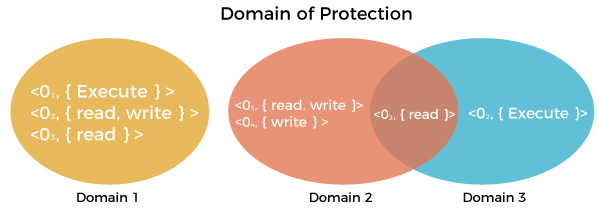
## **Role of Protection in Operating System**

Its main role is to provide a mechanism for implementing policies that define the use of resources in a computer system. Some rules are set during the system's design, while others are defined by system administrators to secure their files and programs.

Every program has distinct policies for using resources, and these policies may change over time. Therefore, system security is not the responsibility of the system's designer, and the programmer must also design the protection technique to protect their system against infiltration.

## **Domain of Protection**

Various domains of protection in operating system are as follows:

1. The protection policies restrict each process's access to its resource handling. A process is obligated to use only the resources necessary to fulfil its task within the time constraints and in the mode in which it is required. It is a process's protected domain.
2. Processes and objects are abstract data types in a computer system, and these objects have operations that are unique to them. A domain component is defined as **<object, {set of operations on object}>**.  
   
3. Each domain comprises a collection of objects and the operations that may be implemented on them. A domain could be made up of only one process, procedure, or user. If a domain is linked with a procedure, changing the domain would mean changing the procedure ID. Objects may share one or more common operations.

### **Association between Process and Domain**

When processes have the necessary access rights, they can switch from one domain to another. It could be of two types, as shown below.

**1. Fixed or Static**

In a fixed association, all access rights could be given to processes at the start. However, the results in a large number of access rights for domain switching. As a result, a technique of changing the domain's contents is found dynamically.

**2. Changing or dynamic**

A process may switch dynamically and creating a new domain in the process.

## **Security measures of Operating System**

There are various security measures of the operating system that the users may take. Some of them are as follows:

1. The network used for file transfers must be secure at all times. During the transfer, no alien software should be able to harvest information from the network. It is referred to as network sniffing, and it could be avoided by implementing encrypted data transfer routes. Moreover, the OS should be capable of resisting forceful or even accidental violations.
2. Passwords are a good authentication method, but they are the most common and vulnerable. It is very easy to crack passwords.
3. Security measures at various levels are put in place to prevent malpractices, like no one being allowed on the premises or access to the systems.
4. The best authentication techniques include a username-password combination, eye retina scan, fingerprint, or even user cards to access the system.

## **System Authentication**

**One-time passwords, encrypted passwords,** and **cryptography** are used to create a strong password and a formidable authentication source.

**1. One-time Password**

It is a way that is unique at every login by the user. It is a combination of two passwords that allow the user access. The system creates a random number, and the user supplies a matching one. An algorithm generates a random number for the system and the user, and the output is matched using a common function.

**2. Encrypted Passwords**

It is also a very effective technique of authenticating access. Encrypted data is passed via the network, which transfers and checks passwords, allowing data to pass without interruption or interception.

**3. Cryptography**

It's another way to ensure that unauthorized users can't access data transferred over a network. It aids in the data secure transmission. It introduces the concept of a key to protecting the data. The key is crucial in this situation. When a user sends data, he encodes it using a computer that has the key, and the receiver must decode the data with the same key. As a result, even if the data is stolen in the middle of the process, there's a good possibility the unauthorized user won't be able to access it.

# File Systems in Operating System

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From user’s perspective a file is the smallest allotment of logical secondary storage.

**The name  of the file is divided into two parts as shown below:**

* name
* extension, separated by a period.

### Files attributes and its operations:

|  |  |  |
| --- | --- | --- |
| Attributes | Types | Operations |
| Name | Doc | Create |
| Type | Exe | Open |
| Size | Jpg | Read |
| Creation Data | Xis | Write |
| Author | C | Append |
| Last Modified | Java | Truncate |
| protection | class | Delete |
|  |  | Close |

|  |  |  |
| --- | --- | --- |
| File type | Usual extension | Function |
| Executable | exe, com, bin | Read to run machine language program |
| Object | obj, o | Compiled, machine language not linked |
| Source Code | C, java, pas, asm, a | Source code in various languages |
| Batch | bat, sh | Commands to the command interpreter |
| Text | txt, doc | Textual data, documents |
| Word Processor | wp, tex, rrf, doc | Various word processor formats |
| Archive | arc, zip, tar | Related files grouped into one compressed file |
| Multimedia | mpeg, mov, rm | For containing audio/video information |
| Markup | xml, html, tex | It is the textual data and documents |
| Library | lib, a ,so, dll | It contains libraries of routines for programmers |
| Print or View | gif, pdf, jpg | It is a format for printing or viewing a ASCII or binary file. |

**FILE DIRECTORIES:**

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

**Information contained in a device directory are:**

* Name
* Type
* Address
* Current length
* Maximum length
* Date last accessed
* Date last updated
* Owner id
* Protection information

**Operation performed on directory are:**

* Search for a file
* Create a file
* Delete a file
* List a directory
* Rename a file
* Traverse the file system

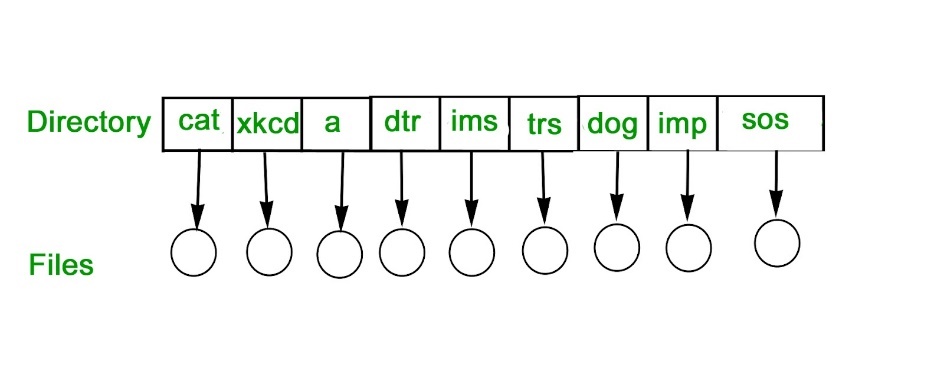
**Advantages of maintaining directories are:**

* **Efficiency:** A file can be located more quickly.
* **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
* **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

**SINGLE-LEVEL DIRECTORY**

In this a single directory is maintained for all the users.

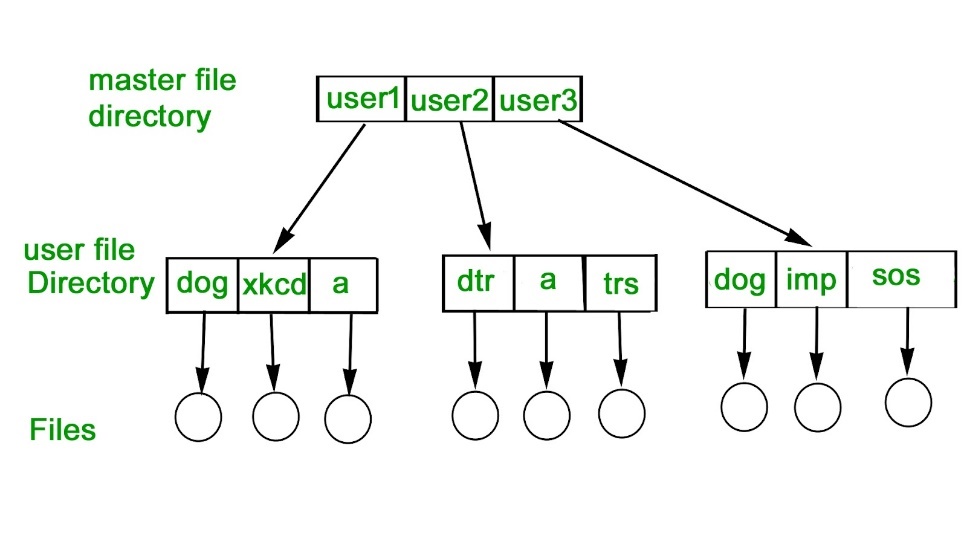
* **Naming problem:** Users cannot have same name for two files.
* **Grouping problem:** Users cannot group files according to their need.



**TWO-LEVEL DIRECTORY**

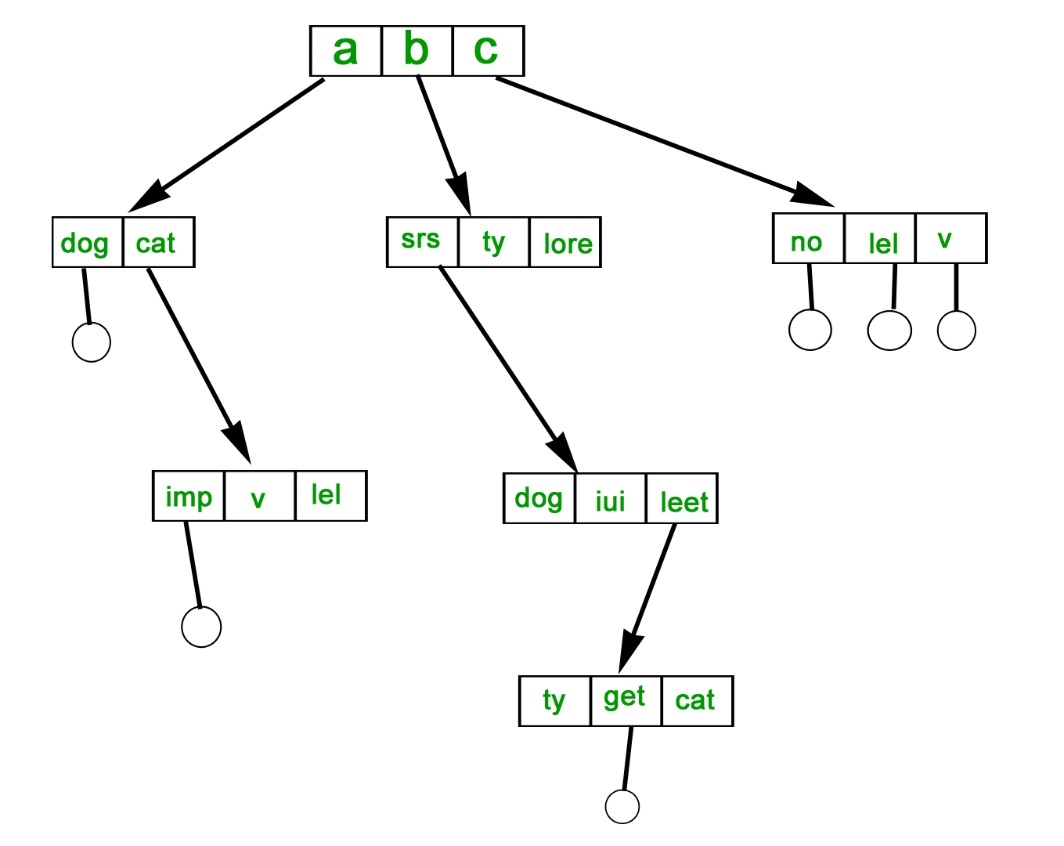
In this separate directories for each user is maintained.

* Path name:Due to two levels there is a path name for every file to locate that file.
* Now,we can have same file name for different user.
* Searching is efficient in this method.



**TREE-STRUCTURED DIRECTORY :**

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



**FILE ALLOCATION METHODS** **:**

**1. Continuous Allocation –**

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.

# What is a File ?

A file can be defined as a data structure which stores the sequence of records. Files are stored in a file system, which may exist on a disk or in the main memory. Files can be simple (plain text) or complex (specially-formatted).

The collection of files is known as Directory. The collection of directories at the different levels, is known as File System.



## **Attributes of the File**

**1.Name**

Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

**2.Identifier**

Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**

**3.Type**

In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.

**4.Location**

In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

**5.Size**

The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.

**6.Protection**

The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.

**7.Time and Date**

Every file carries a time stamp which contains the time and date on which the file is last modified.

# File Systems in Operating System: Structure, Attributes, Types

## What is File System?

A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes. It is a method of data collection that is used as a medium for giving input and receiving output from that program.

In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user. Every File has a logical location where they are located for storage and retrieval.

In this operating system tutorial, you will learn:

* [What is File System?](https://www.guru99.com/file-systems-operating-system.html#what-is-file-system)
* [Objective of File management System](https://www.guru99.com/file-systems-operating-system.html#objective-of-file-management-system)
* [Properties of a File System](https://www.guru99.com/file-systems-operating-system.html#properties-of-a-file-system)
* [File structure](https://www.guru99.com/file-systems-operating-system.html#file-structure)
* [File Attributes](https://www.guru99.com/file-systems-operating-system.html#file-attributes)
* [File Type](https://www.guru99.com/file-systems-operating-system.html#file-type)
* [Functions of File](https://www.guru99.com/file-systems-operating-system.html#functions-of-file)
* [Commonly used terms in File systems](https://www.guru99.com/file-systems-operating-system.html#commonly-used-terms-in-file-systems)
* [File Access Methods](https://www.guru99.com/file-systems-operating-system.html#file-access-methods)
* [Space Allocation](https://www.guru99.com/file-systems-operating-system.html#space-allocation)
* [File Directories](https://www.guru99.com/file-systems-operating-system.html#file-directories)
* [File types- name, extension](https://www.guru99.com/file-systems-operating-system.html#file-types-name-extension)

## Objective of File management System

Here are the main objectives of the file management system:

* It provides I/O support for a variety of storage device types.
* Minimizes the chances of lost or destroyed data
* Helps OS to standardized I/O interface routines for user processes.
* It provides I/O support for multiple users in a multiuser systems environment.

## Properties of a File System

Here, are important properties of a file system:

* Files are stored on disk or other storage and do not disappear when a user logs off.
* Files have names and are associated with access permission that permits controlled sharing.
* Files could be arranged or more complex structures to reflect the relationship between them.

## File structure

A File Structure needs to be predefined format in such a way that an operating system understands. It has an exclusively defined structure, which is based on its type.

Three types of files structure in OS:

* A text file: It is a series of characters that is organized in lines.
* An object file: It is a series of bytes that is organized into blocks.
* A source file: It is a series of functions and processes.

**Reading Suggestion:-**[What is Operating System? Explain Types of OS, Features and Examples.](https://www.guru99.com/operating-system-tutorial.html)

## File Attributes

A file has a name and data. Moreover, it also stores meta information like file creation date and time, current size, last modified date, etc. All this information is called the attributes of a file system.

Here, are some important File attributes used in OS:

* **Name:** It is the only information stored in a human-readable form.
* **Identifier**: Every file is identified by a unique tag number within a file system known as an identifier.
* **Location:** Points to file location on device.
* **Type:** This attribute is required for systems that support various types of files.
* **Size**. Attribute used to display the current file size.
* **Protection**. This attribute assigns and controls the access rights of reading, writing, and executing the file.
* **Time, date and security:** It is used for protection, security, and also used for monitoring

## File Type

It refers to the ability of the operating system to differentiate various types of files like text files, binary, and source files. However, Operating systems like MS\_DOS and UNIX has the following type of files:

### Character Special File

It is a hardware file that reads or writes data character by character, like mouse, printer, and more.

### Ordinary files

* These types of files stores user information.
* It may be text, executable programs, and databases.
* It allows the user to perform operations like add, delete, and modify.

### Directory Files

* Directory contains files and other related information about those files. Its basically a folder to hold and organize multiple files.

### Special Files

* These files are also called device files. It represents physical devices like printers, disks, networks, flash drive, etc.

## Functions of File

* Create file, find space on disk, and make an entry in the directory.
* Write to file, requires positioning within the file
* Read from file involves positioning within the file
* Delete directory entry, regain disk space.
* Reposition: move read/write position.

## Commonly used terms in File systems

### Field:

This element stores a single value, which can be static or variable length.

### DATABASE:

Collection of related data is called a database. Relationships among elements of data are explicit.

### FILES:

Files is the collection of similar record which is treated as a single entity.

### RECORD:

A Record type is a complex data type that allows the programmer to create a new data type with the desired column structure. Its groups one or more columns to form a new data type. These columns will have their own names and data type.

## File Access Methods

File access is a process that determines the way that files are accessed and read into memory. Generally, a single access method is always supported by operating systems. Though there are some operating system which also supports multiple access methods.

**Three file access methods are:**

* Sequential access
* Direct random access
* Index sequential access

### Sequential Access

In this type of file access method, records are accessed in a certain pre-defined sequence. In the sequential access method, information stored in the file is also processed one by one. Most compilers access files using this access method.

### Random Access

The random access method is also called direct random access. This method allow accessing the record directly. Each record has its own address on which can be directly accessed for reading and writing.

### Sequential Access

This type of accessing method is based on simple sequential access. In this access method, an index is built for every file, with a direct pointer to different memory blocks. In this method, the Index is searched sequentially, and its pointer can access the file directly. Multiple levels of indexing can be used to offer greater efficiency in access. It also reduces the time needed to access a single record.

## Space Allocation

In the Operating system, files are always allocated disk spaces.

Three types of space allocation methods are:

* Linked Allocation
* Indexed Allocation
* Contiguous Allocation

### Contiguous Allocation

In this method,

* Every file users a contiguous address space on memory.
* Here, the OS assigns disk address is in linear order.
* In the contiguous allocation method, external fragmentation is the biggest issue.

### Linked Allocation

In this method,

* Every file includes a list of links.
* The directory contains a link or pointer in the first block of a file.
* With this method, there is no external fragmentation
* This File allocation method is used for sequential access files.
* This method is not ideal for a direct access file.

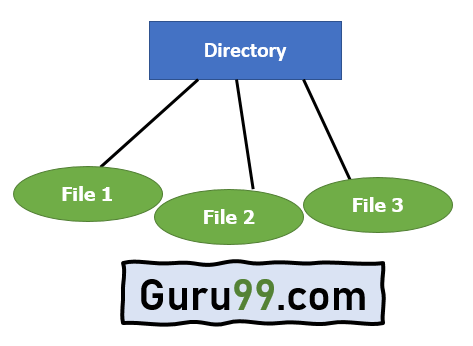
### Indexed Allocation

In this method,

* Directory comprises the addresses of index blocks of the specific files.
* An index block is created, having all the pointers for specific files.
* All files should have individual index blocks to store the addresses for disk space.

## File Directories

A single directory may or may not contain multiple files. It can also have sub-directories inside the main directory. Information about files is maintained by Directories. In Windows OS, it is called folders.



Single Level Directory

Following is the information which is maintained in a directory:

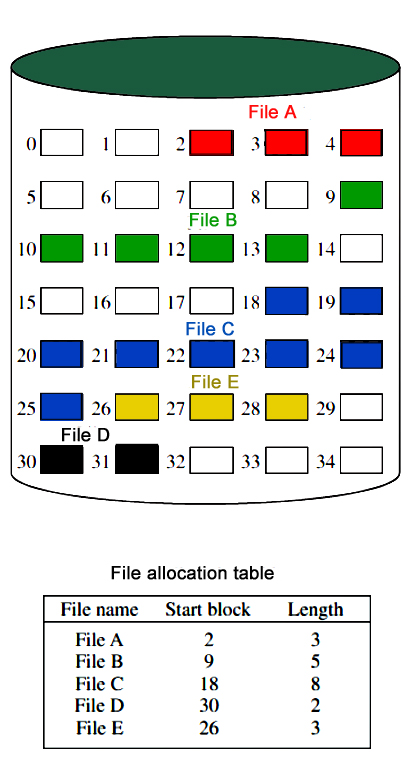
* **Name** The name which is displayed to the user.
* **Type**: Type of the directory.
* **Position**: Current next-read/write pointers.
* **Location**: Location on the device where the file header is stored.
* **Size**: Number of bytes, block, and words in the file.
* **Protection**: Access control on read/write/execute/delete.
* **Usage**: Time of creation, access, modification

## File types- name, extension

|  |  |  |
| --- | --- | --- |
| **File Type** | **Usual extension** | **Function** |
| Executable | exe, com, bin or none | ready-to-run machine- language program |
| Object | obj, o | complied, machine language, not linked |
| Source code | c. p, pas, 177, asm, a | source code in various languages |
| Batch | bat, sh | Series of commands to be executed |
| Text | txt, doc | textual data documents |
| Word processor | doc,docs, tex, rrf, etc. | various word-processor formats |
| Library | lib, h | libraries of routines |
| Archive | arc, zip, tar | related files grouped into one file, sometimes compressed. |

## Summary:

* A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes.
* It provides I/O support for a variety of storage device types.
* Files are stored on disk or other storage and do not disappear when a user logs off.
* A File Structure needs to be predefined format in such a way that an operating system understands it.
* File type refers to the ability of the operating system to differentiate different types of files like text files, binary, and source files.
* Create find space on disk and make an entry in the directory.
* Indexed Sequential Access method is based on simple sequential access
* In Sequential Access method records are accessed in a certain pre-defined sequence
* The random access method is also called direct random access
* Three types of space allocation methods are:
* Linked Allocation
* Indexed Allocation
* Contiguous Allocation
* Information about files is maintained by Directories



**Disadvantage –**

* External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. Compaction algorithm will be necessary to free up additional space on disk.
* Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

**2. Linked Allocation(Non-contiguous allocation) –**

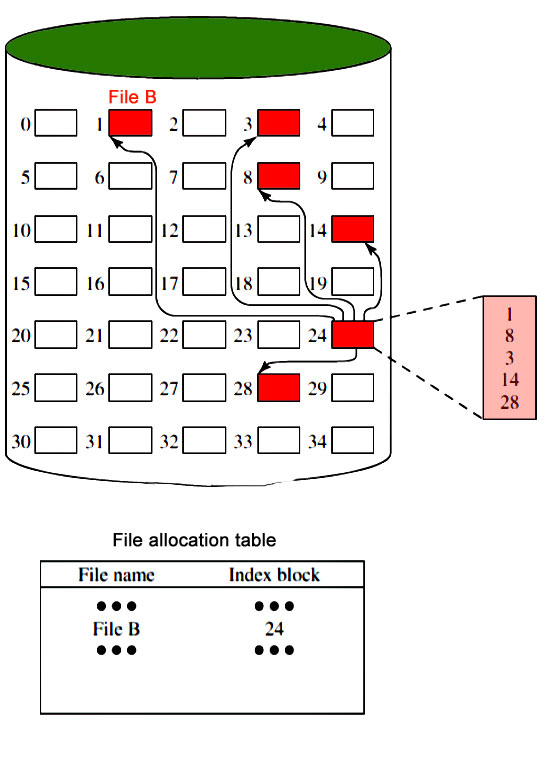
Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.

**Disadvantage –**

* Internal fragmentation exists in last disk block of file.
* There is an overhead of maintaining the pointer in every disk block.
* If the pointer of any disk block is lost, the file will be truncated.
* It supports only the sequential access of files.

**3. Indexed Allocation –**

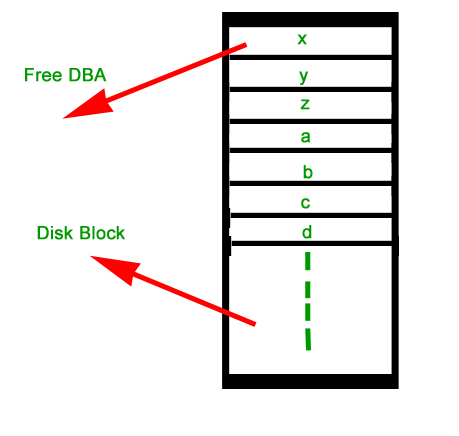
It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.



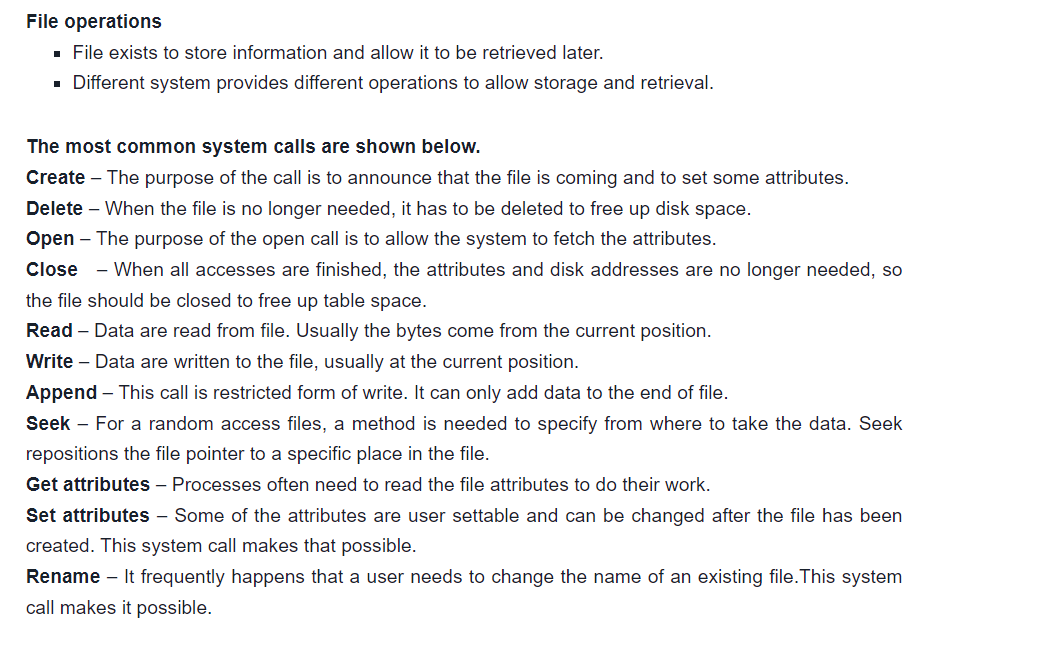
**Disk Free Space Management :**

Just as the space that is allocated to files must be managed ,so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques,it is necessary to know what blocks on the disk are available. Thus we need a disk allocation table in addition to a file allocation table.The following are the approaches used for free space management.

1. **Bit Tables** : This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use.   
   For example: 00011010111100110001   
   In this vector every bit correspond to a particular block and 0 implies that, that particular block is free and 1 implies that the block is already occupied. A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods. Another advantage is that it is as small as possible.
2. **Free Block List** : In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.



**File Operations**



There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

There are following methods which can be used for allocation.

1. Contiguous Allocation.
2. Extents
3. Linked Allocation
4. Clustering
5. FAT
6. Indexed Allocation
7. Linked Indexed Allocation
8. Multilevel Indexed Allocation
9. Inode

We will discuss three of the most used methods in detail.

## Overview

The File allocation methods in OS are different ways that are used for storing the file on the hard disk. There are 5 different ways in which we can store the files on the hard disk in such a manner that there is efficient utilization of disk space and the file can be accessed faster by the Operating System.

## Scope

The Scope of this article is as follows :

* This article explains the file allocation methods in the Operating System.
* The article is example oriented as all the file allocation methods are explained using examples.
* Along with that all the examples are explained using diagrams of file allocation.
* It covers the pros and cons of each file allocation and how to overcome some cons.

## What is File Allocation in OS?

Whenever a hard disk is formatted, a system has many small areas called blocks or sectors that are used to store any kind of file. File allocation methods are different ways by which the operating system stores information in memory blocks, thus allowing the hard drive to be utilized effectively and the file to be accessed. Below are the types of file allocation methods in the Operating System.

## Types of File Allocation Methods in Operating System.

* Contiguous File allocation
* Linked File Allocation
* Indexed File Allocation
* File Allocation Table (FAT)
* Inode

Let's have an in-detail explanation about each of them,

## Contiguous File Allocation.

First, let's understand the meaning of contiguous, here contiguous means adjacent or touching. Now let's understand what is contiguous file allocation.

#### What is Contiguous File allocation?

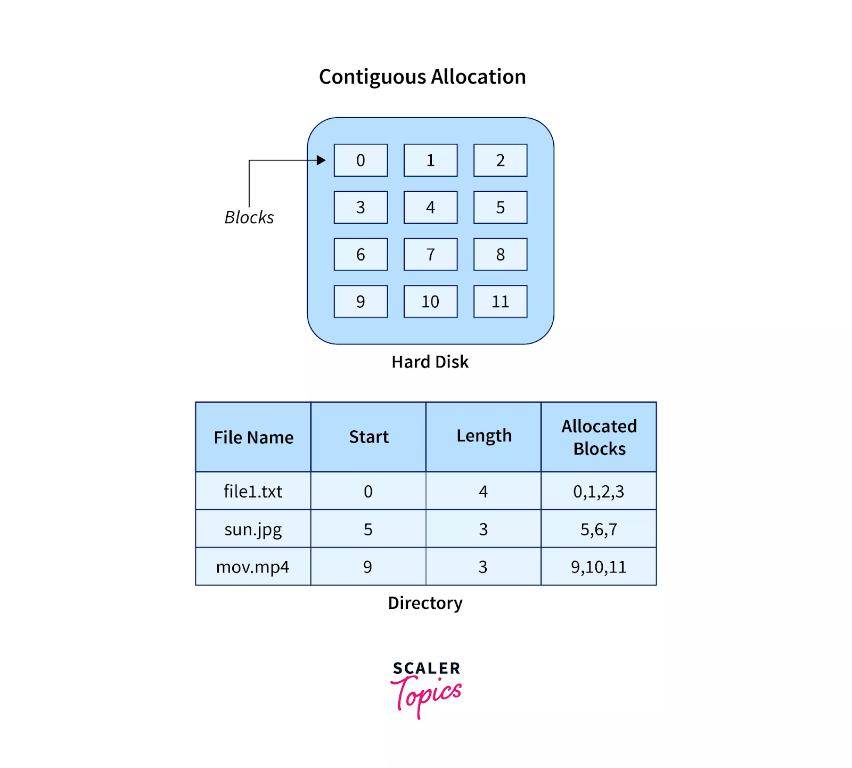
In contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are adjacent.

Assuming a file needs 'n' number of blocks in the disk and the file begins with a block at position'x', the next blocks to be assigned to it will be x+1,x+2,x+3,...,x+n-1 so that they are in a contiguous manner.

Let's understand this diagrammatically.

### Example

We have three different types of files that are stored in a contiguous manner on the hard disk.



In the above image on the left side, we have a memory diagram where we can see the blocks of memory. At first, we have a text file named file1.txt which is allocated using contiguous memory allocation, it starts with the memory block 0 and has a length of 4 so it takes the 4 contiguous blocks 0,1,2,3. Similarly, we have an image file and video file named sun.jpg and mov.mp4 respectively, which you can see in the directory that they are stored in the contiguous blocks. 5,6,7 and 9,10,11 respectively.

Here the directory has the entry of each file where it stores the address of the starting block and the required space in terms of the block of memory.

### Advantages and Disadvantages

Advantages

* It is very easy to implement.
* There is a minimum amount of seek time.
* The disk head movement is minimum.
* Memory access is faster.
* It supports sequential as well as direct access.

Disadvantages

* At the time of creation, the file size must be initialized.
* As it is pre-initialized, the size cannot increase. As
* Due to its constrained allocation, it is possible that the disk would fragment internally or externally.

## Linked File Allocation.

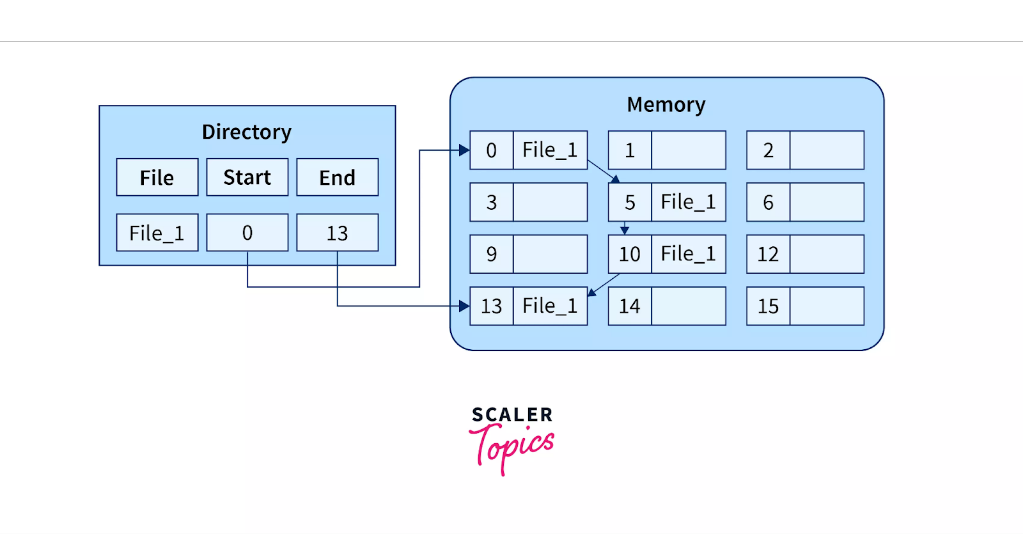
### What is Linked File Allocation?

The Linked file allocation overcomes the drawback of contiguous file allocation. Here the file which we store on the hard disk is stored in a scattered manner according to the space available on the hard disk. Now, you must be thinking about how the OS remembers that all the scattered blocks belong to the same file. So as the name linked File Allocation suggests, the pointers are used to point to the next block of the same file, therefore along with the entry of each file each block also stores the pointer to the next block.

Let's understand this better diagrammatically by taking an example.

### Example

Here we have one file which is stored using Linked File Allocation.



In the above image on the right, we have a memory diagram where we can see memory blocks. On the left side, we have a directory where we have the information like the address of the first memory block and the last memory block.

In this allocation, the starting block given is 0 and the ending block is 15, therefore the OS searches the empty blocks between 0 and 15 and stores the files in available blocks, but along with that it also stores the pointer to the next block in the present block. Hence it requires some extra space to store that link.

### Advantages and Disadvantages

Advantages

* There is no external fragmentation.
* The directory entry just needs the address of starting block.
* The memory is not needed in contiguous form, it is more flexible than contiguous file allocation.

Disadvantages

* It does not support random access or direct access.
* If pointers are affected so the disk blocks are also affected.
* Extra space is required for pointers in the block.

## Indexed File Allocation.

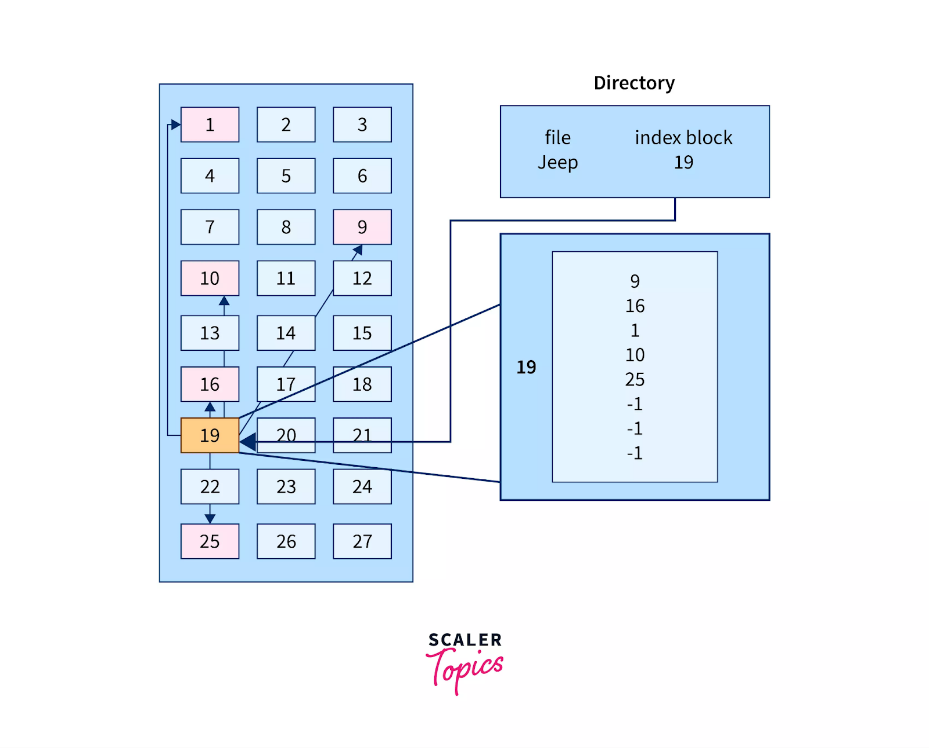
### What is Indexed File Allocation?

The indexed file allocation is somewhat similar to linked file allocation as indexed file allocation also uses pointers but the difference is here all the pointers are put together into one location which is called index block. That means we will get all the locations of blocks in one index file. The blocks and pointers were spread over the memory in the Linked Allocation method, where retrieval was accomplished by visiting each block sequentially. But here in indexed allocation, it becomes easier with the index block to retrieve.

Let's take an example to explain this better.

### Example

As shown in the diagram below block 19 is the index block which contains all the addresses of the file named text1. In order, the first storage block is 9, followed by 16, 1, then 10, and 25. The negative number -1 here denotes the empty index block list as the file text1 is still too small to fill more blocks.



### Advantages and Disadvantages

Advantages

* It reduces the possibilities of external fragmentation.
* Rather than accessing sequentially it has direct access to the block.

Disadvantages

* Here more pointer overhead is there.
* If we lose the index block we cannot access the complete file.
* It becomes heavy for the small files.
* It is possible that a single index block cannot keep all the pointers for some large files.

To resolve this issue, we can use the following approaches:

1. Linked scheme
2. Multilevel Index
3. Combined Scheme

### 1. Linked Scheme

If the file is big then more blocks are required so one index block is insufficient to store all the pointers, therefore to store the pointers two or more index blocks are used where these index boxes are connected using linked file allocation that is each index block stores the pointer to the next index block.

### 2. Multilevel Index

In this method, the multiple indexes blocks along with the levels of these blocks. Here, the level 1 block is used for pointing to the level 2 block which points to the blocks occupied by the file. These index blocks can be extended to three or more levels according to the size of the file.

### 3. Combined Scheme

In Combined Scheme, a special block is used to store all the information related to the file like name, authority, size, etc. The special block is called inode(information-node). Some space of this special block is used to store the information related to the field as mentioned above and the remaining space is used to store the addresses of blocks that contain the actual file. *The inode is explained further in detail.*

## File Allocation Table (FAT).

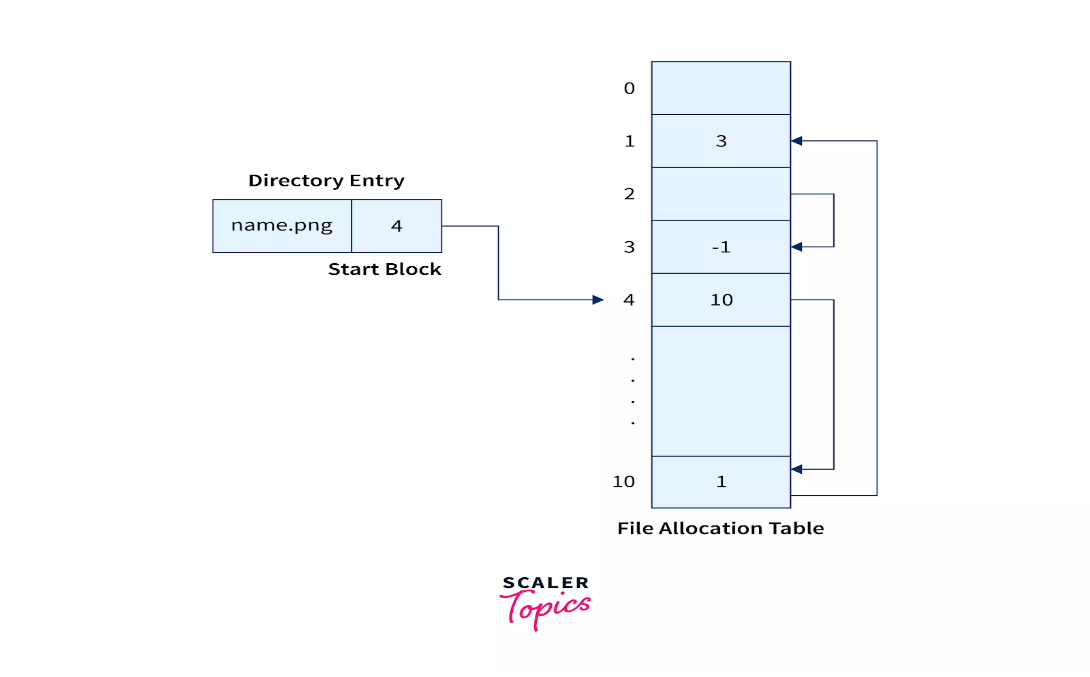
The File Allocation Table (FAT) overcomes the drawback of Linked File allocation. The random access of a particular block is not possible in the linked file allocation. To access a particular block it is necessary to access all its previous blocks. Let's see how the File Allocation Table works.

### Explanation

In the file allocation table, all disk block links are collected and maintained. Here the number of head seeks is reduced by caching the file allocation table so that the head does not need to go through all the disk blocks to access one particular block.

The whole process of randomly accessing any block using FAT is completed by reading the desired entry of a block from the file allocation table and accessing that particular block.

The diagrammatic representation of FAT is given below -



### Advantages and Disadvantages

Advantages

* Random Access to the block is possible in FAT.
* One bad/corrupted disk block cannot corrupt all the other blocks.
* It uses all the disk blocks for data as in linked file allocation it needs extra space for pointers.

Disadvantages

* If entries increase so the FAT size also increases.
* Each entry requires the FAT entry.
* If Entries increase the FAT size increases which increases the size of a block so there are chances of internal fragmentation.

## Inode.

In Operating systems based on UNIX, every file is indexed using Inode(information-node). The inode is the special disk block that is created with the creation of the file system. This special block is used to store all the information related to the file like name, authority, size, etc along with the pointers to the other blocks where the file is stored.

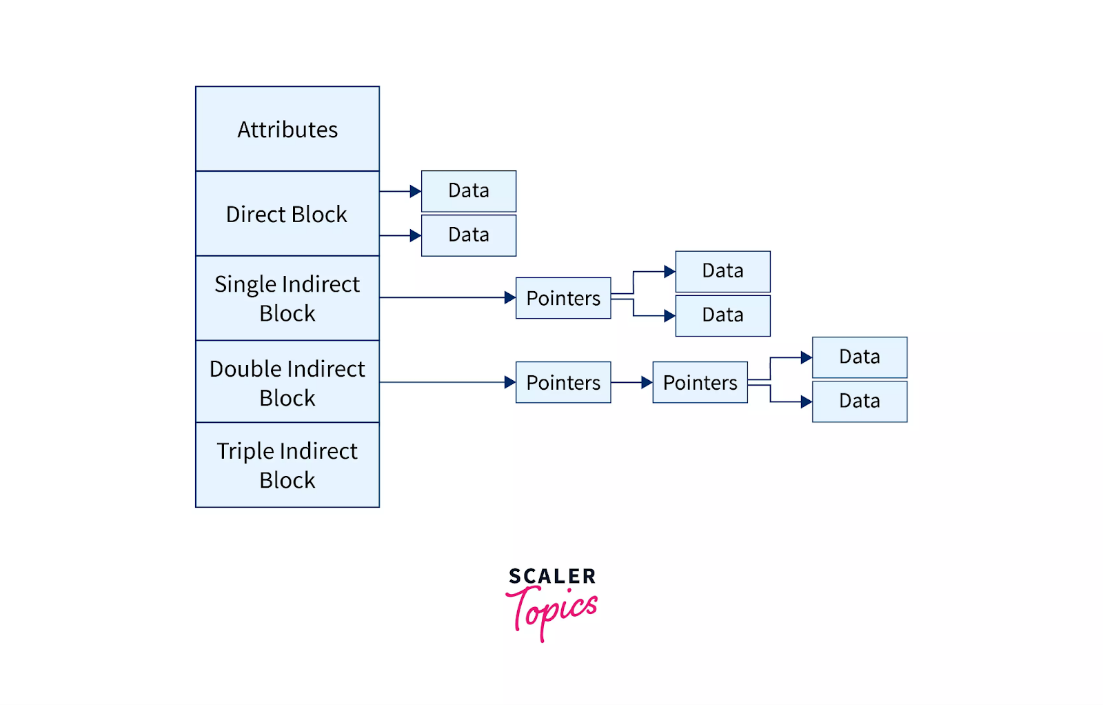
### Explanation

In this special block, some of its space is used to store the information related to the field as mentioned above and the remaining space is used to store the addresses of blocks that contain the actual file.

The first few pointers in Inode point to direct blocks, i.e they contain the addresses of the disk blocks containing the file data. A few pointers in inode point to the indirect blocks. There are three types of indirect blocks: *single indirect*, *double indirect*, and *triple indirect*.

A single indirect block contains nothing but the address of the block containing the file data, not the file itself as shown in the figure below. Furthermore, the double indirect block points to the pointers which again point to the pointers which point to the data blocks. Further, it goes in a similar way for triple indirect block.

The diagrammatic representation of Inode is given below -



### Advantages and Disadvantages

Advantages

* Accessibility of file becomes easy as all the information like metadata and block address is stored inside inode.
* Read-write and creation timestamps are stored inside the inode.
* Filenames do not affect inodes. In other words, a single file can be copied and renamed without losing its address.

Disadvantages

* All new files and folders will be rejected as soon as a file system runs out of inodes.
* Upon 100% utilization of inodes system will start to notice OS restarting, data loss, applications crashing, and more OS-related issues.

## Conclusion.

Let's recall what we have learned in this article,

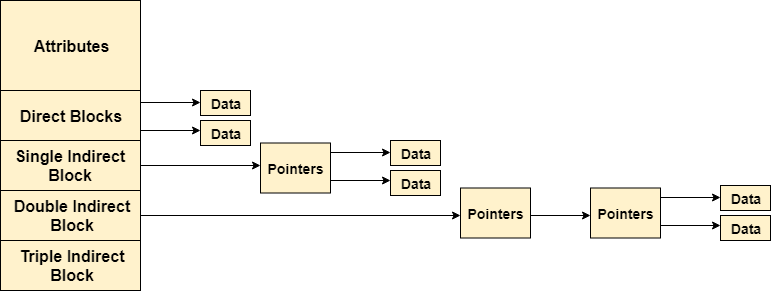
* File allocation methods are different ways by which the operating system stores information in memory blocks.
* There are 5 types of file allocation methods in OS. Contiguous File allocation, Linked File Allocation, Indexed File Allocation, File Allocation Table (FAT), and Inode.
* In Contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are contiguous(adjacent).
* In Linked file allocation the pointers are used to point to the next block of the same file.
* The Index file allocation is similar to Linked file allocation but here all the pointers are put together into one location which is called an index block.
* Using File allocation table (FAT) we can randomly access the disk block as it maintains the FAT for each file.
* Inode(information-node) contains some metadata about the file and stores all the pointers to the files where it can directly or indirectly access the file.

# Inode

In UNIX based operating systems, each file is indexed by an Inode. Inode are the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of Inodes in the file system.

An Inode includes the following information

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.



# Protection and Security in Operating System

Protection and security requires that computer resources such as CPU, softwares, memory etc. are protected. This extends to the operating system as well as the data in the system. This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protected against unauthorized access, viruses, worms etc.

## **Threats to Protection and Security**

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are −

### **Virus**

Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc. They can also spread further by replicating themselves as required.

### **Trojan Horse**

A trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system as a harmless being and wreak havoc.

### **Trap Door**

A trap door is a security breach that may be present in a system without the knowledge of the users. It can be exploited to harm the data or files in a system by malicious people.

### **Worm**

A worm can destroy a system by using its resources to extreme levels. It can generate multiple copies which claim all the resources and don't allow any other processes to access them. A worm can shut down a whole network in this way.

### **Denial of Service**

These types of attacks do not allow the legitimate users to access a system. It overwhelms the system with requests so it is overwhelmed and cannot work properly for other users.

## **Protection and Security Methods**

The different methods that may provide protect and security for different computer systems are −

### **Authentication**

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

* Username/ Password  
  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.
* User Key/ User Card  
  The users need to punch a card into the card slot or use their individual key on a keypad to access the system.
* User Attribute Identification  
  Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

### **One Time Password**

These passwords provide a lot of security for authentication purposes. A one time password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once. The various ways a one time password can be implemented are −

* Random Numbers  
  The system can ask for numbers that correspond to alphabets that are pre arranged. This combination can be changed each time a login is required.
* Secret Key  
  A hardware device can create a secret key related to the user id for login. This key can change each time.

# Difference between Security and Protection in Operating System

An operating system offers a technique to avoid tampering with ***logical*** and ***physical*** resources. Two of these are ***security*** and ***protection.*** Although these terms are frequently used interchangeably, protection and security are different. Protection entails preventing unauthorized users from interfering with the user's applications and data. In contrast, security entails protecting the user's programs and data against disruption by outside parties like unauthorized users of other systems.

In this article, you will learn about Security and Protection in the operating system. But before discussing the differences, you must know about the security and protection of the OS.

## **What is security in the operating system?**

A system's security revolves around its external environment and needs a suitable protective system. Security systems safeguard computer resources against unauthorized access, manipulation, and inconsistency. In this context, resources might be stored in the ***system, CPU, memory, drives,*** etc.

The system's security emphasizes the system's authentication process to secure the physical resources and the integrity of the information contained in the system. Security is a method that protects the user's programs and data against interference produced by an entity or person outside the system. For instance, multiple workers of an organization can access data. Still, it cannot be accessed by a user who does not exist in that organization or a user who works in another organization. An organization's primary responsibility is to implement security measures to prevent unauthorized access to its data by external users.

## **What is Protection in Operating Systems?**

Protection is a component of security that regulates system access by restricting the types of file access permitted to users. A system's security must ensure the permission of processes or users. As a result, authorized users or processes can access the CPU, memory regions, and other resources. The protection technique should include a way of identifying and enforcing the controls that will be enforced.

The protection was thought to be a useful feature for a multiprogramming OS. It prevents unreliable users from sharing a logical and physical namespace, including a directory of files and memory. Protection is required to intercept a user's malicious, purposeful breach of an access constraint. However, it is vital to ensure that each active program component in a system only uses system resources in the ways defined in policies. It protects the user's data and apps from interception by other system users.

## **Key differences between the Security and Protection in Operating System**

There are various key differences between security and protection in the OS. Some key differences between security and protection are as follows:

1. Security is a technique used in operating systems to address threats from outside the system to maintain the system's proper functioning. On the other hand, protection is a technique used in operating systems to control hazards and maintain the system's proper functioning.
2. The security technique specifies whether or not a specific user is allowed to access the system. In contrast, the protection technique outlines which users are permitted to access a certain resource.
3. Security techniques include adding, deleting users, determining whether or not a certain user is authorized, employing anti-malware software, etc. On the other hand, protection includes techniques like modifying a resource's protection information and determining whether a user may access it.
4. Security is a wide phrase that handles more complicated queries. On the other hand, the protection comes with security and covers less complex queries.
5. Security techniques mainly focus on external threats to the system. On the other hand, the protection techniques mainly focus on internal threats to the system.
6. Security offers a technique for protecting system and user resources from unauthorized access. On the other hand, the protection controls access to processes, programs, and user resources.

## **Head-to-head comparison between the Security and Protection in Operating System**

There are various head-to-head comparisons between the security and protection in the operating system. Some comparisons of security and protection are as follows:

|  |  |  |
| --- | --- | --- |
| **Features** | **Security** | **Protection** |
| **Definition** | It is a technique used in operating systems to address threats from outside the system to maintain its proper functioning. | It is a technique used in operating systems to control hazards and maintain the system's proper functioning. |
| **Focus** | It mainly focuses on external threats to the system. | It mainly focuses on the internal threats of the system. |
| **Policy** | It specifies whether or not a specific user is allowed to access the system. | It outlines which users are permitted to access a certain resource. |
| **Functionality** | It offers a technique for protecting system and user resources from unauthorized access. | It offers a technique for controlling access to processes, programs, and user resources. |
| **Mechanism** | Security techniques include adding, deleting users, determining whether or not a certain user is authorized, employing anti-malware software, etc. | It includes techniques like modifying a resource's protection information and determining whether a user may access it. |
| **Queries** | It is a wide phrase that handles more complicated queries. | It comes with security and covers less complex queries. |

## **Conclusion**

Security is a more complex system than protection since protection deals with internal risks and the environment, whereas security deals with external threats.

# Protection in File System

In computer systems, alot of user’s information is stored, the objective of the operating system is to keep safe the data of the user from the improper access to the system. Protection can be provided in a number of ways. For a single laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. For multi-user systems, different mechanisms are used for the protection.

**Types of Access :**

The files which have direct access of the any user have the need of protection. The files which are not accessible to other users doesn’t require any kind of protection. The mechanism of the protection provide the facility of the controlled access by just limiting the types of access to the file. Access can be given or not given to any user depends on several factors, one of which is the type of access required. Several different types of operations can be controlled:

* **Read –** Reading from a file.
* **Write –** Writing or rewriting the file.
* **Execute –** Loading the file and after loading the execution process starts.
* **Append –** Writing the new information to the already existing file, editing must be ended at the end of the existing file.
* **Delete –** Deleting the file which is of no use and using its space for the other data.
* **List –** List the name and attributes of the file.

Operations like renaming, editing the existing file, copying; these can also be controlled. There are many protection mechanisms. Each of these mechanisms have different advantages and disadvantages and must be appropriate for the intended application.

**Access Control :**

There are different methods used by different users to access any file. The general way of protection is to associate *identity-dependent access* with all the files and directories in a list called [access-control list (ACL)](https://www.geeksforgeeks.org/access-lists-acl/) which specify the names of the users and the types of access associate with each of the user. The main problem with the access list is their length. If we want to allow everyone to read a file, we must list all the users with the read access. This technique has two undesirable consequences:

Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of the users in the system.

Previously, the entry of the any directory is of the fixed size but now it changes to the variable size which results in the complicates space management. These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classification of users in connection with each file:

* **Owner –** Owner is the user who has created the file.
* **Group –** A group is a set of members who has similar needs and they are sharing the same file.
* **Universe –** In the system, all other users are under the category called universe.

The most common recent approach is to combine access-control lists with the normal general owner, group, and universe access control scheme. For example: Solaris uses the three categories of access by default but allows access-control lists to be added to specific files and directories when more fine-grained access control is desired.

**Other Protection Approaches:**

The access to any system is also controlled by the password. If the use of password is random and it is changed often, this may result in limit the effective access to a file.

The use of passwords has a few disadvantages:

* The number of passwords is very large so it is difficult to remember the large passwords.
* If one password is used for all the files, then once it is discovered, all files are accessible; protection is on all-or-none basis.