**UNIT-1**

An operating System (OS) is an intermediary between users and computer hardware. It provides an environment in which a user can execute programs conveniently and efficiently. In technical terms, it is system software which manages hardware. An operating System controls the allocation of resources and services such as memory, processors, devices and information.

#### Operating System objectives:

An amazing aspect of operating systems is how varied they are in accomplishing these tasks. Mainframe operating systems are designed primarily to optimize utilization of hardware. Personal computer (PC) operating systems support complex games, business applications, and everything in between. Operating systems for handheld computers are designed to provide an environment in which a user can easily interface with the computer to execute programs. Thus, some operating systems are designed to be convenient, others to be efficient, and others some combination of the two.

* Execute user programs and make solving user problems easier
* Make the computer system convenient to use
* Use the computer hardware in an efficient manner

#### User View:

The user's view of the computer varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor/ keyboard/ mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work (or play) that the user is performing. In this case/ the operating system is designed mostly for ease of use with some attention paid to performance and none paid to resource utilization various hardware and software resources are shared. Performance is, of course, important to the user; but such systems are optimized for the single-user experience rather than the requirements of multiple users. In other cases, a user sits at a terminal connected to a mainframe or a minicomputer. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization to assure that all available CPU time, memory, and I/0 are used efficiently and that no individual user takes more than her fair share.

In still other cases, users sit at workstations connected to networks of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as networking and servers-file, compute, and print servers. Therefore, their operating system is designed to compromise

between individual usability and resource utilization. Recently, many varieties of handheld computers have come into fashion. Most of these devices are standalone units for individual users. Some are connected to networks, either directly by wire or (more often) through wireless modems and networking. Because of power, speed, and interface limitations, they perform relatively few remote operations. Their operating systems are designed mostly for individual usability, but performance per unit of battery life is important as well. Some computers have little or no user view. For example, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their operating systems are designed primarily to run without user intervention.

#### System View:

From the computer's point of view, the operating system is the program most intimately involved with the hardware. In this context, we can view an operating system as a resource allocator. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/0 devices, and so on. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly. As we have seen, resource allocation is especially important where many users access the same mainframe or minicomputer. A slightly different view of an operating system emphasizes the need to control the various I/0 devices and user programs. An operating system is a control program. A control program manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

### Definition

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

### OS Functions:

Following are some of the important functions of an operating System.

* Memory Management
* Processor Management
* Device Management
* File Management
* Security
* Control over system performance
* Job accounting
* Error detecting and Prevention
* Coordination between other software and users

### Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.Main memory provides a fast storage that can be access directly by the CPU. So for a program to be executed, it must be in the main memory.

Operating System does the following activities for memory management.

* Keeps tracks of primary memory i.e. what part of it are in use by whom, what part are not in use.
* In multiprogramming, OS decides which process will get memory when and how much.
* Allocates the memory when the process requests it to do so.
* De-allocates the memory when the process no longer needs it or has been terminated.

### Processor Management

In a multiprogramming environment, OS decides which process gets the processor when and how much time. This function is called process scheduling. The Operating System does the following activities for processor management.

* Keeps tracks of processor and status of process. Program responsible for this task is known as the traffic controller.
* Allocates the Processor (CPU) to a process.
* De-allocates processor when processor is no longer required.

### Device Management

OS manages device communication via their respective drivers. Operating System does the following activities for device management.

* Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
* Decides which process gets the device when and for how much time.
* Allocates the device in the efficient way.
* De-allocates devices.

### File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Operating System does the following activities for file management.

* Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
* Decides who gets the resources.
* Allocates the resources.
* De-allocates the resources.

Following are some of the important activities that Operating System does:

* Security -- By means of password and similar other techniques, preventing unauthorized access to programs and data.
* Control over system performance -- Recording delays between request for a service and response from the system.
* Job accounting -- Keeping track of time and resources used by various jobs and users.
* Error detecting aids -- Production of dumps, traces, error messages and other debugging and error detecting aids.
* Coordination between other Softwares and users -- Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Computer System Architecture:

Computer system can be divided into four components

* Hardware – provides basic computing resources CPU, memory, I/O devices
* Operating system

Controls and coordinates use of hardware among various applications and users

* Application programs – define the ways in which the system resources are used to solve the computing problems of the users

Word processors, compilers, web browsers, database systems, video games

* Users

People, machines, other computers Operating-System Operations

* + Dual-mode operation allows OS to protect itself and other system components
  + User mode and kernel mode
  + Mode bit provided by hardware

1. Provides ability to distinguish when the system is running user code or kernel code.
2. Some instructions designated as privileged, only executable in kernel mode.
3. System call changes mode to kernel, return from call resets it to user.

#### Evaluation of Operating Systems

1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System Or parallel system
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Realtime Operating System
8. Handheld System

#### Simple Batch Systems

In this type of system, there is no direct interaction between user and the computer. The user has to submit a job to a computer operator.

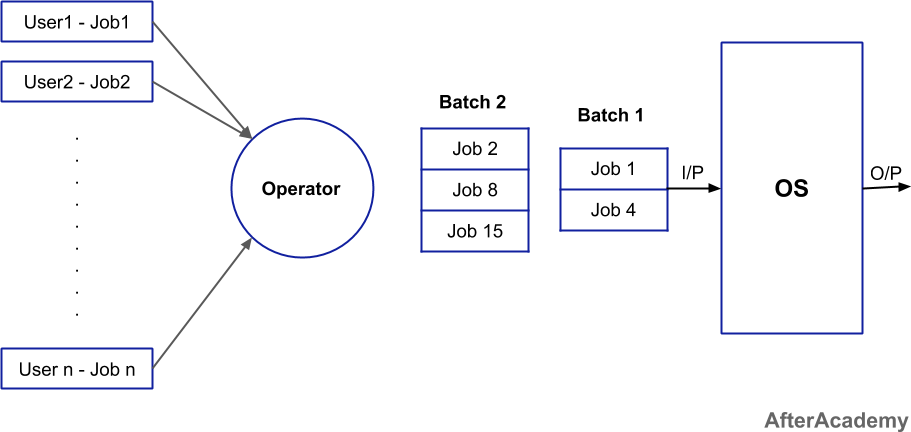
Then the computer operator places a batch of several jobs on an input device. Jobs are batched together by type of languages and requirements.

Then a special program, the monitor, manages the execution of each program in the batch. The monitor is always in the main memory and available for execution.

#### Advantages of Simple Batch Systems

* No interaction between user and computer.
* No mechanism to prioritise the processes.

#### Simple Batch Systems



**Multiprogramming Batch Systems**

In this the operating system picks up and begins to execute one of the jobs from memory.

Once this job needs an I/O operation, the operating system switches to another job (CPU and OS are always busy).

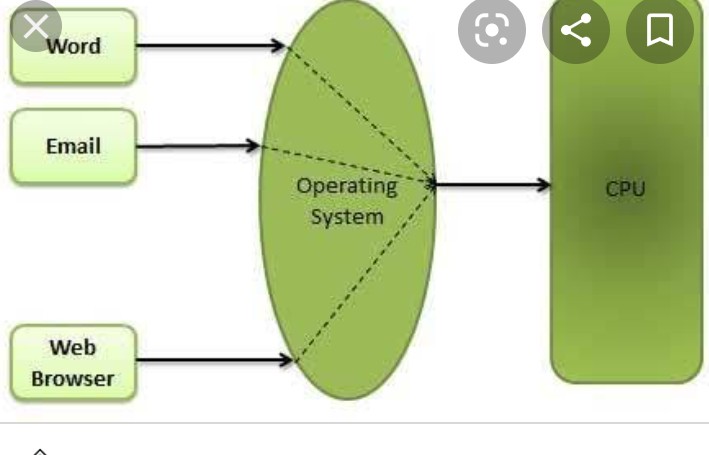
Jobs in the memory are always less than the number of jobs on disk(Job Pool).

If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU Scheduling.

In a Non-multiprogrammed system, there are moments when the CPU sits idle and does not do any work. In a Multiprogramming system, the CPU will never be idle and keeps on processing.

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.

Multiprogramming Batch Systems

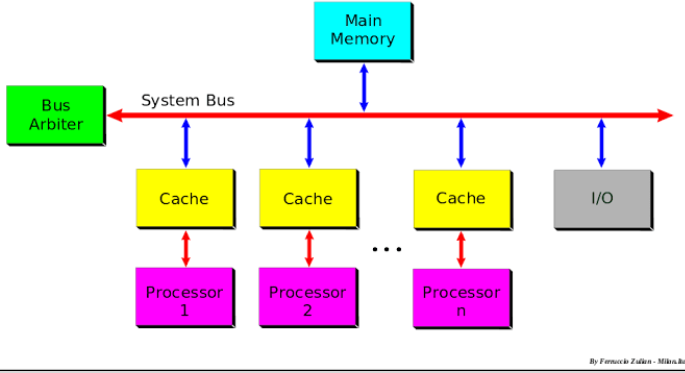
#### Multiprocessor Systems or parallel system

A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Advantages of Multiprocessor Systems

* Enhanced performance
* Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby

speeding up the execution of single tasks.



#### Desktop Systems or personal computer

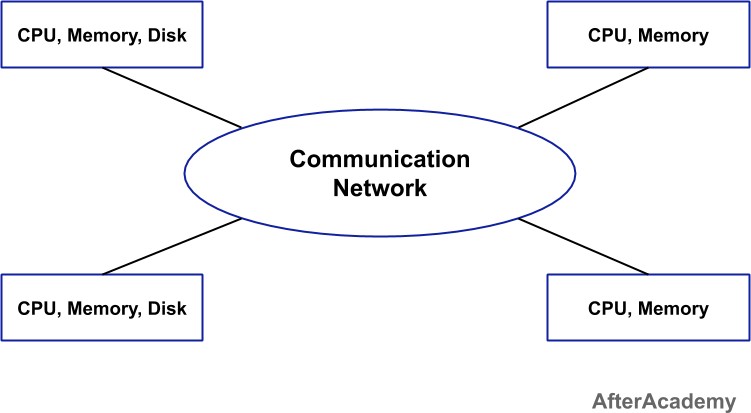
Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither multiuser nor multitasking. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called Desktop Systems and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for mainframes.

Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently low that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

Distributed Operating System

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems consisting of many computers that are interconnected by communication networks. The main benefit of distributed systems is its low price/performance ratio.



Advantages Distributed Operating System

* As there are multiple systems involved, users at one site can utilize the resources of systems at other sites for resource-intensive tasks.
* Fast processing.
* Less load on the Host Machine.

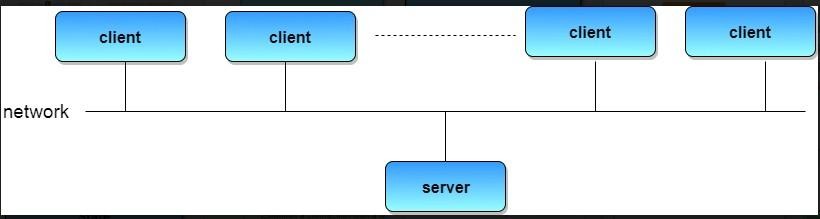
#### Types of Distributed Operating Systems

Following are the two types of distributed operating systems used:

Client-Server Systems Peer-to-Peer Systems

Client-Server Systems

Centralized systems today act as server systems to satisfy requests generated by client systems. The general structure of a client-server system is depicted in the figure below:

Client-Server Systems

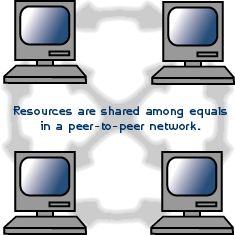
Server Systems can be broadly categorized as: Compute Servers and File Servers.

Compute Server systems provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.

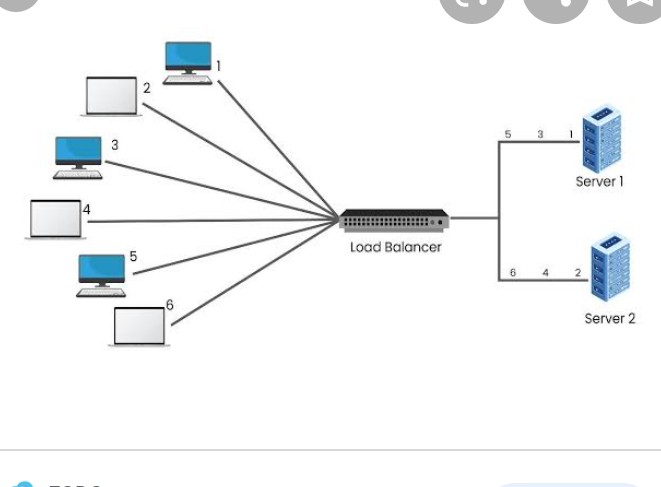
File Server systems, provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems

Peer-to-peer network operating systems allow users to share resources and files located on their computers and to access shared resources found on other computers. However, they do not have a file server or a centralized management source (See fig. 1). In a peer-to-peer network, all computers are considered equal; they all have the same abilities to use the resources available on the network



#### Clustered Systems

Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.

Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.

The definition of the term clustered is not concrete; the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.

Clustering is usually performed to provide high availability.

A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.

Asymmetric Clustering - In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.

Symmetric Clustering - In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.

Parallel Clustering - Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system's usage and its features should expand greatly as Storage Area Networks(SANs). SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

#### Real Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

The Real-Time Operating system which guarantees the maximum time for critical operations and completes them on time are referred to as Hard Real-Time Operating Systems.

While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.

Handheld Systems

Handheld systems include Personal Digital Assistants(PDAs), Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

Many handheld devices have between 512 KB and 8 MB of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.

Advantages

* Notes can be digitized
* Used for sending and receiving invoices
* Used for asset management
* For scanning barcodes
* Watch television through the medium of internet by IPTV
* Can sync information or data and share stuff
* Can provide educational materials to others
* It is best suited in terms to meet the mobility needs and capture data. It’s completely wireless and it is suited for all types of environments and applications.

There are various features of handheld computers such as:

* They are extremely easy to hold and portable.
* Good connectivity — has the ability to stay connected with various other devices. The connectivity with other network connections is also very strong.
* Individuality- in order to suit the needs of an individual it is very adaptable.
* Socially interactive- in the same environment it maintains a good amount of connectivity in order to collaborate with the different other users.

**Differences between Multiprocessing and Multiprogramming**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Multiprocessing Or  parallel operating system | Multiprogramming |
| 1 | Multiprocessing refers to processing of multiple processes at same time by multiple CPUs. | Multiprogramming keeps several programs in main memory at the same time and executes them concurrently utilizing a single CPU. |
| 2 | It utilizes multiple CPUs. | It utilizes a single CPU. |
| 3 | It permits parallel processing. | Context switching takes place. |
| 4 | Less time taken to process the jobs. | More Time taken to process the jobs. |
| 5 | It facilitates much efficient utilization of devices of the computer system. | Less efficient than multiprocessing. |
| 6 | Usually more expensive. | Such systems are less expensive. |

**Differences between Real Time System and Timesharing System**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Real Time System | Timesharing System |
| 1 | In this system, events mostly external to the computer system are accepted and processed within certain deadlines. | In this system, many users are  allowed to simultaneously share the computer resources. |
| 2 | Real time processing is mainly devoted to one application. | Time sharing processing deals with many different applications. |
| 3 | Users can make enquiry only and cannot write or modify programs. | Users can write and modify programs. |
| 4 | Users must get a response within the specified time limit; otherwise it may result in a disaster. | Users should get a response within fractions of seconds but if not, the results are not disastrous. |
| 5 | No context switching takes place in this system. | The CPU switches from one process to another as a time slice expires or a process terminates. |

**EXAMPLES OF OS**

## Windows is a series of [operating systems](https://techterms.com/definition/operating_system) developed by Microsoft. Each version of Windows includes a [graphical user interface](https://techterms.com/definition/gui), with a [desktop](https://techterms.com/definition/desktop) that allows users to view files and folders in [windows](https://techterms.com/definition/window). For the past two decades, Windows has been the most widely used operating system for personal computers [PCs](https://techterms.com/definition/pc).

Microsoft Windows is designed for both home computing and professional purposes. Past versions of Windows home editions include Windows 3.0 (1990), Windows 3.1 (1992), Windows 95 (1995), Windows 98 (1998), Windows Me (2000), Windows XP

## (2001), and Windows Vista (2006).Windows 7, was released in 2009.The current version 20H2, Windows 10, was released in 2020

Best Features of Windows Operating System

1. Speed. ...
2. Compatibility. ...
3. Lower Hardware Requirements. ...
4. Search and Organization. ...
5. Safety and Security. ...
6. Interface and Desktop. ...
7. Taskbar/Start menu

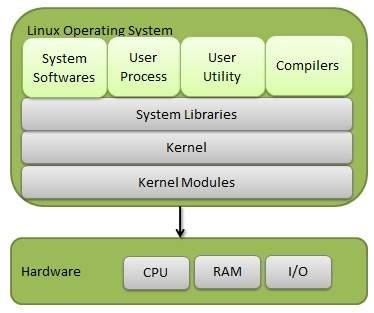
Linux is one of the popular versions of the UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

# Components of Linux System

Linux Operating System has primarily three components

* Kernel − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
* System Library − System libraries are special functions or programs using which application programs or system utilities access Kernel's features. These libraries

implement most of the functionalities of the operating system and do not require kernel module's code access rights.

* System Utility − System Utility programs are responsible to do specialized, individual level tasks.

**Basic Features**

Following are some of the important features of Linux Operating System.

* Portable − Portability means software can work on different types of hardware in the

same way. Linux kernel and application programs support their installation on any kind of hardware platform.

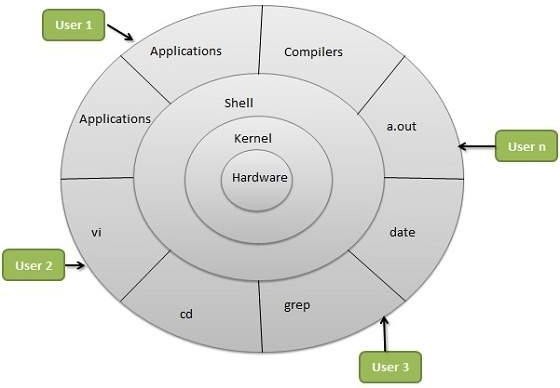
* Open Source − Linux source code is freely available and it is a community based development project. Multiple teams work in collaboration to enhance the capability of the Linux operating system and it is continuously evolving.
* Multi-User − Linux is a multiuser system meaning multiple users can access system resources like memory/ ram/ application programs at same time.
* Multiprogramming − Linux is a multiprogramming system meaning multiple applications can run at same time.
* Hierarchical File System − Linux provides a standard file structure in which system files/ user files are arranged.
* Shell − Linux provides a special interpreter program which can be used to execute

commands of the operating system. It can be used to do various types of operations, call application programs. etc.

* Security − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

**Architecture**

The following illustration shows the architecture of a Linux system −



**The architecture of a Linux System consists of the following layers −**

* Hardware layer − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
* Kernel − It is the core component of the Operating System, interacts directly with hardware, provides low level services to upper layer components.
* Shell − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes the kernel's functions.
* Utilities − Utility programs that provide the user most of the functionalities of an operating system.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Key** | **Linux** | **Windows** |
| 1 | Open Source | Linux is Open Source and is free to use. | Windows is not open source and is not free to use. |
| 2 | Case sensitivity | The Linux file system is case sensitive. | Windows file system is case insensitive. |
| 3 | kernel type | Linux uses a monolithic kernel. | Windows uses a micro kernel. |
| 4 | Efficiency | Linux is more efficient in operations as compared to Windows. | Windows is less efficient in operations. |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | Path  Separator | Linux uses forward slash as path separator between directories. | Windows uses backward slash as a path separator. |
| 6 | Security | Linux is highly secure as compared to Windows. | Windows provides less security as compared to Linux. |

### OPERATING SYSTEM SERVICES

* 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 −

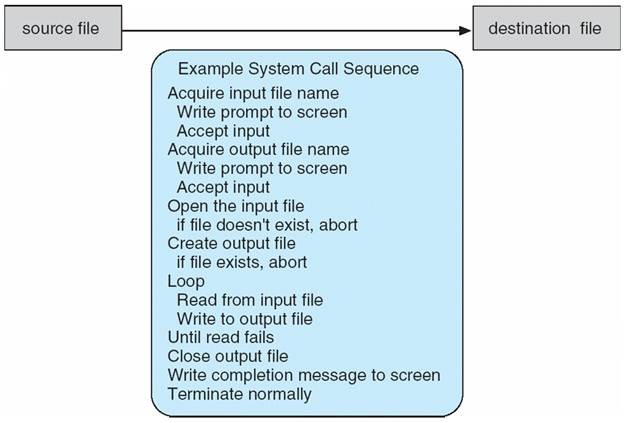
* The OS ensures that all access to system resources is controlled.
* The OS ensures that external I/O devices are protected from invalid access attempts.
* The OS provides authentication features for each user by means of passwords.

### System Calls

* Programming interface to the services provided by the OS
* Typically written in a high-level language (C or C++)
* Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
* Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based

systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

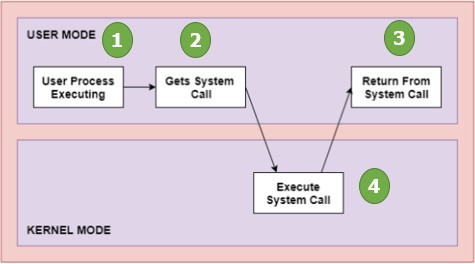
### Example of System Calls



**System Call Implementation**

* Typically, a number associated with each system call
* System-call interface maintains a table indexed according to these Numbers
* The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
* The caller need know nothing about how the system call is implemented
* Just needs to obey API and understand what OS will do as a result call
* Most details of OS interface hidden from programmer by API

Managed by run-time support library (set of functions built into libraries included with compiler)



### Types of System Calls

* Process control
* File management
* Device management
* Information maintenance
* Communications
* Protection

### Examples of Windows and Unix System Calls

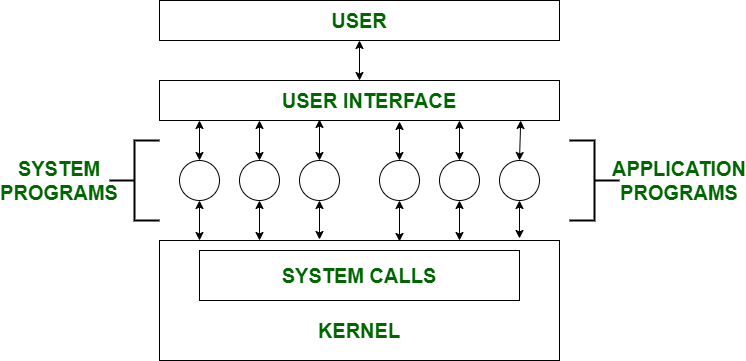
|  |  |  |
| --- | --- | --- |
| **Types of**  **System Calls** | **Windows** | **Linux** |
| Process Control | CreateProcess() ExitProcess()  WaitForSingleObject() | fork()  exit()  wait() |
| File Management | CreateFile() ReadFile() WriteFile()  CloseHandle() | open()  read() write() close() |

|  |  |  |
| --- | --- | --- |
| Device Management | SetConsoleMode() ReadConsole() WriteConsole() | ioctl() read() write() |
| Information Maintenance | GetCurrentProcessID() SetTimer()  Sleep() | getpid() alarm() sleep() |
| Communication | CreatePipe()  CreateFileMapping() MapViewOfFile() | pipe() shmget() mmap( ) |

**System Programs**

System programs provide a convenient environment for program development and execution. These can be divided into:

* File manipulation
* Status information
* File modification
* Programming language support
* Program loading and execution
* Communications
* Application programs



### File Management –

A file is a collection of specific information stored in the memory of a computer system. File management is defined as the process of manipulating files in the

computer system, its management includes the process of creating, modifying and deleting files.

* + - It helps to create new files in the computer system and place them at specific locations.
    - It helps in easily and quickly locating these files in the computer system.
    - It makes the process of sharing files among different users very easy and user-friendly.
    - It helps to store files in separate folders known as directories.
    - These directories help users to search files quickly or to manage files according to their types of uses.
    - It helps users to modify the data of files or to modify the name of files in directories.

### Status Information –

Information like date, time amount of available memory, or disk space is asked by some users. Others provide detailed performance, logging, and debugging

information which is more complex. All this information is formatted and displayed

on output devices or printed. Terminal or other output devices or files or a window of GUI is used for showing the output of programs.

### File Modification –

For modifying the contents of files we use this. For Files stored on disks or other

storage devices, we used different types of editors. For searching contents of files or performing transformations of files we use special commands.

### Programming-Language support –

For common programming languages, we use Compilers, Assemblers, Debuggers, and interpreters which are already provided to users. It provides all support to users. We can run any programming language. All languages of importance are already provided.

### Program Loading and Execution –

When the program is ready after Assembling and compilation, it must be loaded into memory for execution. A loader is part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages for starting a program. Loaders, relocatable loaders, linkage editors, and Overlay loaders are provided by the system.

### Communications –

Virtual connections among processes, users, and computer systems are provided by programs. Users can send messages to another user on their screen, users can send

email, browse on web pages, remote login, the transformation of files from one user to another.

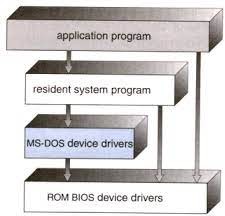
### Some examples of system programs in O.S. are –

* + Anti-virus
  + Disk formatting
  + Computer language translators

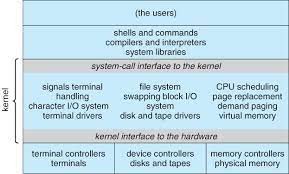
### Operating System Structure:

1. **Simple Structure**
   * MS-DOS – written to provide the most functionality in the least space
   * Not divided into modules
   * Although MS-DOS has some structure, its interfaces and levels of Functionality are not well separated

### MS-DOS Layer Structure



**Traditional UNIX System Structure**



### UNIX

* + UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  + Systems programs
  + The kernel

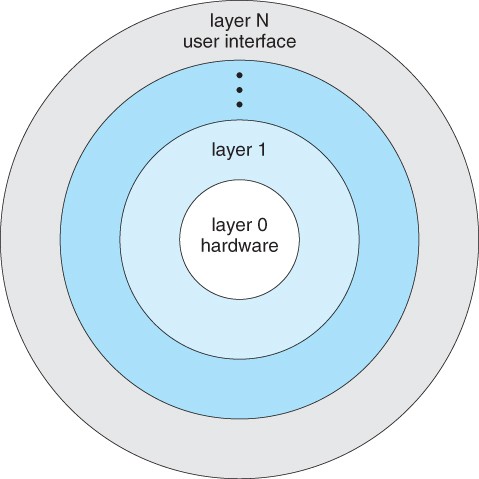
Consists of everything below the system-call interface and above the physical hardware

Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

### Layered Approach

* + The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
  + With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

### Layered Operating System

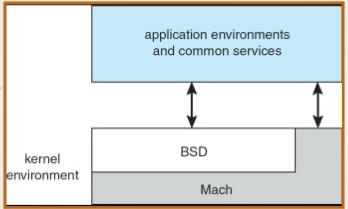


1. **Microkernel System Structure**
   * Moves as much from the kernel into “*user*” space
   * Communication takes place between user modules using message passing

### Benefits:

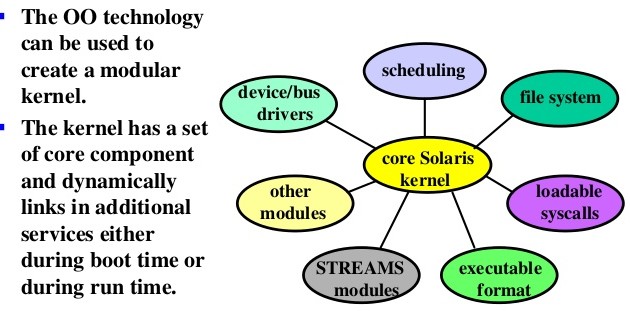
* + Easier to extend a microkernel
  + Easier to port the operating system to new architectures
  + More reliable (less code is running in kernel mode)
  + More secure
  + Performance overhead of user space to kernel space communication

### Mac OS X Structure



1. **Module Structure**
   * Most modern operating systems implement kernel modules
   * Uses object-oriented approach
   * Each core component is separate
   * Each talks to the others over known interfaces
   * Each is loadable as needed within the kernel
   * Overall, similar to layers but with more flexible

### Solaris Modular Approach

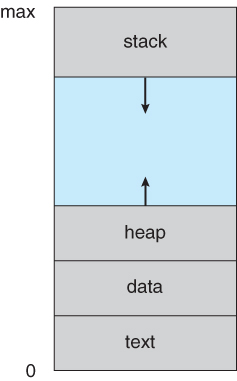


### UNIT-2

***Process:*** A process is a program in execution. The execution of a process must progress in a sequential fashion. Definition of process is following. A process is defined as an entity which represents the basic unit of work to be implemented in the system. Components of process are following. A program is passive entity whereas a process is an active entity. A program can be transformed to process in 2 ways: by double clicking the file and typing the file from command prompt.

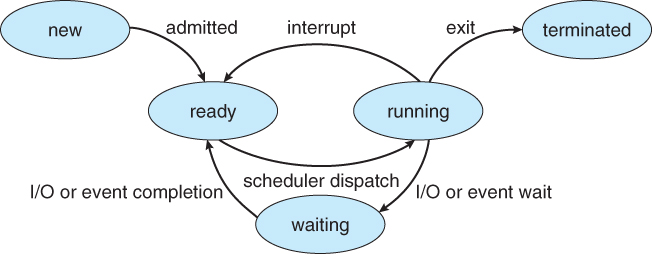
Process memory is divided into four sections as shown:

* The text section comprises the compiled program code, read in from non-volatile storage when the program is launched.
* The data section stores global and static variables, allocated and initialized prior to executing main.
* The heap is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
* The stack is used for local variables. Space on the stack is reserved for local variables when they are declared ( at function entrance or elsewhere, depending on the language ), and the space is freed up when the variables go out of scope. Note that the stack is also used for function return values, and the exact mechanisms of stack management may be language specific.
* Note that the stack and the heap start at opposite ends of the process's free space and grow towards each other. If they should ever meet, then either a stack overflow error will occur, or else a call to new or malloc will fail due to insufficient memory available.
* When processes are swapped out of memory and later restored, additional information must also be stored and restored. Key among them are the program counter and the value of all program registers.



**Process States:** Processes may be in one of 5 states as shown:

* **New** - The process is in the stage of being created.
* **Ready**- The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
* **Running**- The CPU is working on this process's instructions.
* **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
* **Terminated -**The process has completed.

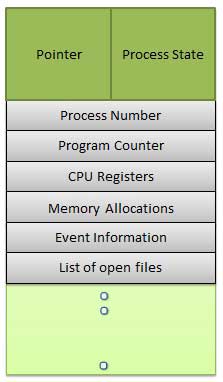


***Process Control Block (PCB)***

Each process is represented in the operating system by a process control block (PCB) also called a task control block. PCB is the data structure ***Process Control Block (PCB)***

Each process is represented in the operating system by a process control block (PCB) also called a task control block. PCB is the data structure used by the operating system. Operating system groups all information that needs about particular process. PCB contains many pieces of information associated with a specific process which is described below.

|  |  |
| --- | --- |
| **S.N.** | **Information & Description** |
| 1 | **Pointer** Pointer points to another process control block. Pointer is used for maintaining the scheduling list. |
| 2 | **Process State** Process state may be new, ready, running, waiting and so on. |
| 3 | **Program Counter** Program Counter indicates the address of the next instruction to be executed for this process. |
| 4 | **CPU registers** CPU registers include general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture. |
| 5 | **Memory management information** This information may include the value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the operating system. This information is useful for de-allocating the memory when the process terminates. |
| 6 | **Accounting information** This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc. |
|  |  |



Process Control Block (**PCB**) includes CPU scheduling, I/O resource management, file management information etc.. The PCB serves as the repository for any information which can vary from process to process. Loader/linker sets flags and registers when a process is created. If that process gets suspended, the contents of the registers are saved on a stack and the pointer to the particular stack frame is stored in the PCB. By this technique, the hardware state can be restored so that the process can be scheduled to run again.

***Process Scheduling***

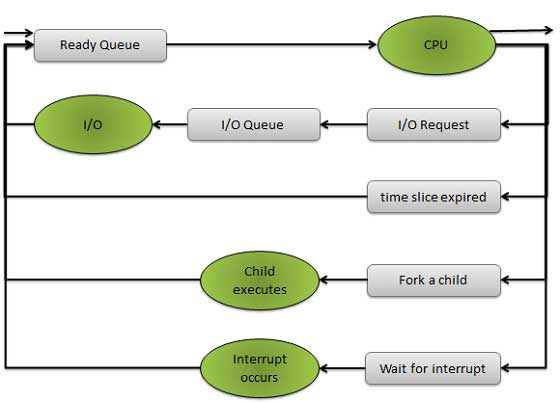
The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. It is an essential part of a Multiprogramming operating system. Such operating systems allow more than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

***Scheduling Queues***

* All processes are stored in the **job queue**.
* Processes in the Ready state are placed in the **ready queue**.
* Processes waiting for a device to become available or to deliver data are placed in **device queues**. There is generally a separate device queue for each device.
* Other queues may also be created and used as needed.

**Queuing diagram:**

* Queue is represented by rectangular box.
* The circles represent the resources that serve the queues.
* The arrows indicate the process flow in the system.



***Schedulers***

Schedulers are special system software’s which handles 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. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution. 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 process changes the state from new to ready, then there is use of long term scheduler.

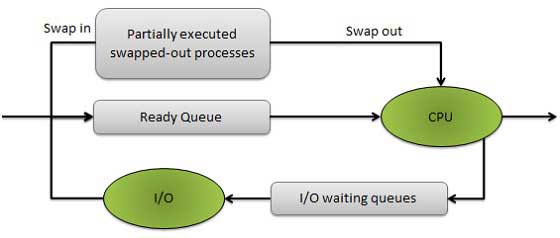
***Short Term Scheduler***

It is also called CPU scheduler. Main objective is increasing 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 process among the processes that are ready to execute and allocates CPU to one of them.

Short term scheduler also known as dispatcher, executed most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than Long term scheduler.

***Medium Term Scheduler***

Medium term scheduling is part of the 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.



Running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other process, 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.

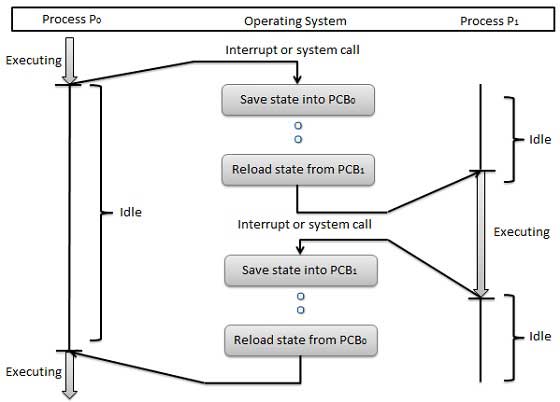
**Comparison between Schedulers**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 Switch***

A context switch 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 context switcher saves the content of all processor registers for the process being removed from the CPU, in its process descriptor. The context of a process is represented in the process control block of a process.

Context switch time is pure overhead. Context switching can significantly affect performance as modern computers have a lot of general and status registers to be saved. Content switching times are highly dependent on hardware support. Context switch requires ( n + m ) bxK time units to save the state of the processor with ‘n’ general registers, assuming ‘b’ are the store operations are required to save ‘n’ and ‘m’ registers of two process control blocks and each store instruction requires ‘K’ time units.



Some hardware systems employ two or more sets of processor registers to reduce the amount of context switching time. When the process is switched, the following information is stored.

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

**CPU Scheduler**

Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state

2. Switches from running to ready state

3. Switches from waiting to ready

4. Terminates

Scheduling under 1 and 4 is **non-preemptive and** all other scheduling is **preemptive**

**Dispatcher**

* Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
* switching context
* switching to user mode
* jumping to the proper location in the user program to restart that program
* **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

**Scheduling Criteria**

* **CPU utilization** – keep the CPU as busy as possible
* **Throughput** – # of processes that complete their execution per time unit
* **Turnaround time** – amount of time to execute a particular process
* **Waiting time** – amount of time a process has been waiting in the ready queue
* **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
* Max CPU utilization
* Max throughput
* Min turnaround time
* Min waiting time
* Min response time

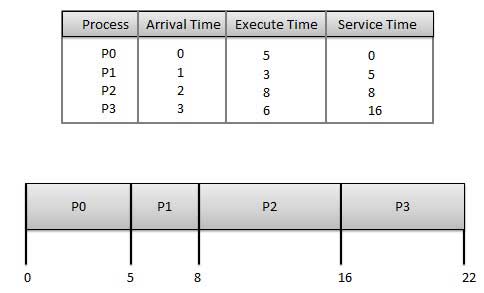
**Scheduling algorithms**

Four major scheduling algorithms here which are following :

* First Come First Serve (FCFS) Scheduling
* Shortest-Job-First (SJF) Scheduling
* Priority Scheduling
* Round Robin(RR) Scheduling

***First Come First Serve (FCFS)***

* Jobs are executed on first come, first serve basis.
* Easy to understand and implement.
* Poor in performance as average wait time is high.



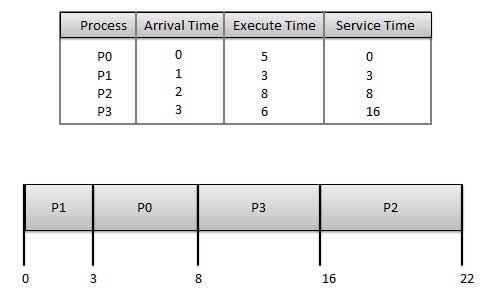
Wait time of each process is following

|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | 0 - 0 = 0 |
| P1 | 5 - 1 = 4 |
| P2 | 8 - 2 = 6 |
| P3 | 16 - 3 = 13 |

Average Wait Time: (0+4+6+13) / 4 = 5.55

***Shortest Job First (SJF)***

* Best approach to minimize waiting time.
* Impossible to implement
* Processer should know in advance how much time process will take.



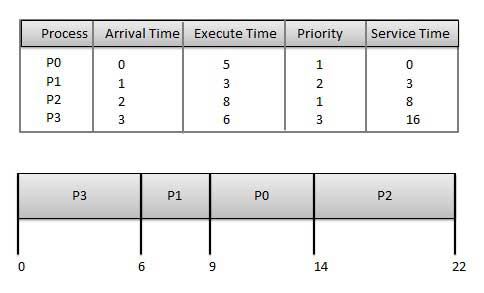
Wait time of each process is following

|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | 3 - 0 = 3 |
| P1 | 0 - 0 = 0 |
| P2 | 16 - 2 = 14 |
| P3 | 8 - 3 = 5 |

Average Wait Time: (3+0+14+5) / 4 = 5.50

***Priority Based Scheduling***

* Each process is assigned a priority. Process with highest priority is to be executed first and so on.
* Processes with same priority are executed on first come first serve basis.
* Priority can be decided based on memory requirements, time requirements or any other resource requirement.



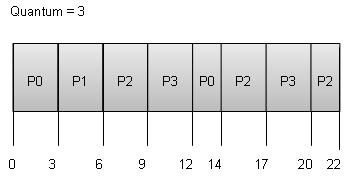
Wait time of each process is following

|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | 9 - 0 = 9 |
| P1 | 6 - 1 = 5 |
| P2 | 14 - 2 = 12 |
| P3 | 0 - 0 = 0 |

Average Wait Time: (9+5+12+0) / 4 = 6.5

***Round Robin Scheduling***

* Each process is provided a fix time to execute called quantum.
* Once a process is executed for given time period. Process is preempted and other process executes for given time period.
* Context switching is used to save states of preempted processes.



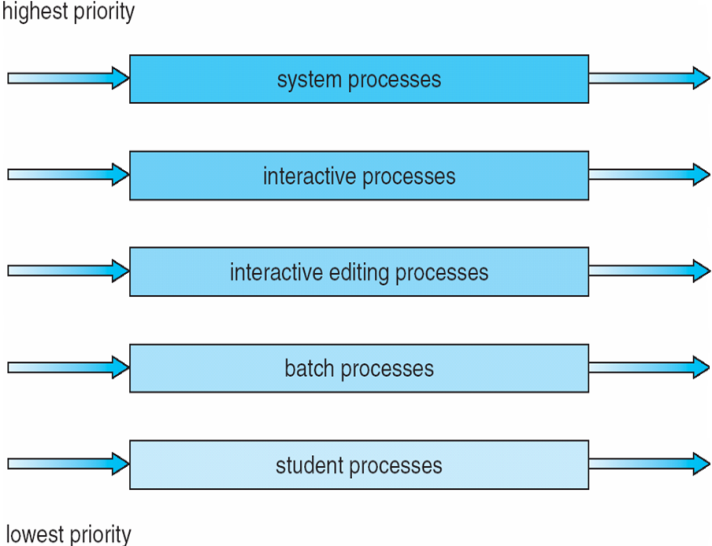
Wait time of each process is following

|  |  |
| --- | --- |
| **Process** | **Wait Time : Service Time - Arrival Time** |
| P0 | (0-0) + (12-3) = 9 |
| P1 | (3-1) = 2 |
| P2 | (6-2) + (14-9) + (20-17) = 12 |
| P3 | (9-3) + (17-12) = 11 |

Average Wait Time: (9+2+12+11) / 4 = 8.5

**1. Multilevel Queue**

* Ready queue is partitioned into separate queues:  
  foreground (interactive) background (batch)
* Each queue has its own scheduling algorithm
* foreground – RR
* background – FCFS
* Scheduling must be done between the queues
* Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
* Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR

****20% to background in FCFS

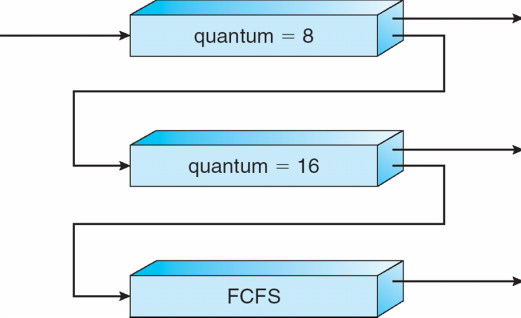
**2.Multilevel Feedback Queue**

* A process can move between the various queues; aging can be implemented this way
* Multilevel-feedback-queue scheduler defined by the following parameters:
* number of queues
* scheduling algorithms for each queue
* method used to determine when to upgrade a process
* method used to determine when to demote a process
* method used to determine which queue a process will enter when that process needs service

**Example of Multilevel Feedback Queue**

Three queues:

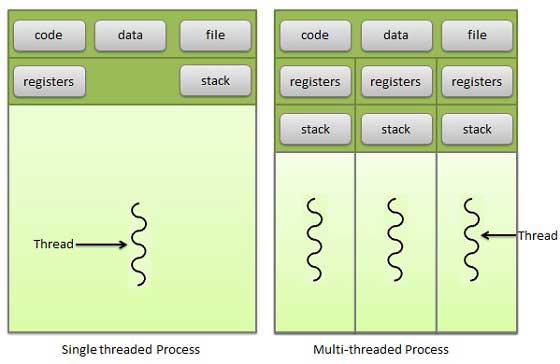
* *Q*0 – RR with time quantum 8 milliseconds
* *Q*1 – RR time quantum 16 milliseconds
* *Q*2 – FCFS
* Scheduling
* A new job enters queue *Q0*which is servedFCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue *Q*1.
* At *Q*1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue *Q*2.

**Multilevel Feedback Queues**

***Thread Concepts:***

A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents separate flows of control.Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



***Difference between Process and Thread***

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Process** | **Thread** |
| 1 | Process is heavy weight or resource intensive. | Thread is light weight taking lesser resources than a process. |
| 2 | Process switching needs interaction with operating system. | Thread switching does not need to interact with operating system. |
| 3 | In multiple processing environments each process executes the same code but has its own memory and file resources. | All threads can share same set of open files, child processes. |
| 4 | If one process is blocked then no other process can execute until the first process is unblocked. | While one thread is blocked and waiting, second thread in the same task can run. |
| 5 | Multiple processes without using threads use more resources. | Multiple threaded processes use fewer resources. |
| 6 | In multiple processes each process operates independently of the others. | One thread can read, write or change another thread's data. |

***Advantages of Threads***

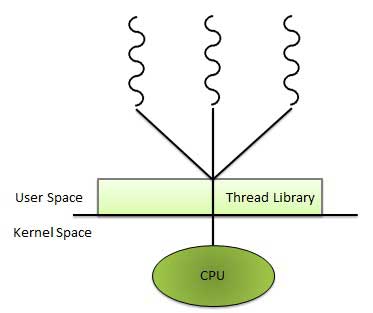
* Threads minimize context switching time.
* Use of threads provides concurrency within a process.
* Efficient communication.
* Economy- It is more economical to create and context switch threads.
* Utilization of multiprocessor architectures to a greater scale and efficiency.

***Types of Threads***

Threads are implemented in following two ways:

* **User Level Threads** -- User managed threads
* **Kernel Level Threads** -- Operating System managed threads acting on kernel, an operating system core.

***User Level Threads:*** In this case, application manages thread management, kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins running in that thread.



***Advantages***

* Thread switching does not require Kernel mode privileges.
* User level thread can run on any operating system.
* Scheduling can be application specific in the user level thread.
* User level threads are fast to create and manage.

***Disadvantages***

* In a typical operating system, most system calls are blocking.
* Multithreaded application cannot take advantage of multiprocessing.

***Kernel Level Threads***

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

***Advantages***

* Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
* If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
* Kernel routines themselves can multithreaded.

***Disadvantages***

* Kernel threads are generally slower to create and manage than the user threads.
* Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

***Multithreading Models***

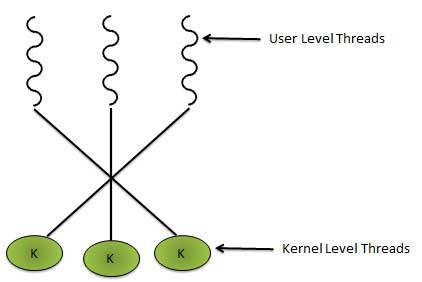
Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types :

* Many to many relationship.
* Many to one relationship.
* One to one relationship.

***Many to Many Models***

In this model, many user level threads multiplex to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

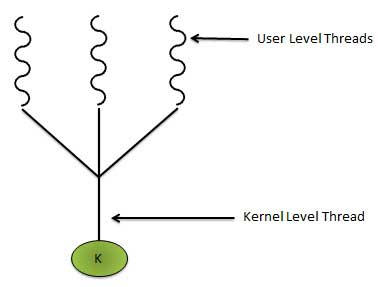
Following diagram shows the many to many models. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



***Many to One Model***

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

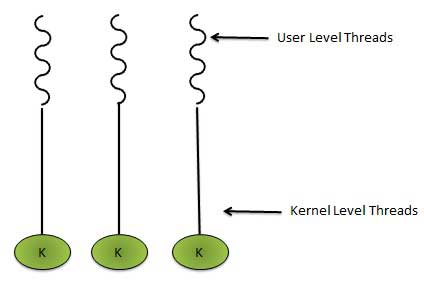
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



***One to One Model***

There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2 , Windows NT and Windows 2000 use one to one relationship model.



***Difference between User Level & Kernel Level Thread***

|  |  |  |
| --- | --- | --- |
| **S.N.** | **User Level Threads** | **Kernel Level Thread** |
| 1 | User level threads are faster to create and manage. | Kernel level threads are slower to create and manage. |
| 2 | Implementation is by a thread library at the user level. | Operating system supports creation of Kernel threads. |
| 3 | User level thread is generic and can run on any operating system. | Kernel level thread is specific to the operating system. |
| 4 | Multi-threaded application cannot take advantage of multiprocessing. | Kernel routines themselves can be multithreaded. |

# Process Synchronization

* To introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data
* To present both software and hardware solutions of the critical-section problem
* To introduce the concept of an atomic transaction and describe mechanisms to ensure atomicity
* Concurrent access to shared data may result in data inconsistency
* Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
* Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer

**Producer**

while (true) {

/\* produce an item and put in nextProduced \*/

while (count == BUFFER\_SIZE)

; // do nothing

buffer [in] = nextProduced;

in = (in + 1) % BUFFER\_SIZE;

count++;

}

**Consumer**

while (true) {

while (count == 0)

; // do nothing

nextConsumed = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

count--;

/\* consume the item in nextConsumed

}

**Race Condition**

count++ could be implemented as  
 register1 = count  
 register1 = register1 + 1  
 count = register1

count-- could be implemented as  
 register2 = count  
 register2 = register2 - 1  
 count = register2

Consider this execution interleaving with “count = 5” initially:

S0: producer execute register1 = count {register1 = 5}  
S1: producer execute register1 = register1 + 1 {register1 = 6}   
S2: consumer execute register2 = count {register2 = 5}   
S3: consumer execute register2 = register2 - 1 {register2 = 4}   
S4: producer execute count = register1 {count = 6 }   
S5: consumer execute count = register2 {count = 4}

**Solution to Critical-Section Problem**

1. Mutual Exclusion - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections

2. Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

3. 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

* Assume that each process executes at a nonzero speed
* No assumption concerning relative speed of the N processes

**Peterson’s Solution**

* Two process solution
* Assume that the LOAD and STORE instructions are atomic; that is, can’t be interrupted.
* The two processes share two variables:
* int turn;
* Boolean flag[2]
* The variable turn indicates whose turn it is to enter the critical section.
* The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = true implies that process Pi is ready!

**Algorithm for Process Pi**

do {

flag[i] = TRUE;

turn = j;

while (flag[j] && turn == j);

critical section

flag[i] = FALSE;

remainder section

} while (TRUE);

**Synchronization Hardware**

* Many systems provide hardware support for critical section code
* Uniprocessors – could disable interrupts
* Currently running code would execute without preemption
* Generally too inefficient on multiprocessor systems

Operating systems using this not broadly scalable

* Modern machines provide special atomic hardware instructions

Atomic = non-interruptable

* Either test memory word and set value Or swap contents of two memory words

**Solution to Critical-section Problem Using Locks**

do {

acquire lock

critical section

release lock

remainder section

} while (TRUE);

**TestAndSet Instruction**

Definition:

booleanTestAndSet (boolean \*target)

{

booleanrv = \*target;

\*target = TRUE;

returnrv:

}

**Solution using TestAndSet**

Shared boolean variable lock., initialized to false.

Solution:

do {

while ( TestAndSet (&lock ))

; // do nothing

// critical section

lock = FALSE;

// remainder section

} while (TRUE);

**Swap Instruction**

Definition:

void Swap (boolean \*a, boolean \*b)

{

boolean temp = \*a;

\*a = \*b;

\*b = temp:

}

**Solution using Swap**

Shared Boolean variable lock initialized to FALSE; Each process has a local Boolean variable key

Solution:

do {

key = TRUE;

while ( key == TRUE)

Swap (&lock, &key );

// critical section

lock = FALSE;

// remainder section

} while (TRUE);

**Bounded-waiting Mutual Exclusion with TestandSet()**

do {

waiting[i] = TRUE;

key = TRUE;

while (waiting[i] && key)

key = TestAndSet(&lock);

waiting[i] = FALSE;

// critical section

j = (i + 1) % n;

while ((j != i) && !waiting[j])

j = (j + 1) % n;

if (j == i)

lock = FALSE;

else

waiting[j] = FALSE;

// remainder section

} while (TRUE);

**Semaphore**

* Synchronization tool that doesnt require busy waiting
* . Semaphore *S* is integer variable
* Two standard operations modify S: wait() and signal()
* Originally called P() andV()
* Less complicated
* Can only be accessed via two indivisible (atomic) operations

wait (S) {

while S <= 0

; // no-op

S--;

}

signal (S) {

S++;

}

**Semaphore as General Synchronization Tool**

* Counting semaphore – integer value can range over an unrestricted domain
* Binary semaphore – integer value can range between 0 & 1; simpler to implement
* Also known as mutex locks
* . Can implement a counting semaphore S as a binary semaphore
* Provides mutual exclusion

Semaphoremutex; // initialized to

do {

wait (mutex);

// Critical Section

signal (mutex);

// remainder section

} while (TRUE);

**Semaphore Implementation**

* Must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time
* Thus, implementation becomes the critical section problem where the wait and signal code are placed in the crtical section.
* Could now have busy waiting in critical section implementation

But implementation code is short

Little busy waiting if critical section rarely occupied

* Note that applications may spend lots of time in critical sections and therefore this is not a good solution.

**Semaphore Implementation with no Busy waiting**

* With each semaphore there is an associated waiting queue. Each entry in a waiting queue has two data items:
* value (of type integer)
* pointer to next record in the list
* Two operations:
* block – place the process invoking the operation on the appropriate waiting queue.
* wakeup – remove one of processes in the waiting queue and place it in the ready queue.

Implementation of wait:

wait(semaphore \*S) {

S->value--;

if (S->value < 0) {

add this process to S->list;

block();

}

}

Implementation of signal:

signal(semaphore \*S) {

S->value++;

if (S->value <= 0) {

remove a process P from S->list;

wakeup(P);

}

}

**Deadlock and Starvation**

* Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
* Let S and Q be two semaphores initialized to 1

*P*0 *P*1

wait (S); wait (Q);

wait (Q); wait (S);

. .

. .

signal (S); signal (Q);

signal (Q); signal (S);

* Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended
* Priority Inversion - Scheduling problem when lower-priority process holds a lock needed by higher-priority process

**Classical Problems of Synchronization**

* Bounded-Buffer Problem
* Readers and Writers Problem
* Dining-Philosophers Problem

**Bounded-Buffer Problem**

* *N* buffers, each can hold one item
* Semaphore mutex initialized to the value 1
* Semaphore full initialized to the value 0
* Semaphore empty initialized to the value N.

The structure of the producer process

do {

// produce an item in nextp

wait (empty);

wait (mutex);

// add the item to the buffer

signal (mutex);

signal (full);

} while (TRUE);

The structure of the consumer process

do {

wait (full);

wait (mutex);

// remove an item from buffer to nextc

signal (mutex);

signal (empty);

// consume the item in nextc

} while (TRUE);

**Readers-Writers Problem** A data set is shared among a number of concurrent processes

* Readers – only read the data set; they do **not** perform any updates
* Writers – can both read and write
* nProblem – allow multiple readers to read at the same time. Only one single writer can access the shared data at the same time
* Shared Data
* Data set
* Semaphore mutex initialized to 1
* Semaphore wrt initialized to 1
* Integer readcount initialized to 0

The structure of a writer process

do {

wait (wrt) ;

// writing is performed

signal (wrt) ;

} while (TRUE);

The structure of a reader process

do {

wait (mutex) ;

readcount ++ ;

if (readcount == 1)

wait (wrt) ;

signal (mutex)

// reading is performed

wait (mutex) ;

readcount - - ;

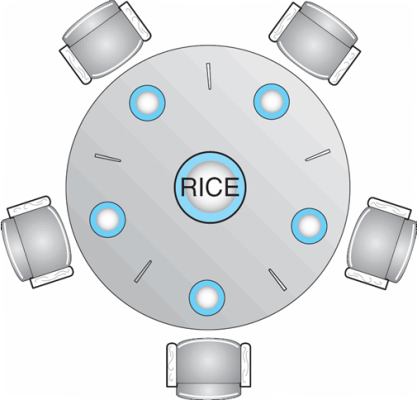
if (readcount == 0)

signal (wrt) ;

signal (mutex) ;

} while (TRUE);

**Dining-Philosophers Problem**

Shared data

* Bowl of rice (data set)
* Semaphore chopstick [5] initialized to 1
* The structure of Philosopher *i*:

do {

wait ( chopstick[i] ); wait ( chopStick[ (i + 1) % 5] );

// eat

signal ( chopstick[i] ); signal (chopstick[ (i + 1) % 5] );

// think

} while (TRUE);

**Problems with Semaphores**

Incorrect use of semaphore operations:

signal (mutex)

….

wait (mutex)

wait (mutex) …

Omitting of wait (mutex) or signal (mutex) (or both)

**Monitors**

A high-level abstraction that provides a convenient & effective mechanism for process synch.

Only one process may be active within the monitor at a time

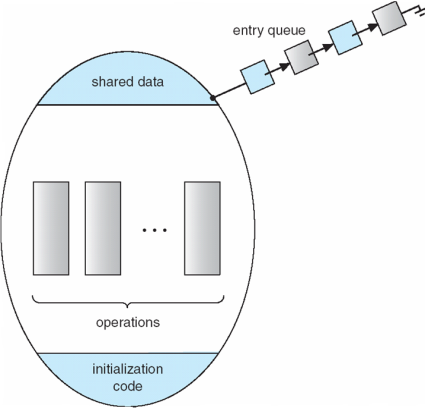
monitor monitor-name {

// shared variable declarations

procedure P1 (…) { …. }

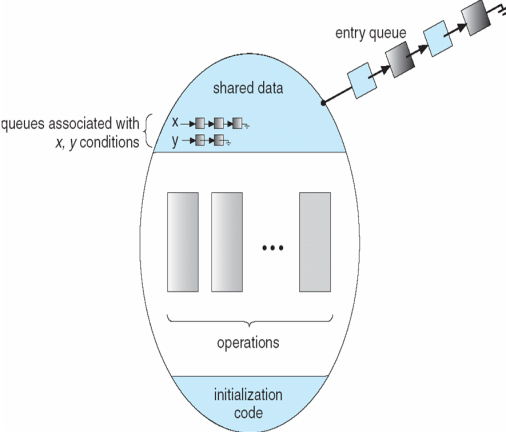
…

procedurePn (…) {……}



Initialization code ( ….) { … }

…

 } }

**Schematic view of a Monitor**

**Condition Variables**

condition x, y;

Two operations on a condition variable:

x.wait () – a process that invokes the operation is suspended.

x.signal()–resumes 1 of processes(if any) that invoked x.wait() **monitor with condition var’s**

**Solution to Dining Philosophers**

monitor DP{

enum { THINKING; HUNGRY, EATING) state [5] ;

condition self [5];

void pickup (inti) {

state[i] = HUNGRY;

test(i);

if (state[i] != EATING) self [i].wait;

}

void putdown (inti) {

state[i] = THINKING;

// test left and right neighbors

test((i + 4) % 5);

test((i + 1) % 5);

}

void test (inti) {

if ( (state[(i + 4) % 5] != EATING) &&

(state[i] == HUNGRY) &&

(state[(i + 1) % 5] != EATING) ) {

state[i] = EATING ;

self[i].signal () ;

} }

initialization\_code() {

for (inti = 0; i< 5; i++)

state[i] = THINKING;

} }

Each philosopher *I* invokes theoperations pickup()

and putdown() in the following sequence:

DiningPhilosophters.pickup (i);

EAT

DiningPhilosophers.putdown (i);

**Monitor Implementation Using Semaphores**

**Variables**

semaphoremutex; // (initially = 1)

semaphore next; // (initially = 0)

int next-count = 0;

nEach procedure ***F*** will be replaced by

wait(mutex);

…

body of *F*;

…

if (next\_count> 0)

signal(next)

else

signal(mutex);

nMutual exclusion within a monitor is ensured.

**Monitor Implementation**

For each condition variable ***x***, we have:

semaphorex\_sem; // (initially = 0)

int x-count = 0;

nThe operation x.wait can be implemented as:

x-count++;

if (next\_count> 0)

signal(next);

else

signal(mutex);

wait(x\_sem);

x-count--;

The operation x.signal can be implemented as:

if (x-count > 0) {

next\_count++;

signal(x\_sem);

wait(next);

next\_count--;

}

**A Monitor to Allocate Single Resource**

monitorResourceAllocator{

boolean busy;

condition x;

void acquire(int time) {

if (busy)

x.wait(time);

busy = TRUE;

}

void release() {

busy = FALSE;

x.signal();

}

initialization code() {

busy = FALSE;

} }

### UNIT-3

***System Model***

A system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs. Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc. Some categories may have a single resource.

In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:

* Request - If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ), malloc( ), new( ), and request( ).
* Use - The process uses the resource, e.g. prints to the printer or reads from the file.
* Release - The process relinquishes the resource. so that it becomes available for other processes. For example, close( ), free( ), delete( ), and release( ).

A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress. )

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



#### *Necessary Conditions*

There are four conditions that are necessary to achieve deadlock:

* **Mutual Exclusion** - At least one resource must be held in a non-sharable mode; If any other process requests this resource, then that process must wait for the resource to be released.
* **Hold and Wait** - A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.
* **No preemption** - Once a process is holding a resource ( i.e. once its request has been granted ), then that resource cannot be taken away from that process until the process voluntarily releases it.
* **Circular Wait** - A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[ i ] is waiting for P[ ( i + 1 ) % ( N + 1 ) ].

#### *Resource-Allocation Graph In some cases deadlocks can be understood more clearly through the use of Resource-Allocation Graphs, having the following properties:*

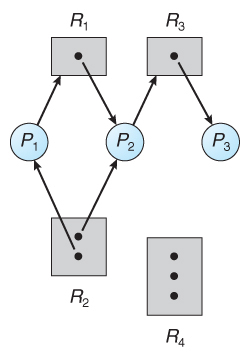
A set of resource categories, { R1, R2, R3, . . ., RN }, which appear as square nodes on the graph. Dots inside the resource nodes indicate specific instances of the resource. ( E.g. two dots might represent two laser printers. )

A set of processes, { P1, P2, P3, . . ., PN }

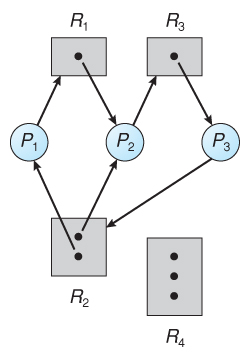
**Request Edges -** A set of directed arcs from Pi to Rj, indicating that process Pi has requested Rj, and is currently waiting for that resource to become available.

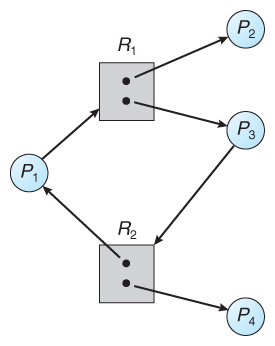
**Assignment Edges -** A set of directed arcs from Rj to Pi indicating that resource Rj has been allocated to process Pi, and that Pi is currently holding resource Rj.

Note that a **request edge** can be converted into an **assignment edge** by reversing the direction of the arc when the request is granted. For example:

  
**Figure : Resource allocation graph**

* If a resource-allocation graph contains no cycles, then the system is not deadlocked. If a resource-allocation graph does contain cycles **AND** each resource category contains only a single instance, then a deadlock exists.
* If a resource category contains more than one instance, then the presence of a cycle in the resource-allocation graph indicates the possibility of a deadlock, but does not guarantee one. Consider, for example,

  
**Figure : Resource allocation graph with a deadlock**

  
**Figure - Resource allocation graph with a cycle but no deadlock**

### *Methods for Handling Deadlocks*

Generally speaking there are three ways of handling deadlocks:

* Deadlock prevention or avoidance-Do not allow system to get into a deadlocked state.
* Deadlock detection and recovery - Abort a process or preempt some resources when deadlocks are detected.
* Ignore the problem all together - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process may request in future. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.

* If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes.

***Deadlock Prevention***

* Deadlocks can be prevented by preventing at least one of the four required conditions:

#### *Mutual Exclusion*

* Shared resources such as read-only files do not lead to deadlocks.
* Unfortunately some resources, such as printers and tape drives, require exclusive access by a single process.

#### *Hold and Wait*

To prevent this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others. There are several possibilities for this:

* Require that all processes request all resources at one time. This can be wasteful of system resources if a process needs one resource early in its execution and doesn't need some other resource until much later.
* Require that processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request. This can be a problem if a process has partially completed an operation using a resource and then fails to get it re-allocated after releasing it.
* Either of the methods described above can lead to starvation if a process requires one or more popular resources.

#### *No Preemption Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.*

* One approach is that if a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, ( preempted ), forcing this process to re-acquire the old resources along with the new resources in a single request, similar to the previous discussion.
* Another approach is that when a resource is requested and not available, then the system looks to see what other processes currently have those resources and are them blocked waiting for some other resource. If such a process is found, then some of their resources may get preempted and added to the list of resources for which the process is waiting.
* Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.

#### *Circular Wait*

* One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
* In order to request resource Rj, a process must first release all Ri such that i>= j.
* Big challenge in this scheme is determining the relative ordering of different resource

### *Deadlock Avoidance*

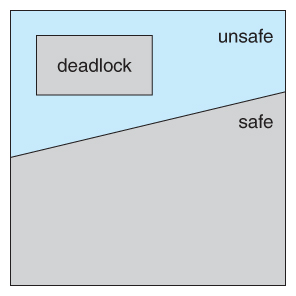
The general idea behind deadlock avoidance is to prevent deadlocks from ever happening, by preventing at least one of the aforementioned conditions.

* This requires more information about each process, and tends to lead to low device utilization. (i.e. it is a conservative approach. )
* In some algorithms the scheduler only needs to know the maximum number of each resource that a process might potentially use. In more complex algorithms the scheduler can also take advantage of the schedule of exactly what resources may be needed in what order.
* When a scheduler sees that starting a process or granting resource requests may lead to future deadlocks, then that process is just not started or the request is not granted.
* A resource allocation ***state*** is defined by the number of available and allocated resources and the maximum requirements of all processes in the system.

#### *Safe State*

A state is ***safe*** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state. A state is safe if there exists a ***safe sequence*** of processes { P0, P1, P2, ..., PN } such that all of the resource requests for Pi can be granted using the resources currently allocated to Pi and all processes Pj where j <i. ( i.e. if all the processes prior to Pi finish and free up their resources, then Pi will be able to finish also, using the resources that they have freed up. )

If a safe sequence does not exist, then the system is in an unsafe state, which ***MAY*** lead to deadlock. ( All safe states are deadlock free, but not all unsafe states lead to deadlocks. )

  
**Figure - Safe, unsafe, and deadlocked state spaces.**

For example, consider a system with 12 tape drives, allocated as follows. Is this a safe state? What is the safe sequence?

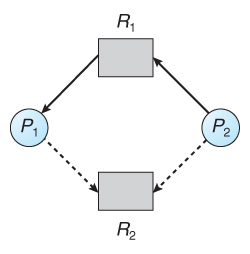
|  |  |  |
| --- | --- | --- |
|  | **Maximum Needs** | **Current Allocation** |
| **P0** | 10 | 5 |
| **P1** | 4 | 2 |
| **P2** | 9 | 2 |

* What happens to above table if process P2 requests & is granted one more tape drive?
* Key to the safe state approach is that when a request is made for resources, the request is granted only if the resulting allocation state is a safe one.

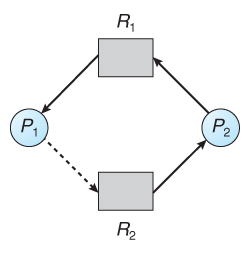
#### *Resource-Allocation Graph Algorithm*

* If resource categories have only single instances of their resources, then deadlock states can be detected by cycles in the resource-allocation graphs.
* In this case, unsafe states can be recognized and avoided by augmenting the resource-allocation graph with ***claim edges***, noted by dashed lines, which point from a process to a resource that it may request in the future.
* In order for this technique to work, all claim edges must be added to the graph for any particular process before that process is allowed to request any resources. (Alternatively, processes may only make requests for resources for which they have already established claim edges, and claim edges cannot be added to any process that is currently holding resources. )
* When a process makes a request, the claim edge Pi->Rj is converted to a request edge. Similarly when a resource is released, the assignment reverts back to a claim edge.
* This approach works by denying requests that would produce cycles in the resource-allocation graph, taking claim edges into effect.

Consider for example what happens when process P2 requests resource R2:

  
**Figure - Resource allocation graph for deadlock avoidance**

* The resulting resource-allocation graph would have a cycle in it, and so the request cannot be granted.

  
**Figure - An unsafe state in a resource allocation graph**

***Banker's Algorithm***

For resource categories that contain more than one instance the resource-allocation graph method does not work, and more complex (and less efficient) methods must be chosen.

* The Banker's Algorithm gets its name because it is a method that bankers could use to assure that when they lend out resources they will still be able to satisfy all their clients.
* When a process starts up, it must state in advance the maximum allocation of resources it may request, up to the amount available on the system.
* When a request is made, the scheduler determines whether granting the request would leave the system in a safe state. If not, then the process must wait until the request can be granted safely.

The banker's algorithm relies on several key data structures: (where n is the number of processes and m is the number of resource categories.)

* Available[ m ] indicates how many resources are currently available of each type.
* Max[ n ][ m ] indicates the maximum demand of each process of each resource.
* Allocation[ n ][ m ] indicates the number of each resource category allocated to each process.
* Need[ n ][ m ] indicates the remaining resources needed of each type for each process. ( Note that Need[ i ][ j ] = Max[ i ][ j ] - Allocation[ i ][ j ] for all i, j. )

For simplification of discussions, we make the following notations / observations:

* One row of the Need vector, Need[ i ], can be treated as a vector corresponding to the needs of process i, and similarly for Allocation and Max.
* A vector X is considered to be <= a vector Y if X[ i ] <= Y[ i ] for all i.

#### *Safety Algorithm*

In order to apply the Banker's algorithm, we first need an algorithm for determining whether or not a particular state is safe.This algorithm determines if the current state of a system is safe, according to the following steps:

1) Let Work and Finish be vectors of length m and n respectively.

* Work is a working copy of the available resources, which will be modified during the analysis.
* Finish is a vector of Boolean indicating whether a particular process can finish. ( or has finished so far in the analysis. )
* Initialize Work to Available, and Finish to false for all elements.

2) Find an i such that both (A) Finish[ i ] == false, and (B) Need[ i ] < Work. This process hasn’t finished,but could with the given available working set. If no such i exists, go to step4.

3) Set Work = Work + Allocation[ i ], and set Finish[ i ] to true. This corresponds to process i finishing up and releasing its resources back into the work pool.Then loop back to step 2.

4) If finish[ i ] == true for all i, then the state is a safe state, because a safe sequence is found.

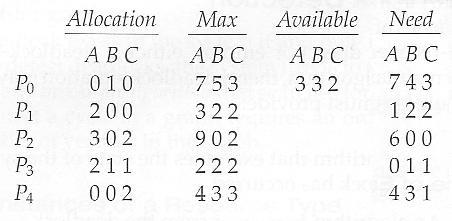
#### *Resource-Request Algorithm (The Bankers Algorithm)*

Now that we have a tool for determining if a particular state is safe or not, we are now ready to look at the Banker's algorithm itself. This algorithm determines if a new request is safe, and grants it only if it is safe to do so.

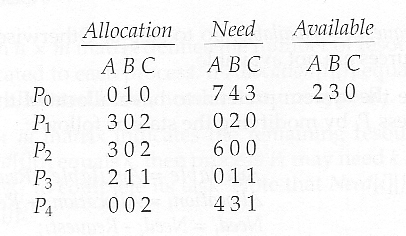
* When a request is made (that does not exceed currently available resources), pretend it has been granted, and then see if the resulting state is a safe one. If so, grant the request, and if not, deny the request, as follows:
* Let Request[ n ][ m ] indicate the number of resources of each type currently requested by processes. If Request[ i ] > Need[ i ] for any process i, raise an error condition.
* If Request[ i ] > Available for any process i, then that process must wait for resources to become available. Otherwise the process can continue to step 3.
* Check to see if the request can be granted safely, by pretending it has been granted and then seeing if the resulting state is safe. If so, grant the request, and if not, then the process must wait until its request can be granted safely. The procedure for granting a request ( or pretending to for testing purposes ) is:
  + - Available = Available - Request
    - Allocation = Allocation + Request
    - Need = Need - Request

#### *An Illustrative Example*

* Consider the following situation:



* And now consider what happens if process P1 requests 1 instance of A and 2 instances of C. ( Request[ 1 ] = ( 1, 0, 2 ) )



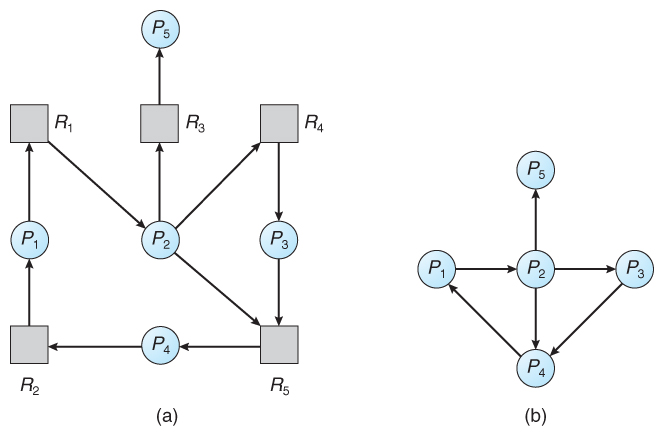
* What about requests of ( 3, 3,0 ) by P4? or ( 0, 2, 0 ) by P0? Can these be safely granted? Why or why not?

### *Deadlock Detection*

* If deadlocks are not avoided, then another approach is to detect when they have occurred and recover somehow.
* In addition to the performance hit of constantly checking for deadlocks, a policy / algorithm must be in place for recovering from deadlocks, and there is potential for lost work when processes must be aborted or have their resources preempted.

#### Single Instance of Each Resource Type

* If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a ***wait-for graph***.
* A wait-for graph can be constructed from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
* An arc from Pi to Pj in a wait-for graph indicates that process Pi is waiting for a resource that process Pj is currently holding.

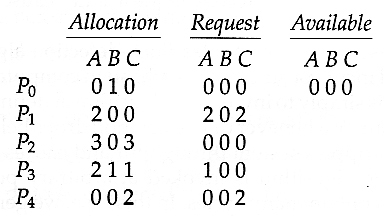
  
**Figure - (a) Resource allocation graph. (b) Corresponding wait-for graph**

* As before, cycles in the wait-for graph indicate deadlocks.
* This algorithm must maintain the wait-for graph, and periodically search it for cycles.

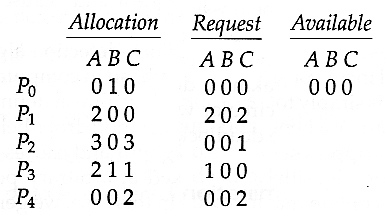
#### *Several Instances of a Resource Type*

The detection algorithm outlined here is essentially the same as the Banker's algorithm, with two subtle differences:

* In step 1, the Banker's Algorithm sets Finish[ i ] to false for all i. The algorithm presented here sets Finish[ i ] to false only if Allocation[ i ] is not zero. If the currently allocated resources for this process are zero, the algorithm sets Finish[ i ] to true. This is essentially assuming that IF all of the other processes can finish, then this process can finish also. Furthermore, this algorithm is specifically looking for which processes are involved in a deadlock situation, and a process that does not have any resources allocated cannot be involved in a deadlock, and so can be removed from any further consideration.
* Steps 2 and 3 are unchanged
* In step 4, the basic Banker's Algorithm says that if Finish[ i ] == true for all i, that there is no deadlock. This algorithm is more specific, by stating that if Finish[ i ] == false for any process Pi, then that process is specifically involved in the deadlock which has been detected.
* Consider, for example, the following state, and determine if it is currently deadlocked:



* Now suppose that process P2 makes a request for an additional instance of type C, yielding the state shown below. Is the system now deadlocked?



### *Recovery from Deadlock*

There are three basic approaches to recovery from deadlock:

* Inform the system operator, and allow him/her to take manual intervention.
* Terminate one or more processes involved in the deadlock.
* Preempt resources.

#### Process Termination

Two basic approaches, both of which recover resources allocated to terminate processes:

* Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
* Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.
* In the latter case there are many factors that can go into deciding which processes to terminate next:
  + Process priorities.
  + How long the process has been running, and how close it is to finishing.
  + How many and what type of resources is the process holding. ( Are they easy to preempt and restore? )
  + How many more resources does the process need to complete.
  + How many processes will need to be terminated
  + Whether the process is interactive or batch.
  + Whether or not the process has made non-restorable changes to any resource.

#### Resource Preemption

When preempting resources to relieve deadlock, there are three important issues to be addressed:

* **Selecting a victim** - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined.
* **Rollback** - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. ( i.e. abort the process and make it start over. )
* **Starvation** - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Memory Management in Operating System

**Computer Memory** is basically known as a collection of data that is represented in binary format**. Main Memory** refers to a physical memory that is the internal memory of the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in the main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memory for execution. Sometimes the complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, which enhances the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, the CPU links the dependent programs to the main executing program when required. This mechanism is known as **Dynamic Linking**.

Let us understand how the data is being stored in the computer system.

What is Memory Management in Operating System?

Memory Management is the process of coordinating and controlling the memory in a computer, Blocks are assigning portions that are assigned to various running programs in order to optimize the overall performance of the system. This technique helps in keeping the track of each and every memory location, whether the memory location is allocated to some process or it is free.

This technique decides which process will get memory at what time. It also keeps the count of how much memory can be allocated to a process. As it keeps the tracks of everything so whenever some memory gets freed or unallocated it updates the status correspondingly.

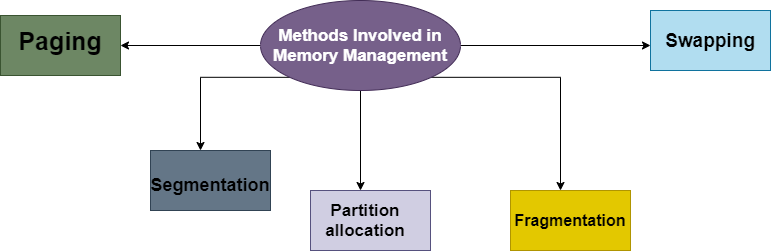
Need for Memory Management in OS

Memory management technique is needed for the following reasons:

* This technique helps in placing the programs in memory in such a way so that memory is utilized at its fullest extent.
* This technique helps to protect different processes from each other so that they do not interfere with each other's operations.
* It helps to allocate space to different application routines.
* This technique allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
* It keeps the track of each memory location whether it is free or allocated.
* This technique keeps the track of inventory whenever memory gets freed or unallocated and it will update the status accordingly.

Methods Involved in Memory Management

There are various methods and with their help Memory Management can be done intelligently by the Operating System:



Swapping in OS

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, the excess process is kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in the main memory, running it for a while, and then putting it back to the disk.

Contiguous Memory Allocation

In contiguous memory allocation, each process is contained in a single contiguous block of memory. Memory is divided into several fixed-size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

Memory Protection

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

Memory Allocation in OS

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

1. **First Fit Allocation**

The first hole that is big enough is allocated to the program.

1. **Best Fit Allocation**

The smallest hole that is big enough is allocated to the program.

1. **Worst Fit Allocation**

The largest hole that is big enough is allocated to the program.

Fragmentation in OS

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as the inability to use the available memory.

In such a situation, processes are loaded and removed from the memory. As a result of this, free holes exist to satisfy a request but are non-contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation.**

Also, at times the physical memory is broken into fixed-size blocks, and memory is allocated in the unit of block sizes. The memory allocated to space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

Paging

A solution to the fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between the *virtual address* and *physical addresses.*

The virtual address is also known as the Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Segmentation in OS

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows the breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

Segmentation with Paging

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into the following 3 parts :

* Segment numbers(S)
* Page number (P)
* The displacement or offset number (D)

Logical and Physical Address Space in Operating Systems

In this tutorial, we will be covering the concept of logical and physical address space.

The address that is generated by the CPU is commonly referred to as the **Logical Address**. It is basically a virtual address. The logical address is basically the address of an instruction or data used by any program.

The set of all logical addresses that are generated by any program is referred to as**Logical Address Space.**

The address that is loaded into the**memory-address** **register** of the memory is commonly referred to as a **Physical address**. A physical address cannot be accessed by the user directly but the user can calculate the physical address with the help of a **Logical address.**

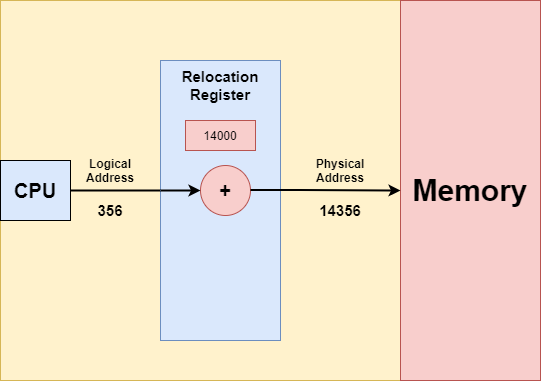
The user's program mainly generates the logical address and the user thinks that the program is running in this logical address but the program mainly needs physical memory in order to complete its execution.

The set of all**physical addresses** corresponding to the**Logical addresses** is commonly known as**Physical Address Space.**

Memory Management Unit(MMU) in OS

It is a hardware device that does the run-time mapping from the virtual address to the physical address. It is located within the Central Processing Unit.

Let us understand the concept of mapping with the help of a simple MMU scheme and that is a **base-register scheme**.



In the above diagram, the base register is termed the **Relocation register.** The relocation register is a special register in the CPU and is used for the mapping of logical addresses used by a program to physical addresses of the system's main memory.

The value in the relocation register is added to every address that is generated by the user process at the time when the address is sent to the memory.

MMU Example

Suppose the **base is at 14000**, then an attempt by the user to address location **0** is relocated dynamically to **14000**; thus access to location **356** is mapped to **14356.**

It is important to note that the user program never sees the real physical addresses. The Program can create a pointer to location 356 and store it in the memory and then manipulate it after that compare it with other addresses as number 356.

User program always deals with the logical addresses. The **Memory Mapping unit** mainly converts the logical addresses into the physical addresses. The final location of the referenced memory address is not determined**until the reference is made.**

There are two types of addresses that we have:

**logical addresses**(lies in the range 0 to max).

**physical addresses**(lies in the range R+0 to R+max for the base value R)

As we have told you above the user generates only a logical address and then thinks that the process runs in locations **0 to max.**However, these logical addresses must be mapped to physical addresses before they are used.

Difference Between Logical Address and Physical Address

Let us now cover the differences between the Logical addresses and Physical addresses in the Operating System

| **S.No** | **Logical Address** | **Physical Address** |
| --- | --- | --- |
| **1.** | Users can access the **logical address** of the Program. | User can never access the **physical address** of the Program |
| **2.** | The logical address is generated by the CPU. | The physical address is located in the memory unit. |
| **3.** | The user can access the physical address with the help of a logical address. | A physical address can be accessed by a user indirectly b ut not directly. |
| **4.** | The logical address does not exist physically in the memory and thus termed as a Virtual address. | On the other hand, the physical address is a location in the memory. Thus it can be accessed physically. |
| **5.** | The set of all logical addresses that are generated by any program is referred to as**Logical Address Space.** | The set of all**physical addresses** corresponding to the**Logical addresses** is commonly known as**Physical Address Space.** |
| **6.** | This address is generated by the CPU. | It is computed by the Memory Management Unit(MMU). |

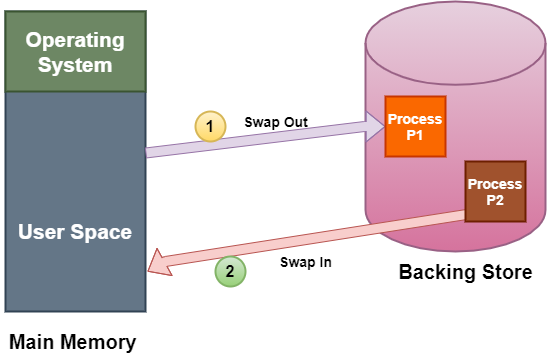
Swapping in Operating System

In this tutorial, we will be covering the concept of swapping in the Operating system.

Swapping is a memory management technique and is used to temporarily remove the inactive programs from the main memory of the computer system. Any process must be in the memory for its execution, but can be swapped temporarily out of memory to a backing store and then again brought back into the memory to complete its execution. Swapping is done so that other processes get memory for their execution.

Due to the swapping technique performance usually gets affected, but it also helps in running multiple and big processes in parallel. **The swapping** process is also known as a technique for **memory compaction.**Basically, low priority processes may be swapped out so that processes with a higher priority may be loaded and executed.

Let us understand this technique with the help of a figure given below:



The above diagram shows swapping of two processes where the disk is used as a Backing store.

In the above diagram, suppose there is a multiprogramming environment with a round-robin scheduling algorithm; whenever the time quantum expires then the memory manager starts to swap out those processes that are just finished and swap another process into the memory that has been freed. And in the meantime, the CPU scheduler allocates the time slice to some other processes in the memory.

The swapping of processes by the memory manager is fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU.

A variant of the swapping technique is the priority-based scheduling algorithm. If any higher-priority process arrives and wants service, then the memory manager swaps out lower priority processes and then load the higher priority processes and then execute them. When the process with higher priority finishes .then the process with lower priority swapped back in and continues its execution. This variant is sometimes known as roll in and roll out.

There are two more concepts that come in the swapping technique and these are:**swap in**and **swap out.**

Swap In and Swap Out in OS

The procedure by which any process gets removed from the **hard disk**and placed in the **main memory or RAM**commonly known as **Swap In.**

On the other hand, **Swap Out** is the method of removing a process from the **main memory or RAM** and then adding it to the **Hard Disk.**

Advantages of Swapping

The advantages/benefits of the Swapping technique are as follows:

1. The swapping technique mainly helps the CPU to manage multiple processes within a single main memory.
2. This technique helps to create and use virtual memory.
3. With the help of this technique, the CPU can perform several tasks simultaneously. Thus, processes need not wait too long before their execution.
4. This technique is economical.
5. This technique can be easily applied to priority-based scheduling in order to improve its performance.

Disadvantages of Swapping

The drawbacks of the swapping technique are as follows:

1. There may occur inefficiency in the case if a resource or a variable is commonly used by those processes that are participating in the swapping process.
2. If the algorithm used for swapping is not good then the overall method can increase the number of page faults and thus decline the overall performance of processing.
3. If the computer system loses power at the time of high swapping activity then the user might lose all the information related to the program.

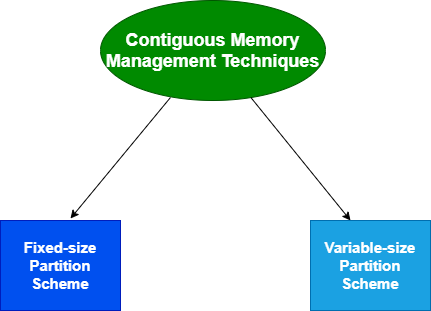
# Contiguous Memory Allocation in Operating System

In this tutorial, we will cover the concept of Contiguous Memory Allocation in an Operating System.

In the Contiguous Memory Allocation, each process is contained in a single contiguous section of memory. In this memory allocation, all the available memory space remains together in one place which implies that the freely available memory partitions are not spread over here and there across the whole memory space.

In **Contiguous memory allocation** which is a memory management technique, whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement. Contiguous Memory allocation is achieved just by dividing the memory into the **fixed-sized partition.**

The memory can be divided either in the **fixed-sized partition or in the variable-sized partition** in order to allocate contiguous space to user processes.



We will cover the concept of different Contiguous Memory allocation techniques one by one.

## Fixed-size Partition Scheme

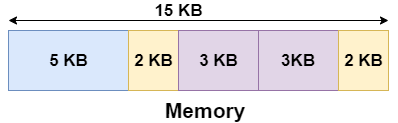
This technique is also known as**Static partitioning**. In this scheme, the system divides the memory into fixed-size partitions. The partitions may or may not be the same size. The size of each partition is fixed as indicated by the name of the technique and it cannot be changed.

In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

Whenever any process terminates then the partition becomes available for another process.

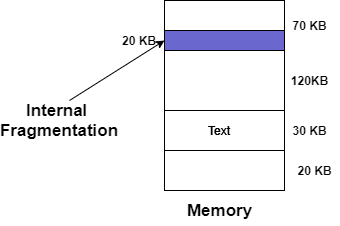
### Example

Let's take an example of fixed size partitioning scheme, we will divide a memory size of 15 KB into fixed-size partitions:



It is important to note that these partitions are allocated to the processes as they arrive and the partition that is allocated to the arrived process basically depends on the algorithm followed.

If there is some wastage inside the partition then it is termed **Internal Fragmentation**.



### Advantages of Fixed-size Partition Scheme

* This scheme is simple and is easy to implement
* It supports multiprogramming as multiple processes can be stored inside the main memory.
* Management is easy using this scheme

### Disadvantages of Fixed-size Partition Scheme

Some disadvantages of using this scheme are as follows:

**1. Internal Fragmentation**

Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation

As we have shown in the above diagram the 70 KB partition is used to load a process of 50 KB so the remaining 20 KB got wasted.

**2**. **Limitation on the size of the process**

If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory. Due to this, a condition is imposed on the size of the process and it is: the size of the process cannot be larger than the size of the largest partition.

**3. External Fragmentation**

It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

**4. Degree of multiprogramming is less**

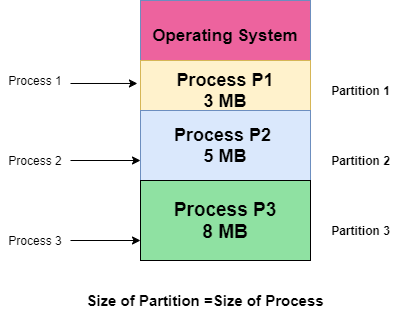
In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

## Variable-size Partition Scheme

This scheme is also known as **Dynamic partitioning** and is came into existence to overcome the drawback i.e internal fragmentation that is caused by **Static partitioning**. In this partitioning, scheme allocation is done dynamically.

The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.

As partition size varies according to the need of the process so in this partition scheme there is no **internal fragmentation.**



### Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

1. **No Internal Fragmentation** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
2. **Degree of Multiprogramming is Dynamic** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
3. **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

### Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

1. **External Fragmentation** As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let’s there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.
2. **Difficult Implementation** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration. As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

# Paging in Operating Systems

In this tutorial, we will be covering the concept of Paging in the Operating System.

Paging permits the **physical address space** of a process to be **non-contiguous.**It is a **fixed-size partitioning scheme**. In the Paging technique, the secondary memory and main memory are divided into equal fixed-size partitions.

Paging solves the problem of fitting memory chunks of varying sizes onto the backing store and this problem is suffered by many memory management schemes.

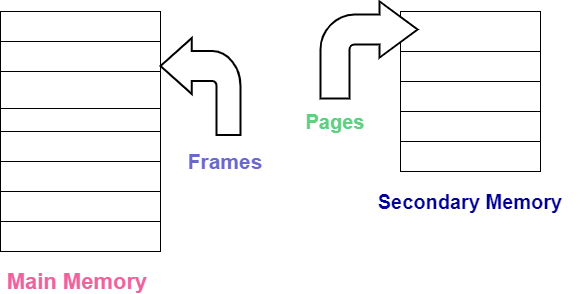
Paging helps to **avoid external fragmentation**and the**need for compaction**.

## Basic Method of Paging

The paging technique divides the physical memory(main memory) into fixed-size blocks that are known as**Frames** and also divide the **logical memory(secondary memory) into blocks of the same size** that are known as**Pages.**

This technique keeps the track of all the free frames.

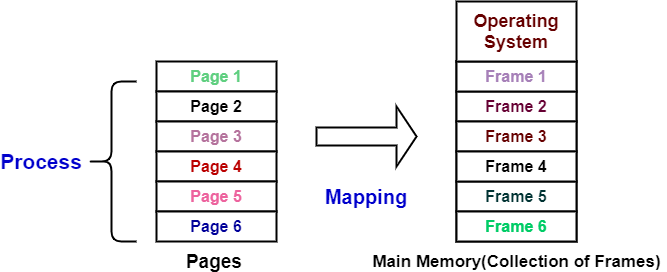
The Frame has the same size as that of a Page. A frame is basically a place where a (logical) page can be (physically) placed.



Each process is mainly divided into parts where the size of each part is the same as the page size.

There is a possibility that the size of the last part may be less than the page size.

* Pages of a process are brought into the main memory only when there is a requirement otherwise they reside in the secondary storage.
* One page of a process is mainly stored in one of the frames of the memory. Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames.



Let us now cover the concept of **translating a logical address into the physical address:**

## Translation of Logical Address into Physical Address

Before moving on further there are some important points to note:

* The CPU always generates a logical address.
* In order to access the main memory always a physical address is needed.

The**logical address generated by CPU always consists of two parts:**

1. Page Number(p)
2. Page Offset (d)

where,

**Page Number** is used to specify the specific page of the process from which the CPU wants to read the data. and it is also used as an index to the page table.

and **Page offset** is mainly used to specify the specific word on the page that the CPU wants to read.

Now let us understand **what is Page Table?**

## Page Table in OS

The Page table mainly**contains the base address of each page** in the Physical memory. The base address is then combined with the page offset in order to define the**physical memory address** which is then sent to the memory unit.

Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

As we have told you above that the **frame number** is combined with the **page offset**and forms the**required physical address.**

So, **The physical address consists of two parts:**

1. Page offset(d)
2. Frame Number(f)

where,

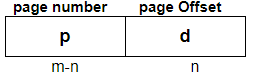
The **Frame number** is used to indicate the specific frame where the required page is stored.

and **Page Offset** indicates the specific word that has to be read from that page.

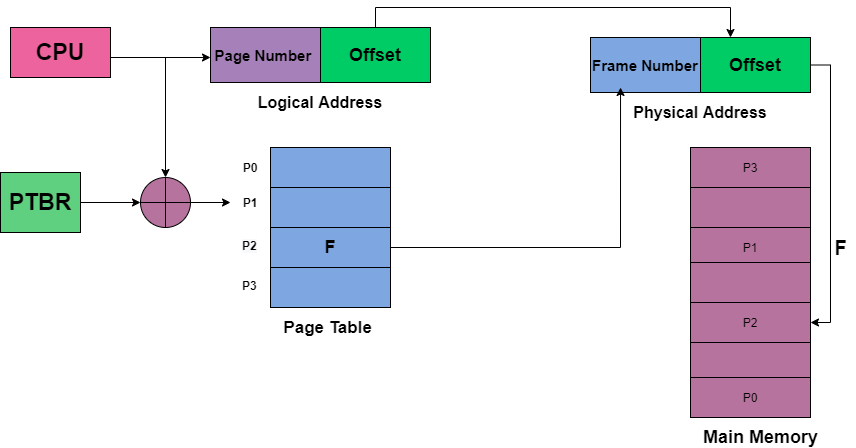
The Page size (like the frame size) is defined with the help of hardware. It is important to note here that the size of the page is typically the power of 2 that varies between 512 bytes and 16 MB per page and it mainly depends on the architecture of the computer.

If the size of **logical address space** is 2 raised to the power m and **page size** is2 raised to the power naddressing units then the high order m-n bits of**logical address designates the page number** and the n low-order **bits designate the page offset**.

The logical address is as follows:



where **p** indicates the index into the page table, and **d** indicates the displacement within the page.



The above diagram indicates the translation of the Logical address into the Physical address. The **PTBR** in the above diagram means page table base register and it basically holds the base address for the page table of the current process.

The **PTBR** is mainly a processor register and is managed by the operating system. Commonly, each process running on a processor needs its own logical address space.

But there is a problem with this approach and that is with the time required to access a user memory location. Suppose if we want to find the location i, we must first find the index into the page table by using the value in the PTBR offset by the page number for I. And this task requires memory access. It then provides us the frame number which is combined with the page offset in order to produce the actual address. After that, we can then access the desired place in the memory.

With the above scheme, two memory accesses are needed in order to access a byte( one for the page-table entry and one for byte). Thus memory access is slower by a factor of 2 and in most cases, this scheme slowed by a factor of 2.

## Translation of look-aside buffer(TLB)

There is the standard solution for the above problem that is to use a special, small, and fast-lookup hardware cache that is commonly known as Translation of look-aside buffer(TLB).

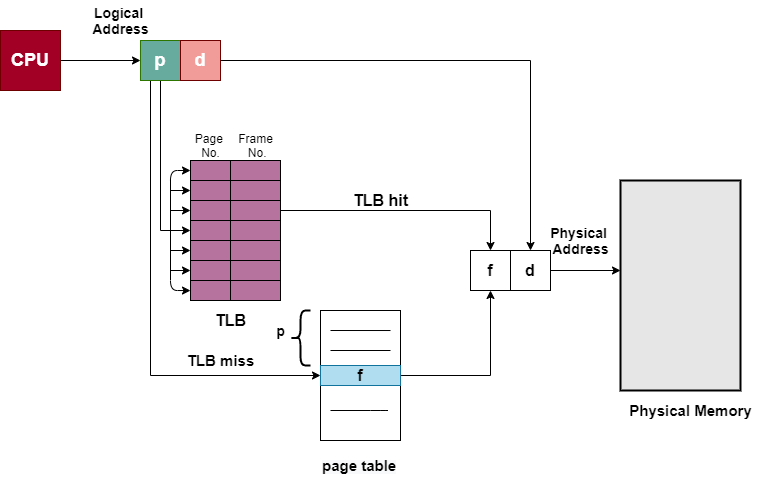
* TLB is associative and high-speed memory.
* Each entry in the TLB mainly consists of two parts: a key(that is the tag) and a value.
* When associative memory is presented with an item, then the item is compared with all keys simultaneously. In case if the item is found then the corresponding value is returned.
* The search with TLB is fast though the hardware is expensive.
* The number of entries in the TLB is small and generally lies in between 64 and 1024.

**TLB** is used with **Page Tables** in the following ways:

The TLB contains only a few of the page-table entries. Whenever the logical address is generated by the CPU then its page number is presented to the TLB.

* If the page number is found, then its frame number is immediately available and is used in order to access the memory. The above whole task may take less than 10 percent longer than would if an unmapped memory reference were used.
* In case if the page number is not in the TLB (which is known as**TLB miss**), then a memory reference to the Page table must be made.
* When the frame number is obtained it can be used to access the memory. Additionally, page number and frame number is added to the TLB so that they will be found quickly on the next reference.
* In case if the **TLB is already full of entries** then the Operating system must select o**ne for replacement.**
* TLB allows some entries to **be wired down**, which means they cannot be removed from the TLB. Typically TLB entries for the kernel code are **wired down.**

## Paging Hardware With TLB



## Advantages of Paging

Given below are some advantages of the Paging technique in the operating system:

* Paging mainly allows to storage of parts of a single process in a **non-contiguous fashion**.
* With the help of **Paging, the problem** of external fragmentation is solved.
* Paging is one of the simplest algorithms for memory management.

## Disadvantages of Paging

Disadvantages of the Paging technique are as follows:

* In Paging, sometimes the page table consumes more memory.
* Internal fragmentation is caused by this technique.
* There is an increase in time taken to fetch the instruction since now two memory accesses are required.

## Paging Hardware

Every address generated by**CPU mainly consists of two parts:**

1. Page Number(p)
2. Page Offset (d)

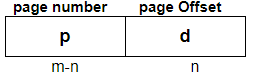
where,

**Page Number** is used as an index **into the page table**that generally contains the base address of each page in the physical memory.

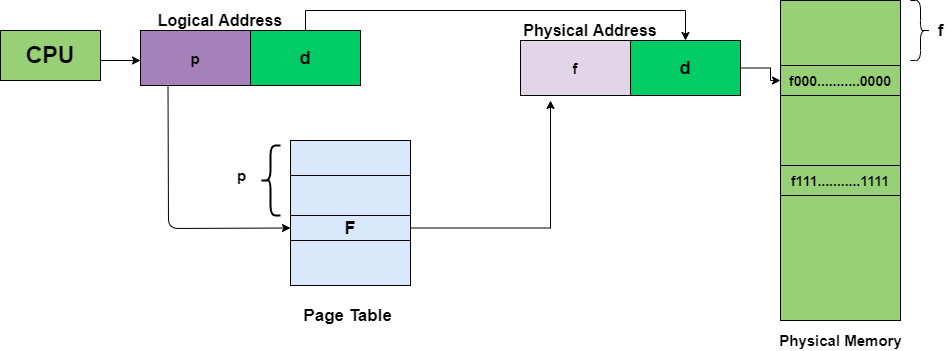
**Page offset**is combined with base address in order to define the physical memory address which is then sent to the memory unit.

If the size of **logical address space** is 2 raised to the power m and **page size** is2 raised to the power naddressing units then the high order m-n bits of**logical address designates the page number** and the n low-order **bits designate the page offset**.

The logical address is as follows:



where p indicates the index into the page table, and d indicates the displacement within the page. The Page size is usually defined by the hardware. The size of the page is typically the power of 2 that varies between 512 bytes and 16 MB per page.



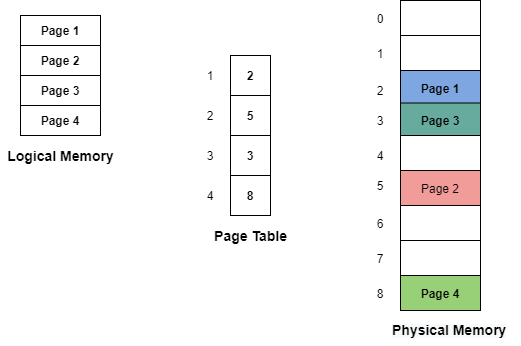
Now its time to cover an example of Paging:

## Paging Example

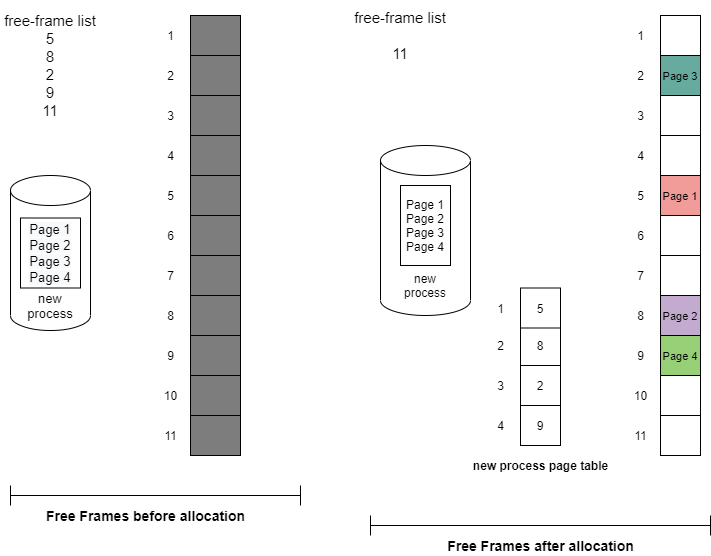
In order to implement paging, one of the simplest methods is to implement the page table as a set of registers. As the size of registers is limited but the size of the page table is usually large thus page table is kept in the main memory.

There is no External fragmentation caused due to this scheme; Any frame that is free can be allocated to any process that needs it. But the internal fragmentation is still there.

* If any process requires n pages then at least n frames are required.
* The **first page** of the process is **loaded into the first frame** that is listed on the **free-frame list**, and then the **frame number** is put into the **page table.**



**The frame table** is a data structure that keeps the information of which frames are allocated or which frames are available and many more things. This table mainly has one entry for each physical page frame.



The Operating system maintains a copy of the page table for each process in the same way as it maintains the copy of the instruction counter and register contents. Also, this copy is used to translate logical addresses to physical addresses whenever the operating system maps a logical address to a physical address manually.

This copy is also used by the CPU dispatcher in order to define the hardware page table whenever a process is to be allocated to the CPU.

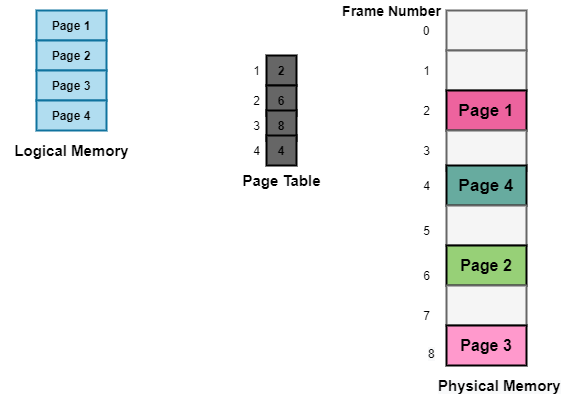
Structure of Page Table in Operating Systems

In this tutorial, we will cover some of the most common techniques used for structuring the Page table.

The data structure that is used by the virtual memory system in the operating system of a computer in order to store the mapping between physical and logical addresses is commonly known as **Page Table**.

As we had already told you that the logical address that is generated by the CPU is translated into the physical address with the help of the page table.

* Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.



The above diagram shows the paging model of Physical and logical memory.

Characteristics of the Page Table

Some of the characteristics of the Page Table are as follows:

* It is stored in the main memory.
* Generally; the Number of entries in the page table = the Number of Pages in which the process is divided.
* **PTBR** means page table base register and it is basically used to hold the base address for the page table of the current process.
* Each process has its own independent page table.

Techniques used for Structuring the Page Table

Some of the common techniques that are used for structuring the Page table are as follows:

1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables

Let us cover these techniques one by one;

Hierarchical Paging

Another name for Hierarchical Paging is multilevel paging.

* There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.
* In this type of Paging the logical address space is broke up into Multiple page tables.
* Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.

Two Level Page Table

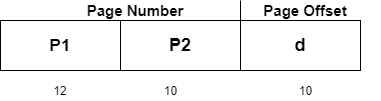
Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:

* Page Number consisting of 22 bits.
* Page Offset consisting of 10 bits.

As we page the Page table, the page number is further divided into :

* Page Number consisting of 12 bits.
* Page Offset consisting of 10 bits.

Thus the Logical address is as follows:



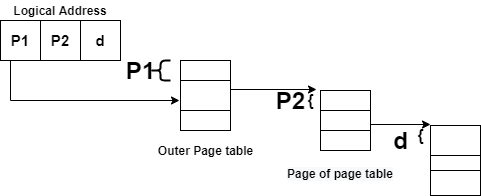
In the above diagram,

P1 is an index into the **Outer Page** table.

P2 indicates the displacement within the page of the **Inner page** Table.

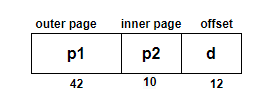
As address translation works from outer page table inward so is known as **forward-mapped Page Table**.

Below given figure below shows the Address Translation scheme for a two-level page table

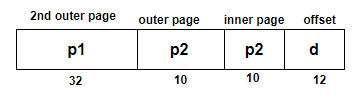


Three Level Page Table

For a system with 64-bit logical address space, a two-level paging scheme is not appropriate. Let us suppose that the page size, in this case, is 4KB.If in this case, we will use the two-page level scheme then the addresses will look like this:



Thus in order to avoid such a large table, there is a solution and that is to divide the outer page table, and then it will result in a **Three-level page table:**



Hashed Page Tables

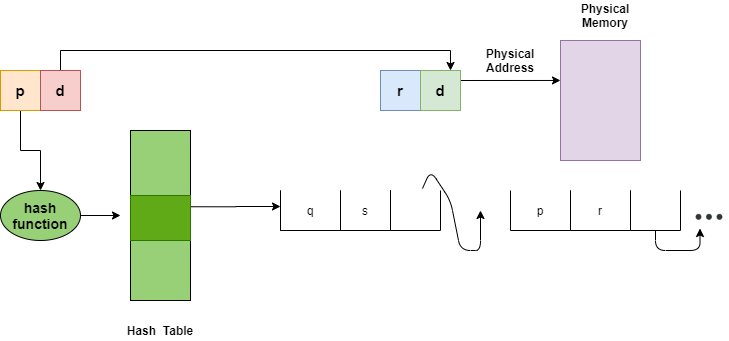
This approach is used to handle address spaces that are larger than 32 bits.

* In this virtual page, the number is hashed into a page table.
* This Page table mainly contains a chain of elements hashing to the same elements.

Each element mainly consists of :

1. The virtual page number
2. The value of the mapped page frame.
3. A pointer to the next element in the linked list.

Given below figure shows the address translation scheme of the Hashed Page Table:



The above Figure shows Hashed Page Table

The Virtual Page numbers are compared in this chain searching for a match; if the match is found then the corresponding physical frame is extracted.

In this scheme, a variation for 64-bit address space commonly uses **clustered page tables**.

Clustered Page Tables

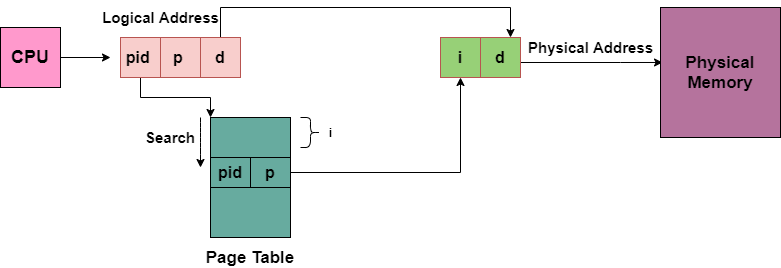
* These are similar to hashed tables but here each entry refers to several pages (that is 16) rather than 1.
* Mainly used for sparse address spaces where memory references are non-contiguous and scattered

Inverted Page Tables

The Inverted Page table basically combines A page table and A frame table into a single data structure.

* There is one entry for each virtual page number and a real page of memory
* And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
* Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

Given below figure shows the address translation scheme of the Inverted Page Table:



In this, we need to keep the track of process id of each entry, because many processes may have the same logical addresses.

Also, many entries can map into the same index in the page table after going through the hash function. Thus chaining is used in order to handle this.

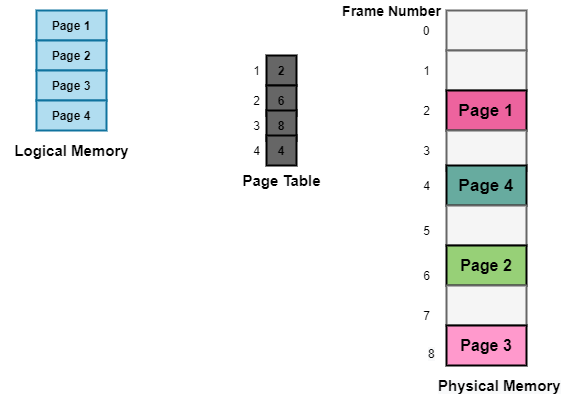
Structure of Page Table in Operating Systems

In this tutorial, we will cover some of the most common techniques used for structuring the Page table.

The data structure that is used by the virtual memory system in the operating system of a computer in order to store the mapping between physical and logical addresses is commonly known as **Page Table**.

As we had already told you that the logical address that is generated by the CPU is translated into the physical address with the help of the page table.

* Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.



The above diagram shows the paging model of Physical and logical memory.

Characteristics of the Page Table

Some of the characteristics of the Page Table are as follows:

* It is stored in the main memory.
* Generally; the Number of entries in the page table = the Number of Pages in which the process is divided.
* **PTBR** means page table base register and it is basically used to hold the base address for the page table of the current process.
* Each process has its own independent page table.

Techniques used for Structuring the Page Table

Some of the common techniques that are used for structuring the Page table are as follows:

1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables

Let us cover these techniques one by one;

Hierarchical Paging

Another name for Hierarchical Paging is multilevel paging.

* There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.
* In this type of Paging the logical address space is broke up into Multiple page tables.
* Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.

Two Level Page Table

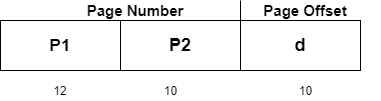
Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:

* Page Number consisting of 22 bits.
* Page Offset consisting of 10 bits.

As we page the Page table, the page number is further divided into :

* Page Number consisting of 12 bits.
* Page Offset consisting of 10 bits.

Thus the Logical address is as follows:



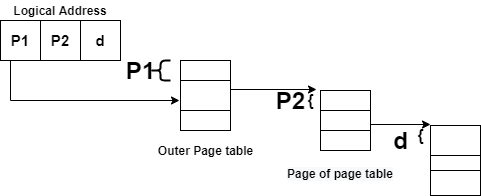
In the above diagram,

P1 is an index into the **Outer Page** table.

P2 indicates the displacement within the page of the **Inner page** Table.

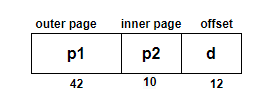
As address translation works from outer page table inward so is known as **forward-mapped Page Table**.

Below given figure below shows the Address Translation scheme for a two-level page table

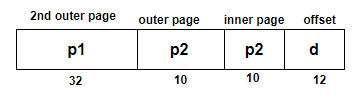


Three Level Page Table

For a system with 64-bit logical address space, a two-level paging scheme is not appropriate. Let us suppose that the page size, in this case, is 4KB.If in this case, we will use the two-page level scheme then the addresses will look like this:



Thus in order to avoid such a large table, there is a solution and that is to divide the outer page table, and then it will result in a **Three-level page table:**



Hashed Page Tables

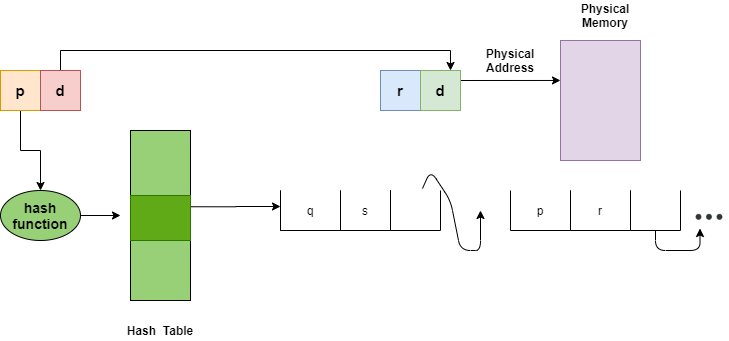
This approach is used to handle address spaces that are larger than 32 bits.

* In this virtual page, the number is hashed into a page table.
* This Page table mainly contains a chain of elements hashing to the same elements.

Each element mainly consists of :

1. The virtual page number
2. The value of the mapped page frame.
3. A pointer to the next element in the linked list.

Given below figure shows the address translation scheme of the Hashed Page Table:



The above Figure shows Hashed Page Table

The Virtual Page numbers are compared in this chain searching for a match; if the match is found then the corresponding physical frame is extracted.

In this scheme, a variation for 64-bit address space commonly uses **clustered page tables**.

Clustered Page Tables

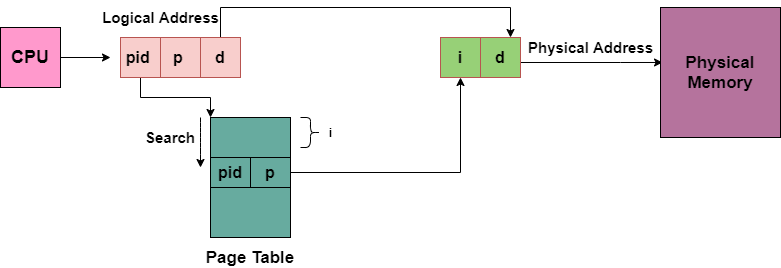
* These are similar to hashed tables but here each entry refers to several pages (that is 16) rather than 1.
* Mainly used for sparse address spaces where memory references are non-contiguous and scattered

Inverted Page Tables

The Inverted Page table basically combines A page table and A frame table into a single data structure.

* There is one entry for each virtual page number and a real page of memory
* And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
* Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

Given below figure shows the address translation scheme of the Inverted Page Table:



In this, we need to keep the track of process id of each entry, because many processes may have the same logical addresses.

Also, many entries can map into the same index in the page table after going through the hash function. Thus chaining is used in order to handle this.

Segmentation in Operating Systems

In this tutorial, we will be covering segmentation in the Operating System.

Segmentation is another way of dividing the addressable memory. It is another scheme of memory management and it generally supports the user view of memory. The Logical address space is basically the collection of segments. Each segment has a name and a length.

Basically, a process is divided into segments. Like paging, segmentation divides or segments the memory. But there is a difference and that is while the**paging** divides the memory into a **fixed size** and on the other hand, segmentation divides the **memory into variable segments** these are then loaded into logical memory space.

A Program is basically a collection of segments. And a segment is a logical unit such as:

* main program
* procedure
* function
* method
* object
* local variable and global variables.
* symbol table
* common block
* stack
* arrays

Types of Segmentation

Given below are the types of Segmentation:

* **Virtual Memory Segmentation**With this type of segmentation, each process is segmented into n divisions and the most important thing is they are not segmented all at once.
* **Simple Segmentation**With the help of this type, each processis segmented into n divisions and they are all together segmented at once exactly but at the runtime and can be non-contiguous (that is they may be scattered in the memory).

Characteristics of Segmentation

Some characteristics of the segmentation technique are as follows:

* The Segmentation partitioning scheme is **variable-size.**
* Partitions of the secondary memory are commonly known as **segments**.
* Partition size mainly depends upon the length of modules.
* Thus with the help of this technique, secondary memory and main memory are divided into unequal-sized partitions.

Need of Segmentation

One of the important drawbacks of memory management in the Operating system is the separation of the user's view of memory and the actual physical memory. and Paging is a technique that provides the separation of these two memories.

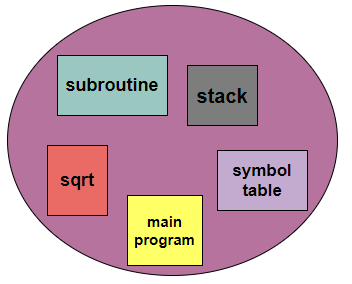
User'view is basically mapped onto physical storage. And this mapping allows differentiation between Physical and logical memory.

It is possible that the operating system divides the same function into different pages and those pages may or may not be loaded into the memory at the same time also Operating system does not care about the User's view of the process. Due to this technique system's efficiency decreases.

Segmentation is a better technique because it divides the process into segments.

User's View of a Program

Given below figure shows the user's view of segmentation:



**logical address**

Basic Method

A computer system that is using segmentation has a logical address space that can be viewed as multiple segments. And the size of the segment is of the variable that is it may grow or shrink. As we had already told you that during the execution each segment has a name and length. And the address mainly specifies both thing name of the segment and the displacement within the segment.

Therefore the user specifies each address with the help of two quantities: segment name and offset.

For simplified Implementation segments are numbered; thus referred to as segment number rather than segment name.

Thus the logical address consists of two tuples:

**<segment-number,offset>**

where,

**Segment Number(s):** Segment Number is used to represent the number of bits that are required to represent the segment.

**Offset(d)** Segment offset is used to represent the number of bits that are required to represent the size of the segment.

Segmentation Architecture

Segment Table

A Table that is used to store the information of all segments of the process is commonly known as Segment Table. Generally, there is no simple relationship between logical addresses and physical addresses in this scheme.

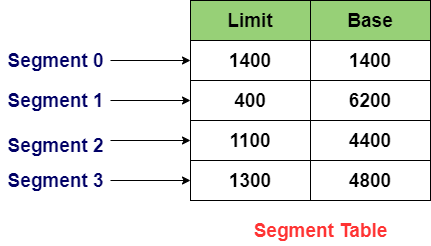
* The mapping of a two-dimensional Logical address into a one-dimensional Physical address is done using the segment table.
* This table is mainly stored as a separate segment in the main memory.
* The table that stores the base address of the segment table is commonly known as the Segment table base register (STBR)

In the segment table each entry has :

1. **Segment Base/base address:** The segment base mainly contain**s** the starting physical address where the segments reside in the memory.
2. **Segment Limit:** The segment limit is mainly used to specify the length of the segment.

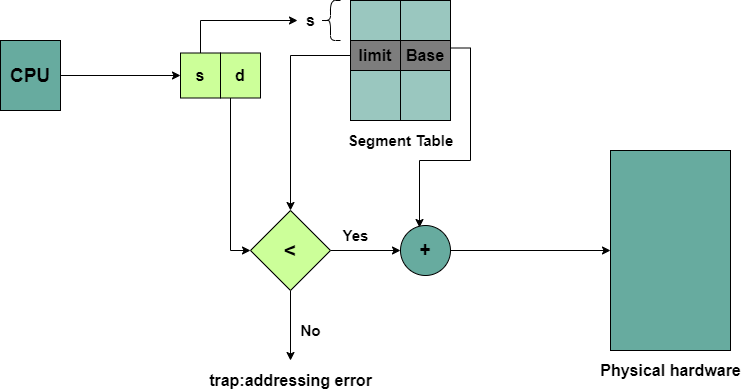
**Segment Table Base Register(STBR)** The STBR register is used to point the segment table's location in the memory.

**Segment Table Length Register(STLR)** This register indicates the number of segments used by a program. The segment number s is legal if **s<STLR**

****

Segmentation Hardware

Given below figure shows the segmentation hardware :



The logical address generated by CPU consist of two parts:

Segment Number(s): It is used as an index into the segment table.

Offset(d): It must lie in between '0' and 'segment limit'.In this case, if the Offset exceeds the segment limit then the trap is generated.

Thus;**correct offset+segment base= address in Physical memory**

and segment table is basically an array of base-limit register pair.

Advantages of Segmentation

The Advantages of the Segmentation technique are as follows:

* In the Segmentation technique, the segment table is mainly used to keep the record of segments. Also, the segment table occupies less space as compared to the paging table.
* There is no Internal Fragmentation.
* Segmentation generally allows us to divide the program into modules that provide better visualization.
* Segments are of variable size.

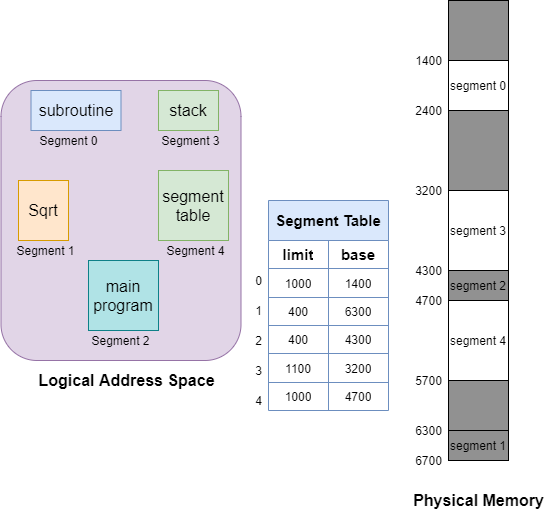
Disadvantages of Segmentation

Some disadvantages of this technique are as follows:

* Maintaining a segment table for each process leads to overhead
* This technique is expensive.
* The time is taken in order to fetch the instruction increases since now two memory accesses are required.
* Segments are of unequal size in segmentation and thus are not suitable for swapping.
* This technique leads to external fragmentation as the free space gets broken down into smaller pieces along with the processes being loaded and removed from the main memory then this will result in a lot of memory waste.

Example of Segmentation

Given below is the example of the segmentation, There are five segments numbered from 0 to 4. These segments will be stored in Physical memory as shown. There is a separate entry for each segment in the segment table which contains the beginning entry address of the segment in the physical memory( denoted as the base) and also contains the length of the segment(denoted as limit).



Segment 2 is 400 bytes long and begins at location 4300. Thus in this case a reference to byte 53 of segment 2 is mapped onto the location 4300 (4300+53=4353). A reference to segment 3, byte 85 is mapped to 3200(the base of segment 3)+852=4052.

A reference to byte 1222 of segment 0 would result in the trap to the OS, as the length of this segment is 1000 bytes.

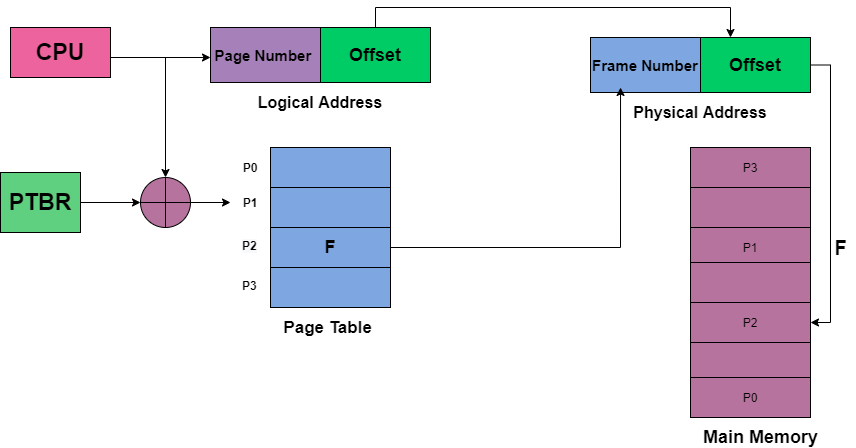
# Difference between Paging and Segmentation

In this tutorial we will be covering the difference between Paging and Segmentation these are two different Memory management techniques in Operating systems.

## Paging in OS

**Paging** is basically a technique of memory management in the operating system. In this technique, the operating system retrieves the processes from the secondary memory into the main memory and this memory is in the form of pages. Paging is a logical concept.

With the help of this technique, the main memory is split into the small blocks of physical memory that are commonly known as frames. In paging size of frames is fixed. In order to prevent external fragmentation and for the maximum usage of the main memory, the frame size must be the same as the page size. This technique helps to access the data faster.

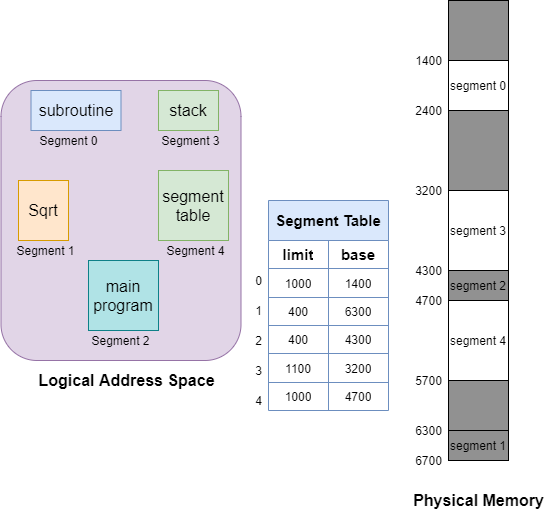


## Segmentation in OS

**Segmentation** is another technique of memory management. This technique is similar to Paging except for the fact that in segmentation the segments of a process are of variable length but in Paging the pages are of fixed size.

The memory is split into variable-length parts in segmentation. Each part is commonly known as segments. Information related to each segment is stored in a table and this table is commonly known as**the segment table.**The **segment table** generally occupies less space as compared to the **paging table.**

| **Paging** | **Segmentation** |
| --- | --- |
| Paging is a memory management technique where memory is partitioned into fixed-sized blocks that are commonly known as **pages.** | Segmentation is also a memory management technique where memory is partitioned into variable-sized blocks that are commonly known as **segments**. |
| With the help of Paging, the logical address is divided into a **page number** and **page offset**. | With the help of Segmentation, the logical address is divided into **section number** and **section offset**. |
| This technique may lead to **Internal Fragmentation**. | Segmentation may lead to **External Fragmentation**. |
| In Paging, the page size is decided by the hardware. | While in Segmentation, the size of the segment is decided by the user. |
| In order to maintain the page data, the page table is created in the Paging | In order to maintain the segment data, the segment table is created in the Paging |
| The page table mainly contains the base address of each page. | The segment table mainly contains the segment number and the offset. |
| This technique is faster than segmentation. | On the other hand, segmentation is slower than paging. |
| In Paging, a list of free frames is maintained by the Operating system. | In Segmentation, a list of holes is maintained by the Operating system. |
| In this technique, in order to calculate the absolute address page number and the offset both are required. | In this technique, in order to calculate the absolute address segment number and the offset both are required. |

****

Difference between Contiguous and Non-Contiguous Memory Allocation

In this tutorial, we will be covering the differences between Contiguous and NonContiguous Memory Allocation.

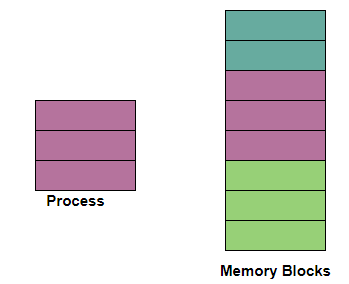
In the Operating System, there are two techniques for memory allocation and these are as follows:

* Contiguous Memory Allocation
* Non-Contiguous Memory Allocation

Contiguous Memory Allocation

In **Contiguous Memory Allocation** whenever any user process request for the memory then a single section of the contiguous memory block is allocated to that process according to the requirements of the process. Contiguous memory allocation is achieved just by **dividing the memory** into the **fixed-sized partition**.

In this, all the available memory space **remains together at one place** and freely available **memory partitions are not distributed** here and there across the **whole memory space.**

****

Non-Contiguous Memory Allocation

With the help of Non-contiguous memory allocation, a process is allowed to**acquire several memory blocks at different locations in the memory** according to its need. In the **non-contiguous memory allocation,** the available free memory space is distributed here and there which means that all the free memory space is not in one place.

In this technique, memory space acquired by a process is not at one place but it is at different locations according to the requirements of the process.

| **Contiguous Memory Allocation** | **Non-Contiguous Memory Allocation** |
| --- | --- |
| The contiguous Memory Allocation technique allocates **one single contiguous block of memory** to the process and memory is allocated to the process in a continuous fashion. | The non-Contiguous Memory allocation technique divides the **process into several blocks** and then places them in the **different address space of the memory** that is memory is allocated to the process in a non-contiguous fashion. |
| In this Allocation scheme, there is **no overhead** in the address translation while the execution of the process. | While in this scheme, there is **overhead** in the address translation while the execution of the process. |
| In Contiguous Memory Allocation, the process **executes faster** because the whole process is in a sequential block. | In Non-contiguous Memory allocation **execution of the process is slow**as the process is in different locations of the memory. |
| Contiguous Memory Allocation**is easier for the Operating System** to control. | The non-Contiguous Memory Allocation scheme**is difficult for the Operating System** to control. |
| In this, the memory space is divided into **fixed-sized partitions** and each partition is allocated only to a single process. | In this scheme, the process is divided into several blocks and then these blocks are placed in different parts of the memory according to the availability of memory space. |
| Contiguous memory allocation includes **single partition allocation** and **multi-partition allocation.** | Non-Contiguous memory allocation includes **Paging and Segmentation**. |
| In this type of **memory allocation**, generally, a table is maintained by the operating system that maintains the list of all **available and occupied partitions** in the memory space. | In this type of memory allocation **generally, a table has to be maintained for each process**that mainly carries the **base addresses of each block** that has been acquired by a process in the memory. |
| There is wastage of memory in Contiguous Memory allocation. | There is no wastage of memory in Non-Contiguous Memory allocation. |
| In this type of allocation,swapped-in processes are arranged in the originally allocated space. | In this type of allocation,swapped-in processes can be arranged in any place in the memory. |

Difference between Internal and External Fragmentation

In this tutorial, we will be covering the difference between Internal and External Fragmentation in Operating System.

Let us first understand what is fragmentation and the causes of fragmentation.

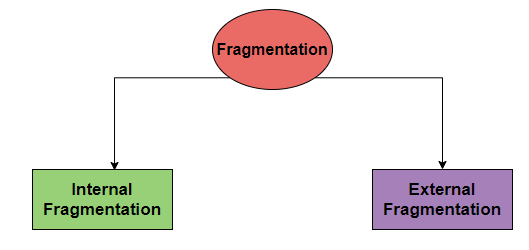
Fragmentation in OS

In the storage system of the computer, as processes are loaded space is allocated to them and this space is divided on the basis of **fixed partitioning**(where the size of partitions is fixed before any process gets loaded into the memory) and **dynamic partitioning**(where the size of partitions is decided at the run time and it is mainly according to the size of the process).

Likewise, processes are loaded and then removed from the memory due to which the free space in the memory gets broken down into smaller pieces. This mainly creates small non utilized fragmented memory spaces that are so small due to which normal processes can not fit into them.

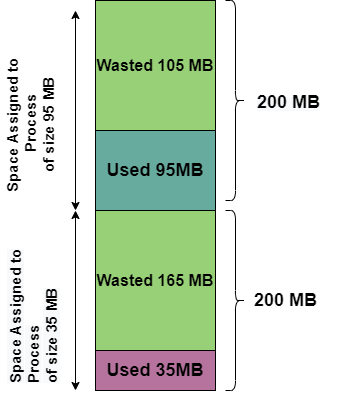
The problem due to which memory space is not getting utilized at all is commonly known as **Fragmentation** in the operating system.

Fragmentation is further divided into two types:



Internal Fragmentation in OS

Internal Fragmentation is a problem that occurs when the process is allocated to a memory block whose size is more than the size of that process and due to which some part of the memory is left unused. Thus the space wasted inside the allocated memory block is due to the restriction on the allowed sizes of allocated blocks.

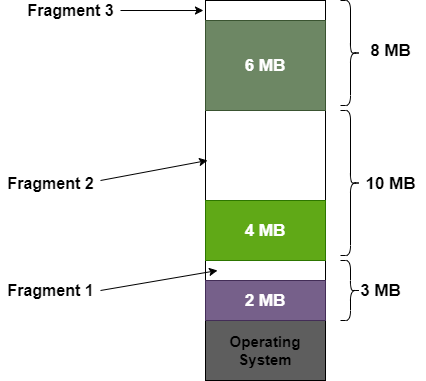


**The above figure shows the Internal Fragmentation in Operating System**

In the above diagram, the difference between memory allocated and required space or memory by the process is called **internal fragmentation**.

External Fragmentation in OS

When the memory space in the system can easily satisfy the requirement of the processes, but this available memory space is **non-contiguous**, So it can’t be utilized further. Then this problem is referred to as **External Fragmentation.**

****

**The above Figure shows the External Fragmentation in Operating System**

Suppose, we want to allocate memory to a process of size 8 MB and as In the above diagram, we can see that, there is enough space that is 9 MB to run a process of size 8MB but the memory that is fragments are not contiguous.

There is a solution for the problem of External Fragmentation and it is **Compaction**. The main goal is to shuffle the memory contents so as to place all the free memory together in one large block.

Also, Compaction is not always possible. Suppose if relocation is static and is done at the load time then in that case compaction cannot be done. Because compaction is only possible if the relocation is dynamic and is done at the execution time.

Difference Between **Internal**and **External Fragmentation**

Now let us cover the major difference between **Internal Fragmentation** and **External Fragmentation**.

| **Internal Fragmentation** | **External Fragmentation** |
| --- | --- |
| Internal Fragmentation occurs when the memory blocks of fixed-size are allocated to the processes. | External Fragmentation occurs when the memory blocks of variable-size are allocated to the processes dynamically. |
| This type of fragmentation mainly occurs when the fixed size partition is assigned to a process whose size is less than the size of the partition due to which the rest of the space in the partition becomes unusable. | When the memory space in the system can easily satisfy the requirement of the processes, but this available memory space is **non-contiguous**, So it can’t be utilized further. |
| The difference between allocated memory and memory required by a process is called Internal fragmentation. | Unused memory spaces between the non-contiguous memory fragments that are too small to serve a new process are called External fragmentation. |
| It mainly refers to the unused space in the partition that resides in the allocated region; as suggested by the name. | It mainly refers to the unused blocks of the memory that are not contiguous and hence are unable to satisfy the requirements of the process. |
| Best-fit block can be used to overcome the problem of Internal fragmentation. | Compaction, segmentation, and paging can be used to overcome the problem of External fragmentation. |
| Paging suffers from Internal fragmentation. | First-fit and best-fit suffer from external fragmentation. |

# Virtual Memory in Operating Systems

In this tutorial, we will be covering the concept of Virtual Memory in an Operating System.

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as a large virtual memory is provided for user programs when a very small physical memory is there. Thus Virtual memory is a technique that allows the execution of processes that are not in the physical memory completely.

Virtual Memory mainly gives the illusion of more physical memory than there really is with the help of Demand Paging.

In real scenarios, most processes never need all their pages at once, for the following reasons :

* Error handling code is not needed unless that specific error occurs, some of which are quite rare.
* Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
* Certain features of certain programs are rarely used.

In an Operating system, the memory is usually stored in the form of units that are known as **pages**. Basically, these are atomic units used to store large programs.

Virtual memory can be implemented with the help of:-

1. Demand Paging
2. Demand Segmentation

## The Need for Virtual Memory

Following are the reasons due to which there is a need for Virtual Memory:

* In case, if a computer running the Windows operating system needs more memory or RAM than the memory installed in the system then it uses a small portion of the hard drive for this purpose.
* Suppose there is a situation when your computer does not have space in the physical memory, then it writes things that it needs to remember into the hard disk in a swap file and that as virtual memory.

## Benefits of having Virtual Memory

1. Large programs can be written, as the virtual space available is huge compared to physical memory.
2. Less I/O required leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very little space on actual physical memory.
4. Therefore, the Logical address space can be much larger than that of physical address space.
5. Virtual memory allows address spaces to be shared by several processes.
6. During the process of creation, virtual memory allows: **copy-on-write** and **Memory-mapped files**

## Execution of the Program in the Virtual memory

With the help of the Operating system few pieces of the program are brought into the main memory:

* A portion of the process that is brought in the main memory is known as Resident Set.

Whenever an address is needed that is not in the main memory, then it generates an interrupt. The process is placed in the blocked state by the Operating system. Those pieces of the process that contains the logical address are brought into the main memory.

## Demand Paging

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On-demand). Initially, only those pages are loaded which will be required by the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry, the rest of the table is empty. In the case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped-out page.

## Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more processes into memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

## Thrashing

A process that is spending more time paging than executing is said to be thrashing. In other words, it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

Initially, when the CPU utilization is low, the process scheduling mechanism, to increase the level of multiprogramming loads multiple processes into the memory at the same time, allocating a limited amount of frames to each process. As the memory fills up, the process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the processes are waiting for pages. Hence the scheduler loads more processes to increase CPU utilization, as this continues at a point in time the complete system comes to a stop.

## Advantages of Virtual Memory

Given below are the advantages of using Virtual Memory:

* Virtual Memory allows you to run more applications at a time.
* With the help of virtual memory, you can easily fit many large programs into smaller programs.
* With the help of Virtual memory, a multiprogramming environment can be easily implemented.
* As more processes should be maintained in the main memory which leads to the effective utilization of the CPU.
* Data should be read from the disk at the time when required.
* Common data can be shared easily between memory.
* With the help of virtual memory, speed is gained when only a particular segment of the program is required for the execution of the program.
* The process may even become larger than all of the physical memory.

## Disadvantages of Virtual Memory

Given below are the drawbacks of using Virtual Memory:

* Virtual memory reduces the stability of the system.
* The performance of Virtual memory is not as good as that of RAM.
* If a system is using virtual memory then applications may run slower.
* Virtual memory negatively affects the overall performance of a system.
* Virtual memory occupies the storage space, which might be otherwise used for long-term data storage.
* This memory takes more time in order to switch between applications.

# Demand Paging in Operating Systems

In this tutorial, we will cover the concept of Demand Paging in the Operating System.

Every process in the virtual memory contains lots of pages and in some cases, it might not be efficient to swap all the pages for the process at once. Because it might be possible that the program may need only a certain page for the application to run. Let us take an example here, suppose there is a 500 MB application and it may need as little as 100MB pages to be swapped, so in this case, there is no need to swap all pages at once.

The demand paging system is somehow similar to the paging system with swapping where processes mainly reside in the main memory(usually in the hard disk). Thus demand paging is the process that solves the above problem only by swapping the pages on Demand. This is also known as **lazy swapper**( It never swaps the page into the memory unless it is needed).

Swapper that deals with the individual pages of a process are referred to as **Pager.**

Demand Paging is a technique in which a page is usually brought into the main memory only when it is needed or demanded by the CPU. Initially, only those pages are loaded that are required by the process immediately. Those pages that are never accessed are thus never loaded into the physical memory.

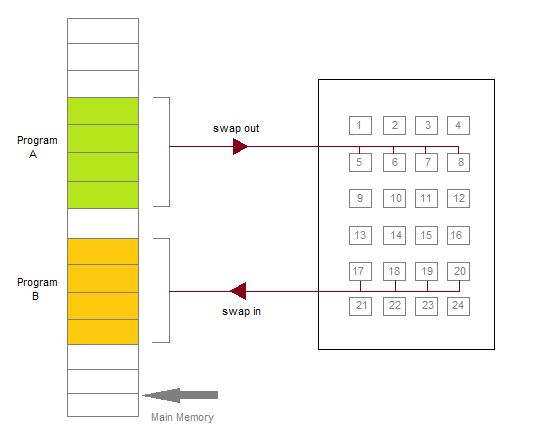


Figure: Transfer of a Paged Memory to the contiguous disk space.

Whenever a page is needed? make a reference to it;

* If the reference is **invalid** then abort it.
* If the page is Not-in-memory then bring it to memory.

## Valid-Invalid Bit

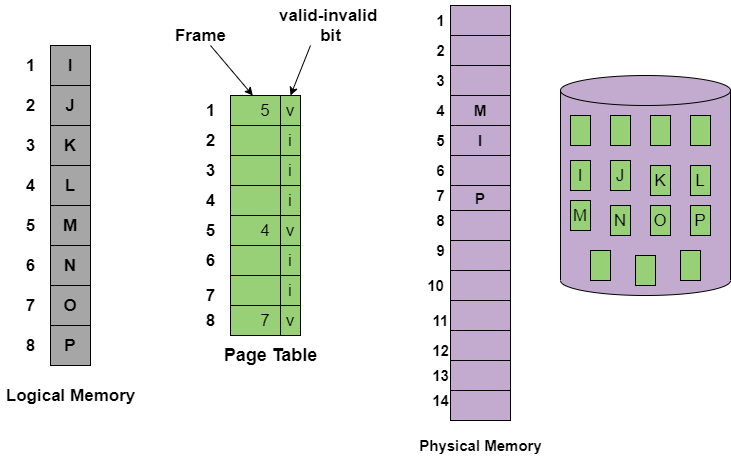
Some form of hardware support is used to distinguish between the pages that are in the memory and the pages that are on the disk. Thus for this purpose Valid-Invalid scheme is used:

* With each page table entry, a valid-invalid bit is associated( where **1** indicates**in the memory** and **0** indicates **not in the memory**)
* Initially, the valid-invalid bit is set to 0 for all table entries.

1. If the bit is set to "**valid**", then the associated page is **both legal and is in memory.**
2. If the bit is set to "**invalid**" then it indicates that the **page is either not valid** or the page is valid **but is currently not on the disk**.

* For the **pages** that are **brought into the memory**, **the page table is set as usual.**
* But for the**pages** that are **not currently in the memory**, the **page table** is either**simply marked as invalid**or it contains the **address of the page on the disk**.

During the translation of address, if the valid-invalid bit in the page table entry is 0 then it leads to **page fault.**



The above figure is to indicates the page table when some pages are not in the main memory.

## How Demand Paging Works?

First of all the components that are involved in the Demand paging process are as follows:

* Main Memory
* CPU
* Secondary Memory
* Interrupt
* Physical Address space
* Logical Address space
* Operating System
* Page Table

1. If a page is not available in the main memory in its active state; then a request may be made to the CPU for that page. Thus for this purpose, it has to generate an interrupt.
2. After that, the Operating system moves the process to the blocked state as an interrupt has occurred.
3. Then after this, the Operating system searches the given page in the Logical address space.
4. And Finally with the help of the page replacement algorithms, replacements are made in the physical address space. Page tables are updated simultaneously.
5. After that, the CPU is informed about that update and then asked to go ahead with the execution and the process gets back into its ready state.

When the process requires any of the pages that are not loaded into the memory, a page fault trap is triggered and the following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If it is found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from the disk to the specified memory location. ( This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime. )
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

## Advantages of Demand Paging

The benefits of using the Demand Paging technique are as follows:

* With the help of Demand Paging, memory is utilized efficiently.
* Demand paging avoids External Fragmentation.
* Less Input/Output is needed for Demand Paging.
* This process is not constrained by the size of physical memory.
* With Demand Paging it becomes easier to share the pages.
* With this technique, portions of the process that are never called are never loaded.
* No compaction is required in demand Paging.

## Disadvantages of Demand paging

Drawbacks of Demand Paging are as follows:

* There is an increase in overheads due to interrupts and page tables.
* Memory access time in demand paging is longer.

## Pure Demand Paging

In some cases when initially no pages are loaded into the memory, pages in such cases are only loaded when are demanded by the process by generating page faults. It is then referred to as **Pure Demand Paging**.

* In the case of pure demand paging, there is not even a single page that is loaded into the memory initially. Thus pure demand paging causes the page fault.
* When the execution of the process starts with no pages in the memory, then the operating system sets the instruction pointer to the first instruction of the process and that is on a non-memory resident page and then in this case the process immediately faults for the page.
* After that when this page is brought into the memory then the process continues its execution, page fault is necessary until every page that it needs is in the memory.
* And at this point, it can execute with no more faults.
* This scheme is referred to as Pure Demand Paging: means never bring a page into the memory until it is required.

# Page Fault in Operating System

In this tutorial, we will be covering the concept of page fault and how to handle page fault in the Operating System.

## Page Fault in OS

Page fault dominates more like an error. It mainly occurs when any program tries to access the data or the code that is in the address space of the program, but that data is not currently located in the RAM of the system.

* So basically when the page referenced by the CPU is not found in the main memory then the situation is termed as Page Fault.
* Whenever any page fault occurs, then the required page has to be fetched from the secondary memory into the main memory.

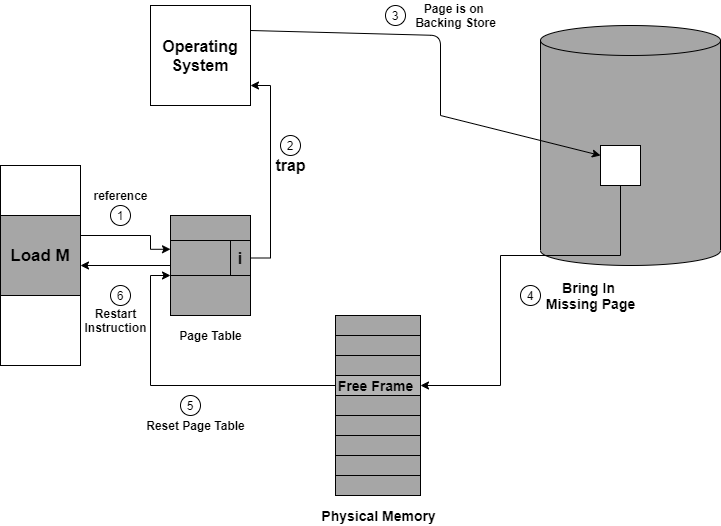
In case if the required page is not loaded into the memory, then a page fault trap arises

The page fault mainly generates an exception, which is used to notify the operating system that it must have to retrieve the "pages" from the virtual memory in order to continue the execution. Once all the data is moved into the physical memory the program continues its execution normally. The Page fault process takes place in the background and thus goes unnoticed by the user.

* The hardware of the computer tracks to the kernel and the program counter (PC) is generally saved on the stack.CPU registers store the information of the current state of instruction.
* An assembly program is started that usually saves the general registers and also saves the other volatile information to prevent the OS from destroying it.

## Handling the Page Fault

Given below is the simple procedure to handle the page fault:



**Figure: Steps to Handle the Page fault**

If you will access a page that is marked as invalid then it also causes a Page Fault. Then the Paging hardware during translating the address through the page table will notice that the invalid bit is set that will cause a trap to the Operating system.

This trap is mainly the result of the failure of the Operating system in order to bring the desired page into memory.

Let us understand the procedure to handle the page fault as shown with the help of the above diagram:

1. First of all, internal table(that is usually the process control block) for this process in order to determine whether the reference was valid or invalid memory access.
2. If the reference is invalid, then we will terminate the process. If the reference is valid, but we have not bought in that page so now we just page it in.
3. Then we **locate the free frame lis**t in order to find the free frame.
4. Now a disk operation is scheduled in order to read the **desired page into the newly allocated fram**e.
5. When the disk is completely read, then the **internal table is modified** that is kept with the process, and the page table that mainly indicates the page is now in memory.
6. Now we will restart the instruction that was interrupted due to the trap. Now the process can access the page as though it had always been in memory.

# Page Replacement Algorithms in Operating System

In this tutorial, we will be covering the concept of Page replacement and its algorithms in the Operating system.

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more processes into memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used.

In this case, if a process requests a new page and supposes there are no free frames, then the Operating system needs to decide which page to replace. The operating system must use any page replacement algorithm in order to select the victim frame. The Operating system must then write the victim frame to the disk then read the desired page into the frame and then update the page tables. And all these require double the disk access time.

* Page replacement prevents the over-allocation of the memory by modifying the page-fault service routine.
* To reduce the overhead of page replacement a **modify bit (dirty bit)** is used in order to indicate whether each page is modified.
* This technique provides complete separation between logical memory and physical memory.

## Page Replacement in OS

In Virtual Memory Management, Page Replacement Algorithms play an important role. The main objective of all the **Page replacement policies** is to decrease the maximum number of **page faults**.

**Page Fault** – It is basically a memory error, and it occurs when the current programs attempt to access the memory page for mapping into virtual address space, but it is unable to load into the physical memory then this is referred to as Page fault.

## Basic Page Replacement Algorithm in OS

Page Replacement technique uses the following approach. If there is no free frame, then we will find the one that is not currently being used and then free it. A-frame can be freed by writing its content to swap space and then change the page table in order to indicate that the page is no longer in the memory.

1. First of all, find the location of the desired page on the disk.
2. Find a free Frame: a) If there is a free frame, then use it. b) If there is no free frame then make use of the page-replacement algorithm in order to select the victim frame. c) Then after that write the victim frame to the disk and then make the changes in the page table and frame table accordingly.
3. After that read the desired page into the newly freed frame and then change the page and frame tables.
4. Restart the process.

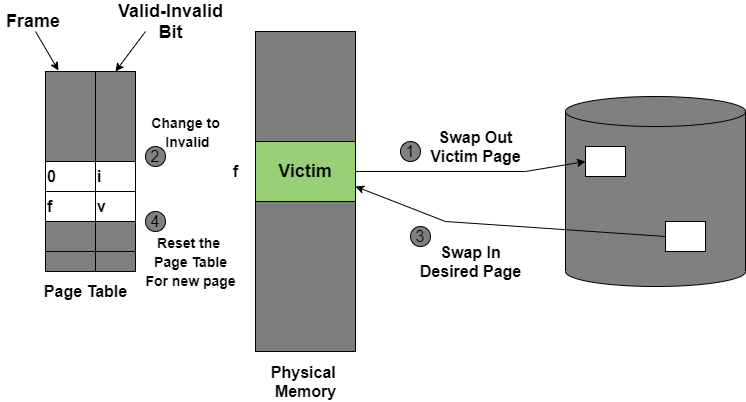
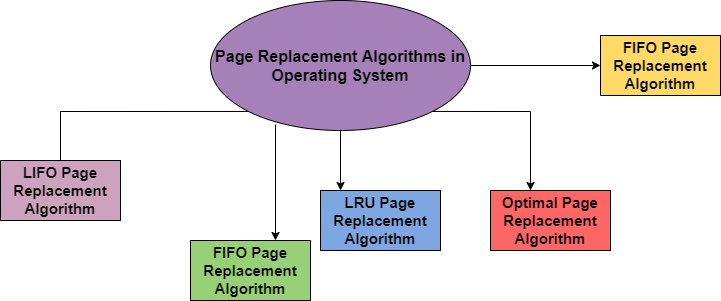


Figure: Page Replacement

## Page Replacement Algorithms in OS

This algorithm helps to decide which pages must be swapped out from the main memory in order to create a room for the incoming page. This Algorithm wants the lowest page-fault rate.

Various Page Replacement algorithms used in the Operating system are as follows;



Let us discuss all algorithms one by one in the upcoming sections:

### 1. FIFO Page Replacement Algorithm

It is a very simple way of Page replacement and is referred to as First in First Out. This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.

* This algorithm is implemented by keeping the track of all the pages in the queue.
* As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.
* This is not an effective way of page replacement but it can be used for small systems.

### Advantages

* This algorithm is simple and easy to use.
* FIFO does not cause more overhead.

### Disadvantages

* This algorithm does not make the use of the frequency of **last used time rather**it just replaces the Oldest Page.
* There is an increase in **page faults** as page frames increases.
* The performance of this algorithm is the worst.

### 2. LIFO Page Replacement Algorithm

This Page Replacement algorithm stands for "Last In First Out".This algorithm works in a similar way to the LIFO principle.

* In this, the newest page is replaced which is arrived at last in the primary memory
* This algorithm makes use of the stack for monitoring all the pages.

### 3. LRU Page Replacement Algorithm in OS

This algorithm stands for "Least recent used" and this algorithm helps the Operating system to search those pages that are used over a short duration of time frame.

* The page that has not been used for the longest time in the main memory will be selected for replacement.
* This algorithm is easy to implement.
* This algorithm makes use of the counter along with the even-page.

### **Advantages of LRU**

* It is an efficient technique.
* With this algorithm, it becomes easy to identify the faulty pages that are not needed for a long time.
* It helps in Full analysis.

### **Disadvantages of LRU**

* It is expensive and has more complexity.
* There is a need for an additional data structure.

### 4. Optimal Page Replacement Algorithm

This algorithm mainly replaces the page that will not be used for the longest time in the future. The practical implementation of this algorithm is not possible.

* Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
* This algorithm leads to less number of page faults and thus is the best-known algorithm

Also, this algorithm can be used to measure the performance of other algorithms.

### **Advantages of OPR**

* This algorithm is easy to use.
* This algorithm provides excellent efficiency and is less complex.
* For the best result, the implementation of data structures is very easy

### **Disadvantages of OPR**

* In this algorithm future awareness of the program is needed.
* Practical Implementation is not possible because the operating system is unable to track the future request

### 5. Random Page Replacement Algorithm

As indicated from the name this algorithm replaces the page randomly. This Algorithm can work like any other page replacement algorithm that is LIFO, FIFO, Optimal, and LRU.

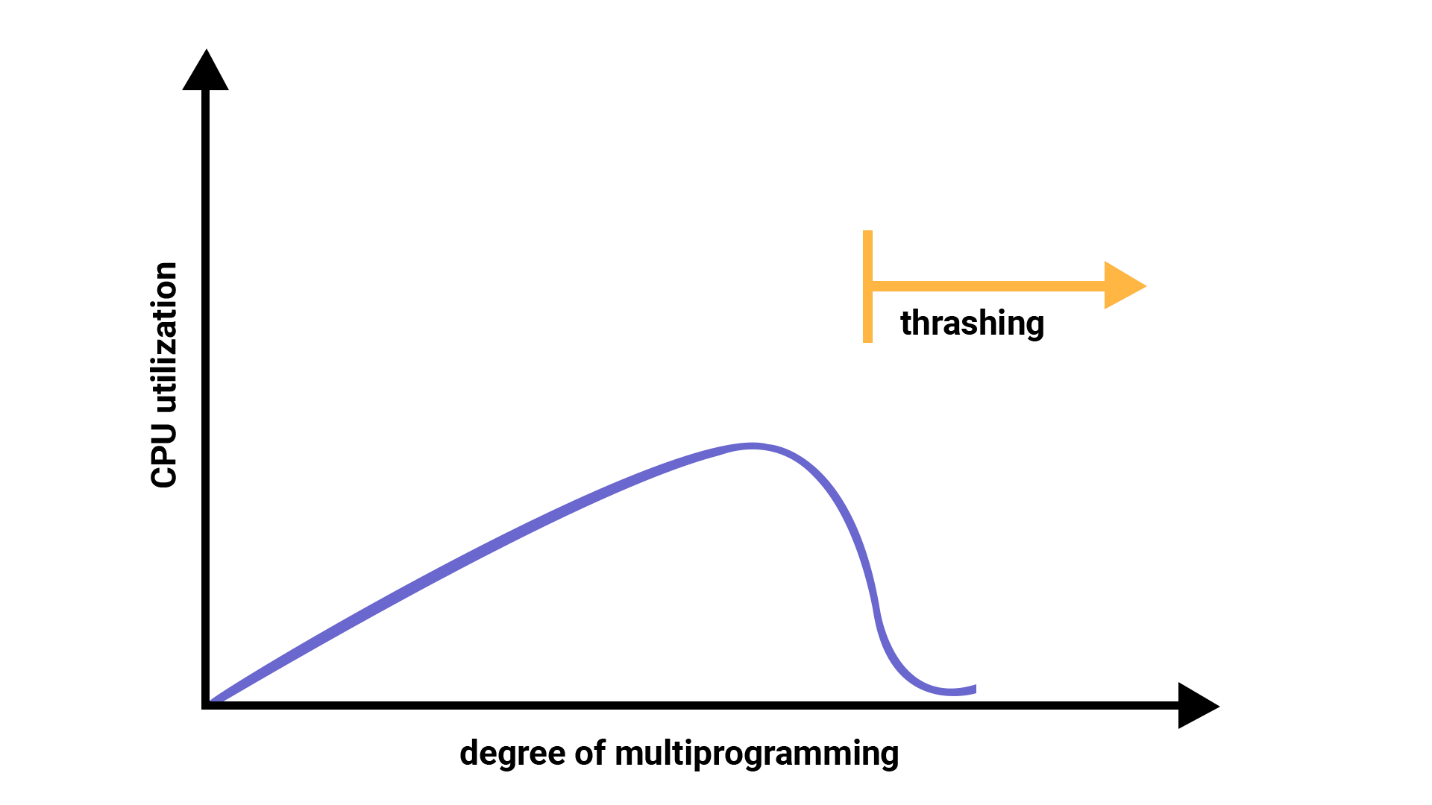
# Thrashing in Operating System

In this tutorial, we will be covering the concept of thrashing in the Operating system.

In case, if the page fault and swapping happens very frequently at a higher rate, then the operating system has to spend more time swapping these pages. This state in the operating system is termed thrashing. Because of thrashing the CPU utilization is going to be reduced.

Let's understand by an example, if any process does not have the number of frames that it needs to support pages in active use then it will quickly page fault. And at this point, the process must replace some pages. As all the pages of the process are actively in use, it must replace a page that will be needed again right away. Consequently, the process will quickly fault again, and again, and again, replacing pages that it must bring back in immediately. This high paging activity by a process is called thrashing.

During thrashing, the CPU spends less time on some actual productive work spend more time swapping.



**Figure: Thrashing**

## Causes of Thrashing

Thrashing affects the performance of execution in the Operating system. Also, thrashing results in severe performance problems in the Operating system.

When the utilization of CPU is low, then the process scheduling mechanism tries to load many processes into the memory at the same time due to which degree of Multiprogramming can be increased. Now in this situation, there are more processes in the memory as compared to the available number of frames in the memory. Allocation of the limited amount of frames to each process.

Whenever any process with high priority arrives in the memory and if the frame is not freely available at that time then the other process that has occupied the frame is residing in the frame will move to secondary storage and after that this free frame will be allocated to higher priority process.

We can also say that as soon as the memory fills up, the process starts spending a lot of time for the required pages to be swapped in. Again the utilization of the CPU becomes low because most of the processes are waiting for pages.

Thus a high degree of multiprogramming and lack of frames are two main causes of thrashing in the Operating system.

## Effect of Thrashing

At the time, when thrashing starts then the operating system tries to apply either the **Global page replacement**Algorithm or the **Local page replacement**algorithm.

### Global Page Replacement

The Global Page replacement has access to bring any page, whenever thrashing found it tries to bring more pages. Actually, due to this, no process can get enough frames and as a result, the thrashing will increase more and more. Thus the global page replacement algorithm is not suitable whenever thrashing happens.

### Local Page Replacement

Unlike the Global Page replacement, the local page replacement will select pages which only belongs to that process. Due to this, there is a chance of a reduction in the thrashing. As it is also proved that there are many disadvantages of Local Page replacement. Thus local page replacement is simply an alternative to Global Page replacement.

## Techniques used to handle the thrashing

As we have already told you the Local Page replacement is better than the Global Page replacement but local page replacement has many disadvantages too, so it is not suggestible. Thus given below are some other techniques that are used:

### Working-Set Model

This model is based on the assumption of the locality. It makes the use of the parameter ? in order to define the working-set window. The main idea is to examine the most recent? page reference. What locality is saying, the recently used page can be used again, and also the pages that are nearby this page will also be used?

### 1. Working Set

The set of the pages in the most recent? page reference is known as the working set. If a page is in active use, then it will be in the working set. In case if the page is no longer being used then it will drop from the working set ? times after its last reference.

The working set mainly gives the approximation of the locality of the program.

The accuracy of the working set mainly depends on? what is chosen?

This working set model avoids thrashing while keeping the degree of multiprogramming as high as possible.

### 2. Page Fault Frequency

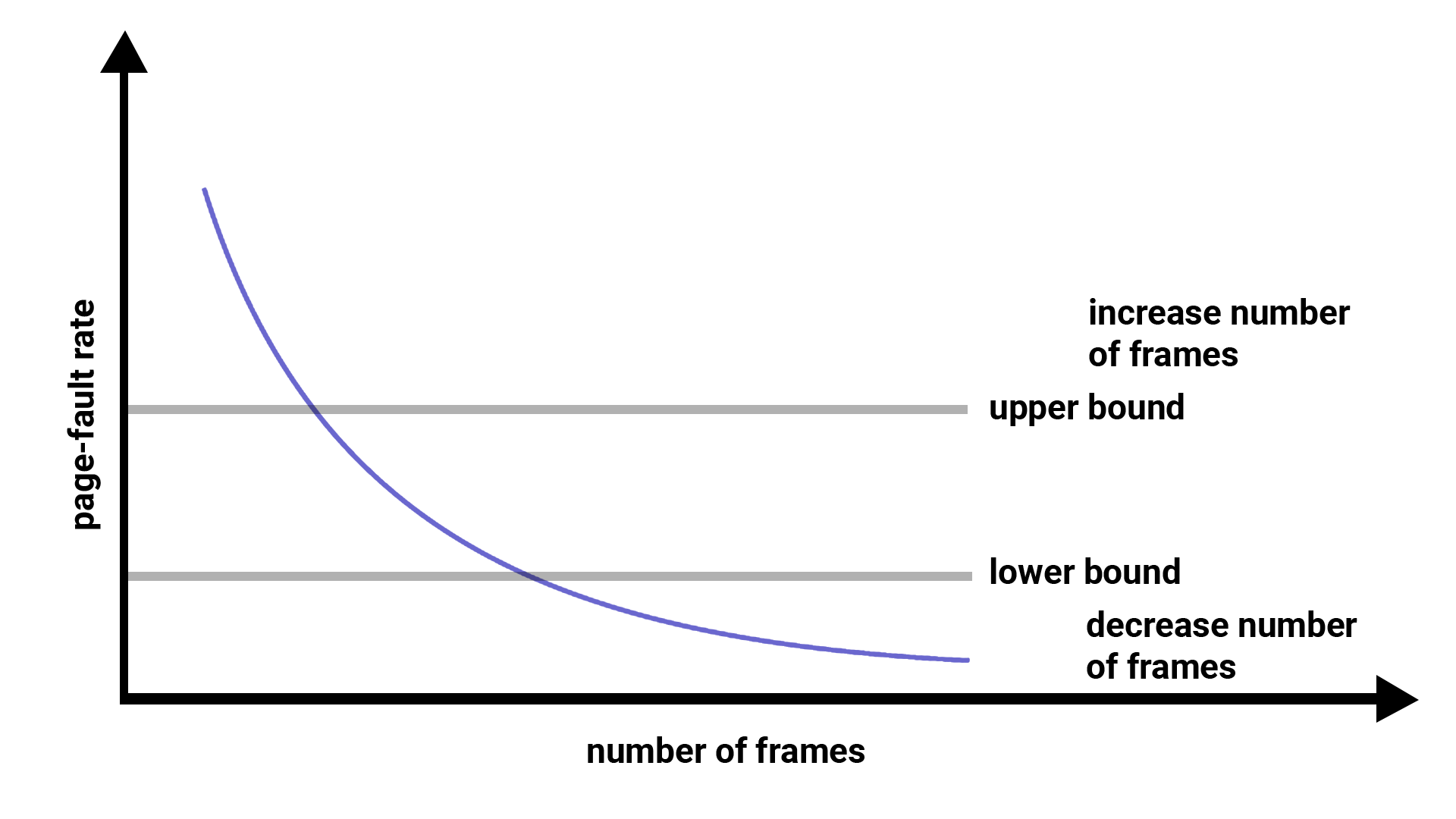


Figure: Page fault frequency

The working-set model is successful and its knowledge can be useful in preparing but it is a very clumpy approach in order to avoid thrashing. There is another technique that is used to avoid thrashing and it is Page Fault Frequency(PFF) and it is a more direct approach.

The main problem is how to prevent thrashing. As thrashing has a high page fault rate and also we want to control the page fault rate.

When the Page fault is too high, then we know that the process needs more frames. Conversely, if the page fault-rate is too low then the process may have too many frames.

We can establish upper and lower bounds on the desired page faults. If the actual page-fault rate exceeds the upper limit then we will allocate the process to another frame. And if the page fault rate falls below the lower limit then we can remove the frame from the process.

Thus with this, we can directly measure and control the page fault rate in order to prevent thrashing.

## **File**

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

## **File Structure**

A File Structure should be according to a required format that the operating system can understand.

* A file has a certain defined structure according to its type.
* A text file is a sequence of characters organized into lines.
* A source file is a sequence of procedures and functions.
* An object file is a sequence of bytes organized into blocks that are understandable by the machine.
* When an operating system defines different file structures, it also contains the code to support these file structures. Unix, MS-DOS support a minimum number of file structures.

## **File Type**

File type refers to the ability of the operating system to distinguish different types of file such as text files, source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

### Ordinary files

* These are the files that contain user information.
* These may have text, databases or executable programs.
* The user can apply various operations on such files like add, modify, delete or even remove the entire file.

### Directory files

* These files contain a list of file names and other information related to these files.

### Special files

* These files are also known as device files.
* These files represent physical devices like disks, terminals, printers, networks, tape drive etc.

**These files are of two types −**

* Character special files − data is handled character by character as in case of terminals or printers.
* Block special files − data is handled in blocks as in the case of disks and tapes.

## **File Access Mechanisms**

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

* Sequential accessFI
* Direct/Random access
* Indexed sequential access

### Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

### Direct/Random access

* Random access file organization provides, accessing the records directly.
* Each record has its own address on the file with the help of which it can be directly accessed for reading or writing.
* The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

### Indexed sequential access

* This mechanism is built up on the basis of sequential access.
* An index is created for each file which contains pointers to various blocks.
* Index is searched sequentially and its pointer is used to access the file directly.

# **Operations on the File**

There are various operations which can be implemented on a file. We will see all of them in detail.

**1.Create**

Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.

**2.Write**

Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which the data needs to be written.

**3.Read**HTML Tutorial

Every file is opened in three different modes : Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

**4.Re-position or seeking.**

Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called seeking.

**5.Delete**

Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.

**6.Truncate**

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

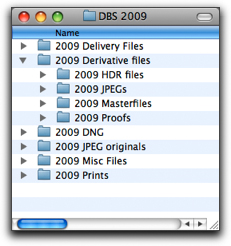
**system calls of files**

* **open** : system call to open a file
* open returns a file descriptor, an integer specifying the position of this open file in the table of open files for the current process
* **creat** : (no 'e') system call to create a file /\* no longer necessary \*/
* **close** : system call to close a file
* **read** : read data from a file opened for reading
* **write** : write data to a file opened for writing
* **lseek** : seek to a specified position in a file (used for random access when reading or writing a file)

**Directory structure**

A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner.





There are several logical structures of a directory, these are given below.

**Single-level directory –**   
The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.   
A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.



**Advantages:**

* Since it is a single directory, so its implementation is very easy.
* If the files are smaller in size, searching will become faster.
* The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

**Disadvantages:**

* There may chance of name collision because two files can not have the same name.
* Searching will become time taking if the directory is large.
* This can not group the same type of files together.

**Two-level directory –**   
As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.   
In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user. system’s *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



**Advantages:**

* We can give full path like /User-name/directory-name/.
* Different users can have the same directory as well as the file name.
* Searching of files becomes easier due to pathname and user-grouping.

**Disadvantages:**

* A user is not allowed to share files with other users.
* Still, it not very scalable, two files of the same type cannot be grouped together in the same user.

**Tree-structured directory –**   
Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.   
This generalization allows the user to create their own subdirectories and to organize their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

**Advantages:**

* Very general, since full pathname can be given.
* Very scalable, the probability of name collision is less.
* Searching becomes very easy, we can use both absolute paths as well as relative.

**Disadvantages:**

* Every file does not fit into the hierarchical model, files may be saved into multiple directories.
* We can not share files.
* It is inefficient, because accessing a file may go under multiple directories.

**Acyclic graph directory –**   
An acyclic graph is a graph with no cycle and allows us to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.   
It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.   
It is the point to note that the shared file is not the same as the copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



**Advantages:**

* We can share files.
* Searching is easy due to different-different paths.

**Disadvantages:**

* We share the files via linking, in case deleting it may create the problem,
* If the link is a soft link then after deleting the file we left with a dangling pointer.
* In the case of a hard link, to delete a file we have to delete all the references associated with it.

**General graph directory structure –**   
In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.   
The main problem with this kind of directory structure is to calculate the total size or space that has been taken by the files and directories.



**Advantages:**

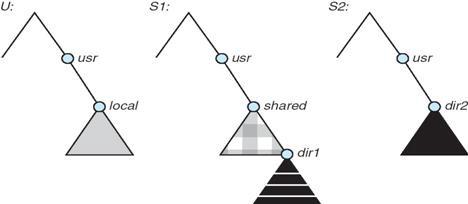
* It allows cycles.
* It is more flexible than other directories structure.

**Disadvantages:**

* It is more costly than others.
* It needs garbage collection.

# **Mounting in File System**

# Operating system organises files on a disk using a filesystem. And these file systems differ with operating systems. We have a greater number of filesystems available for linux. One of the great benefits of linux is that we can access data on many different file systems even if these file systems are from different operating systems. In order to access the file system in the linux first we need to mount it. Mounting refers to making a group of files in a file system structure accessible to a user or group of users. It is done by attaching a root directory from one file system to that of another. This ensures that the other directory or device appears as a directory or subdirectory of that system.



Mounting may be local or remote. In local mounting it connects disk drives as one machine such that they behave as a single logical system, while remote mount uses NFS(Network file system) to connect to directories on other machines so that they can be used as if they are the part of the users file system. Opposite of mounting is unmounting [commands for the same will be discussed later] and it is removal of access to those files/folders.

**File sharing**

File sharing is the practice of sharing or offering access to digital information or resources, including documents, multimedia (audio/video), graphics, computer programs, images and e-books. It is the private or public distribution of data or resources in a network with different levels of sharing privileges.

File sharing can be done using several methods. The most common techniques for file storage, distribution and transmission include the following:

* Removable storage devices
* Centralized file hosting server installations on networks
* World Wide Web-oriented hyperlinked documents
* Distributed peer-to-peer networks

File sharing is a multipurpose computer service feature that evolved from removable media via network protocols, such as File Transfer Protocol (FTP). Beginning in the 1990s, many remote file-sharing mechanisms were introduced, including FTP, hotline and Internet relay chat (IRC).

Operating systems also provide file-sharing methods, such as network file sharing (NFS). Most file-sharing tasks use two basic sets of network criteria, as follows:

* **Peer-to-Peer (P2P) File Sharing**: This is the most popular, but controversial, method of file sharing because of the use of peer-to-peer software. Network computer users locate shared data with third-party software. P2P file sharing allows users to directly access, download and edit files. Some third-party software facilitates P2P sharing by collecting and segmenting large files into smaller pieces.
* **File Hosting Services:** This P2P file-sharing alternative provides a broad selection of popular online material. These services are quite often used with Internet collaboration methods, including email, blogs, forums, or other mediums, where direct download links from the file hosting services can be included. These service websites usually host files to enable users to download them.

Once users download or make use of a file using a file-sharing network, their computer also becomes a part of that network, allowing other users to download files from the user's computer. File sharing is generally illegal, with the exception of sharing material that is not copyrighted or proprietary. Another issue with file-sharing applications is the problem of spyware or adware, as some file-sharing websites have placed spyware programs in their websites. These spyware programs are often installed on users' computers without their consent and awareness

**protection**

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc. We're going to discuss the following topics in this chapter.

* Authentication
* One Time passwords
* Program Threats
* System Threats

## **Authentication**

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

* Username / Password − User needs to enter a registered username and password with the Operating system to login into the system.
* User card/key − User needs to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
* User attribute - fingerprint/ eye retina pattern/ signature − User needs to pass his/her attribute via designated input device used by the operating system to login into the system.

## **One Time passwords**

One-time passwords provide additional security along with normal authentication. In the One-Time Password system, a unique password is required every time a user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time passwords are implemented in various ways.

* Random numbers − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to a few alphabets randomly chosen.
* Secret key − Users are provided a hardware device which can create a secret id mapped with user id. System asks for such a secret id which is to be generated every time prior to login.
* Network password − Some commercial applications send one-time passwords to users on registered mobile/ email which is required to be entered prior to login.

## **Program Threats**

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

* Trojan Horse − Such a program traps user login credentials and stores them to send to malicious user who can later on login to the computer and can access system resources.
* Trap Door − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
* Logic Bomb − Logic bomb is a situation when a program misbehaves only when certain conditions are met otherwise it works as a genuine program. It is harder to detect.
* Virus − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

## System Threats

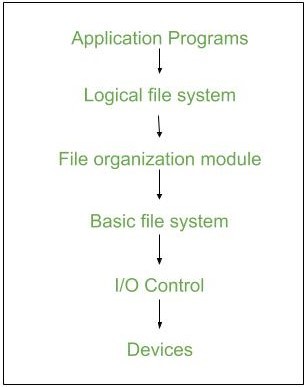
System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

* Worm − Worm is a process which can choke down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, preventing all other processes from getting required resources. Worms processes can even shut down an entire network.
* Port Scanning − Port scanning is a mechanism or means by which a hacker can detect system vulnerabilities to make an attack on the system.
* Denial of Service − Denial of service attacks normally prevents users from making legitimate use of the system. For example, a user may not be able to use the internet if denial of service attacks the browser's content settings.

**File implementation**

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

**File system organized in many layers :**

****

* **I/O Control level –**Device drivers act as interfaces between devices and Os, they help to transfer data between disk and main memory. It takes block number input and as output it gives low level hardware specific instruction.  
  /li>
* **Basic file system –**It Issues general commands to device drivers to read and write physical blocks on disk.It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.
* **File organization Module –**It has information about files, location of files and their logical and physical blocks.Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.
* **Logical file system –**It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

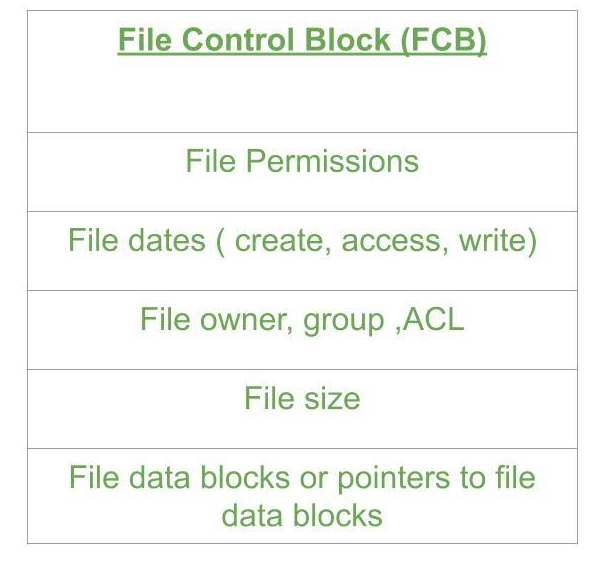
**Advantages :**

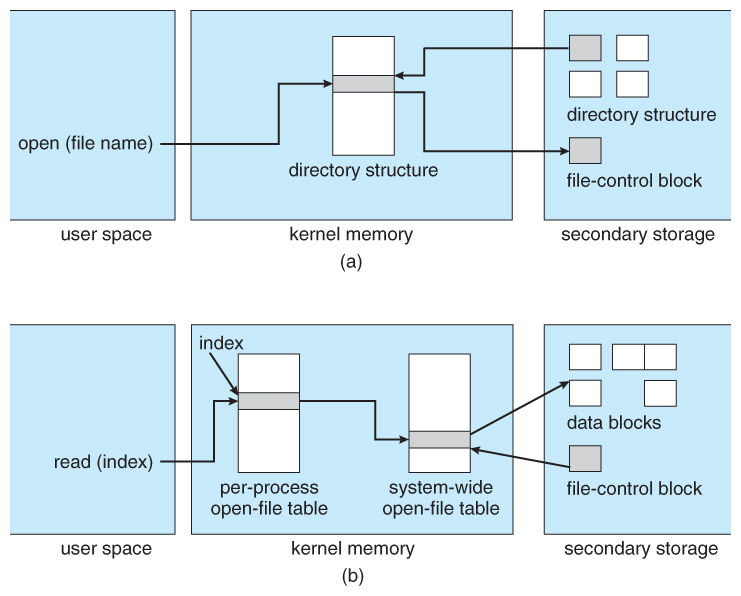
1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

**Disadvantages :** If we access many files at the same time then it results in low performance.

We can **implement** file system by using two types data structures :

1. **1. On-disk Structures –**Generally they contain information about total number of disk blocks, free disk blocks, location of them, etc. Below given are different on-disk structures :
   1. **Boot Control Block –**It is usually the first block of volume and it contains information needed to boot an operating system.In UNIX it is called boot block and in NTFS it is called partition boot sector.
   2. **Volume Control Block –**It has information about a particular partition ex:- free block count, block size and block pointers etc.In UNIX it is called super block and in NTFS it is stored in master file table.
   3. **Directory Structure –**They store file names and associated inode numbers.In UNIX, includes file names and associated file names and in NTFS, it is stored in the master file table.
   4. **Per-File FCB –**It contains details about files and it has a unique identifier number to allow association with directory entry. In NTFS it is stored in master file table.

  
**2. In-Memory Structure :**They are maintained in main-memory and these are helpful for file system management for caching. Several in-memory structures given below :

* 1. **Mount Table –**It contains information about each mounted volume.
  2. **Directory-Structure cache –**This cache holds the directory information of recently accessed directories.
  3. **System wide open-file table –**It contains the copy of FCB of each open file.
  4. **Per-process open-file table –**It contains information opened by that particular process and it maps with appropriate system wide open-file
  5. 

# **Directory Implementation**

There is the number of algorithms by which the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

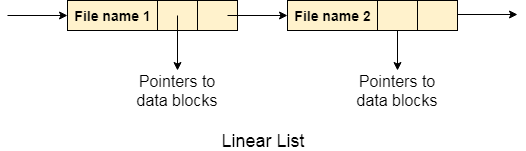
The directory implementation algorithms are classified according to the data structure they are using. There are mainly two algorithms which are used these days.

### 1. Linear List

In this algorithm, all the files in a directory are maintained as a singly linked 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 an 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.

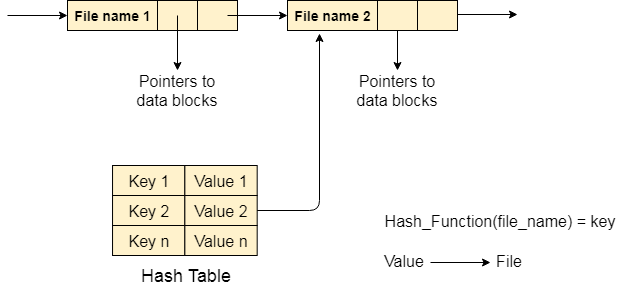


### 2. Hash Table

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.



## **Space Allocation**

## The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

## Contiguous Allocation

## Linked Allocation

## Indexed Allocation

## The main idea behind these methods is to provide:

## Efficient disk space utilization.

## Fast access to the file blocks.

## All the three methods have their own advantages and disadvantages as discussed below:

## 1. **Contiguous Allocation**

## In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

## The directory entry for a file with contiguous allocation contains

## Address of starting block

## Length of the allocated portion.

## The *file ‘mail’* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.

## pic

## Advantages:

## Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).

## This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

## Disadvantages:

## This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

## Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

## **2. Linked List Allocation**

## In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.

## The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

## *The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.*

## *linked*

## Advantages:

## This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.

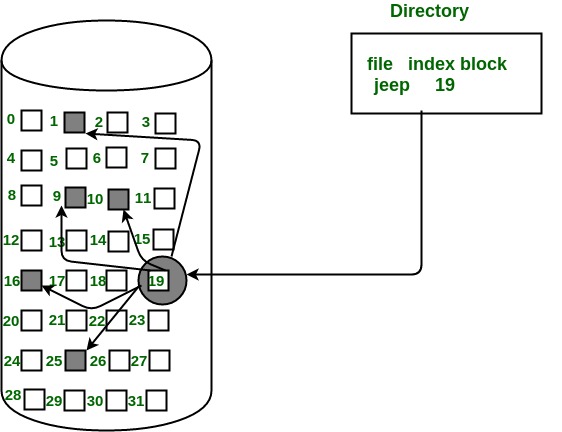
## This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

## Disadvantages:

## Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.

**3. Indexed Allocation**

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



**Advantages:**

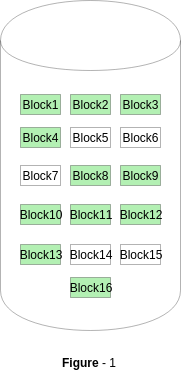
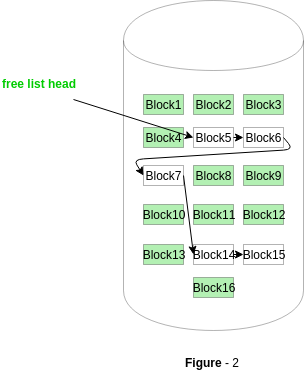
* This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
* It overcomes the problem of external fragmentation.

**Disadvantages:**

* The pointer overhead for indexed allocation is greater than linked allocation.
* For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

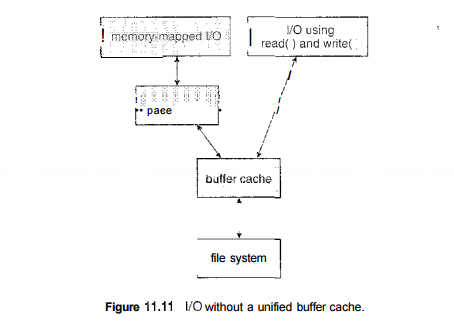
# **Free space management**

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. **Bitmap or Bit vector –**A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.  
   The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.  
     
   **Advantages –**
   * Simple to understand.
   * Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.
2. **Linked List –**In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.  
     
   In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.  
   A drawback of this method is the I/O required for free space list traversal.
3. **Grouping –**This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks.  
   An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.
4. **Counting –**This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.  
   Every entry in the list would contain:
   * Address of first free disk block
   * A number n
5. For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

**Performance**

Even after the basic file-system algorithms have been selected, we can still improve performance in several ways. As will be discussed in Chapter 13, most disk controllers include local memory to form an on-board cache that is large enough to store entire tracks at a time. Once a seek is performed, the track is read into the disk cache starting at the sector under the disk head (reducing latency time). The disk controller then transfers any sector requests to the operating system. Once blocks make it from the disk controller into main memory, the operating system may cache the blocks there. Some systems maintain a separate section of main memory for a buffer cache, where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a page cache.



**Efficiency**

The efficient use of disk space depends heavily on the disk allocation and directory algorithms in use. For instance, UNIX inodes are preallocated on a volume. Even an "empty" disk has a percentage of its space lost to inodes. However, by preallocating the inodes and. spreading them across the volume, we improve the file system's performance. This improved performance results from the UNIX allocation and free-space algorithms, which try to keep a file's data blocks near that file's inode block to reduce seek time. As another example, performance at the cost of internal fragmentation.

### Mass-Storage Management

main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the principal on-line storage medium for both programs and data. Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

The operating system is responsible for the following activities in connection with disk management:

* Free-space management
* Storage allocation
* Disk scheduling Because secondary storage is used frequently, it must be used efficiently.

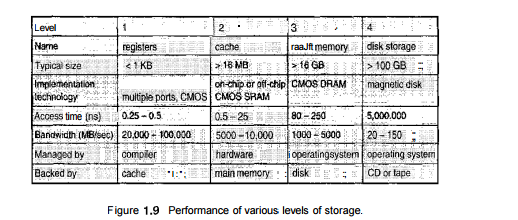
The entire speed of operation of a computer may hinge on the speeds of the disk subsystem and of the algorithms that manipulate that subsystem. There are, however, many uses for storage that is slower and lower in cost (and sometimes of higher capacity) than secondary storage. Backups of disk data, seldom-used data, and long-term archival storage are some examples.

Magnetic tape drives and their tapes and CD and DVD drives and platters are typical tertiary storage devices. The media (tapes and optical platters) vary between WORM (write-once, read-many-times) and RW (read-write) formats.

Tertiary storage is not crucial to system performance, but it still must be managed. Some operating systems take on this task, while others leave tertiary-storage management to application programs. Some of the functions that operating systems can provide include mounting and unmounting media in devices, allocating and freeing the devices for exclusive use by processes, and migrating data from secondary to tertiary storage.

## Caching

Caching is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system—the cache—on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache; if it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon. In addition, internal programmable registers, such as index registers, provide a high-speed cache for main memory.



The programmer (or compiler) implements the register-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory. There are also caches that are implemented totally in hardware.

**Storage structure**

we want the programs and data to reside in [main memory](https://www.geeksforgeeks.org/levels-of-memory-in-operating-system/) permanently.

This arrangement is usually not possible for the following two reasons:

1. Main memory is usually to small to store all needed programs and data permanently.
2. Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

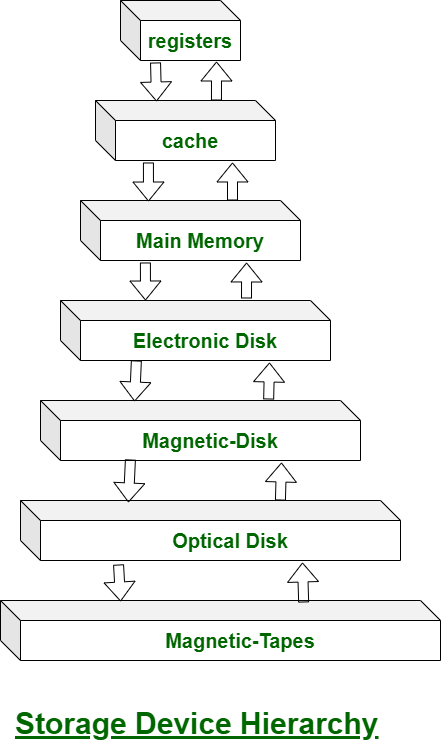
There are two types of storage devices:-

* **Volatile Storage Device –**It looses its contents when the power of the device is removed.
* **Non-Volatile Storage device –**It does not loses its contents when the power is removed. It holds all the data when the power is removed.

[Secondary Storage](https://www.geeksforgeeks.org/introduction-of-secondary-memory/) is used as an extension of main memory. Secondary storage devices can hold the data permanently.

Storage devices consists of [Registers](https://www.geeksforgeeks.org/different-classes-of-cpu-registers/), [Cache](https://www.geeksforgeeks.org/cache-memory-in-computer-organization/), Main-Memory, Electronic-Disk, [Magnetic-Disk](https://www.geeksforgeeks.org/differences-between-magnetic-tape-and-magnetic-disk/), [Optical-Disk](https://www.geeksforgeeks.org/difference-between-magnetic-disk-and-optical-disk/), [Magnetic-Tapes](https://www.geeksforgeeks.org/magnetic-tape-memory/). Each storage system provides the basic system of storing a datum and of holding the datum until it is retrieved at a later time. All the storage devices differ in speed, cost, size and volatility. The most common Secondary-storage device is a Magnetic-disk, which provides storage for both programs and data.

In this fig Hierarchy of storage is shown –



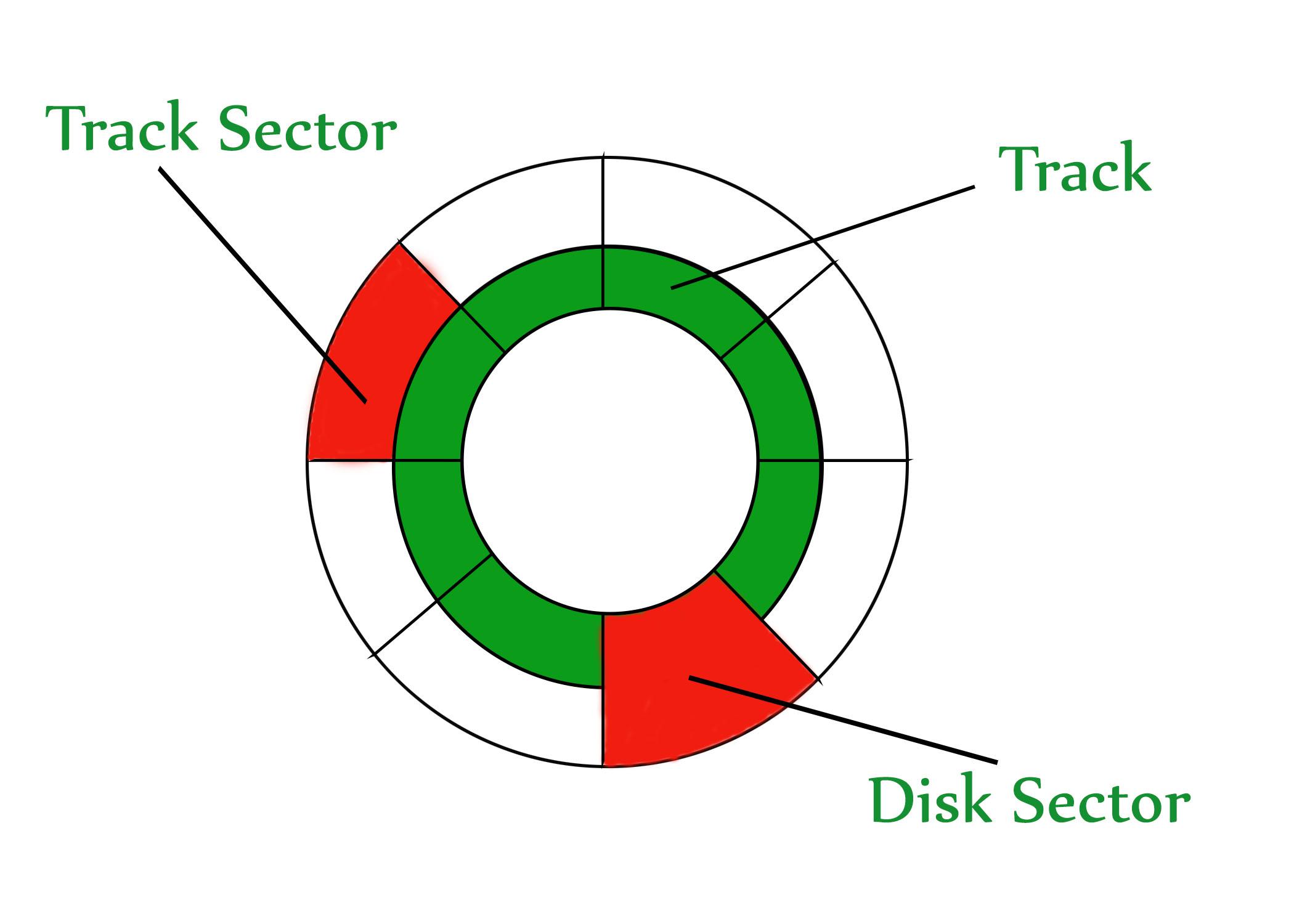
In this hierarchy all the storage devices are arranged according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, where as the access time generally increases.

The storage systems above the Electronic disk are Volatile, where as those below are Non-Volatile.

An Electronic disk can be either designed to be either Volatile or Non-Volatile. During normal operation, the electronic disk stores data in a large [DRAM](https://www.geeksforgeeks.org/difference-between-sram-and-dram/) array, which is Volatile. But many electronic disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic disk controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into the [RAM](https://www.geeksforgeeks.org/different-types-ram-random-access-memory/).

**Disk structure**

A hard disk is a memory storage device which looks like this:



The disk is divided into tracks. Each track is further divided into sectors. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity. This is because the storage density is high in sectors of the inner tracks where as the bits are sparsely arranged in sectors of the outer tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.

## **Disk Attachment**

**Introduction:**-Computers access disk storage in two ways. One way is via I/O ports (or host-attached storage); this is common on small systems. The other way is via a remote host in a distributed file system; this is referred to as network-attached storage.

**Host-Attached Storage:**-Host-attached storage is storage accessed through local I/O ports. These ports use several technologies. The typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supports a maximum of two drives per I/O bus. A newer, similar protocol that has simplified cabling is SATA. High-end workstations and servers generally use more sophisticated I/O architectures, such as SCSI and fiber channel (FC).

SCSI is a bus architecture. Its physical medium is usually a ribbon cable having a large number of conductors. The SCSI protocol supports a maximum of 16 devices on the bus. Generally, the devices include one controller card in the host (the SCSI initiator) and up to 15 storage devices (the SCSI targets). A SCSI disk is a common SCSI target, but the protocol provides the ability to address up to 8 logical units in each SCSI target. A typical use of logical unit addressing is to direct commands to components of a RATD array or components of a removable media library (such as a CD jukebox sending commands to the media-changer mechanism or to one of the drives).

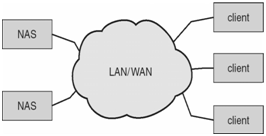
FC is a high-speed serial architecture that can operate over optical fiber or over a four-conductor copper cable. It has two variants. One is a large switched fabric having a 24-bit address space. This variant is expected to dominate in the future and is the basis of storage-area networks (SANs). Because of the large address space and the switched nature of the communication, multiple hosts and storage devices can attach to the fabric, allowing great flexibility in I/O communication. The other PC variant is an arbitrated loop (FC-AL) that can address 126 devices (drives and controllers).

A wide variety of storage devices are suitable for use as host-attached storage. Among these are hard disk drives, RAID arrays, and CD, DVD, and tape drives. The I/O commands that initiate data transfers to a host-attached storage device are reads and writes of logical data blocks directed to specifically identified storage units.

**Network-Attached Storage:** - A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network. Clients access network-attached storage via a remote-procedure-call interface such as NFSfor UNIX systems or CIFS for Windows machines. The remote procedure calls(RPCs) are carried via TCP or UDP over an IP network, usually the same local-area network (LAN) that carries all data traffic to the clients. The network attached storage unit is usually implemented as a RAID array with software that implements the RPC interface. It is easiest to think of NAS as simply another storage-access protocol. For example, rather than using a SCSI device driver and SCSI protocols to access storage, a system using NAS would use RPC overTCP/IP.

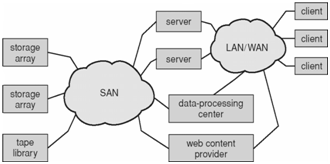
Network-attached storage provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host-attached storage. However, it tends to be less efficient and have lower performance than some direct-attached storage options.

ISCSI is the latest network-attached storage protocol. In essence, it uses the IP network protocol to carry the SCSI protocol. Thus, networks rather than SCSI cables can be used as the interconnects between hosts and their storage. As a result, hosts can treat their storage as if it were directly attached, but the storage can be distant from the host.



**Storage-Area Network:** -One drawback of network-attached storage systems is that the storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication. This problem can be particularly acute in large client-server installations—the communication between servers and clients competes for bandwidth with the communication among servers and storage devices.

A storage-area network (SAN) is a private network (using storage protocols rather than networking protocols) connecting servers and storage units, as shown in Figure. The power of a SAN lies in its flexibility. Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. A SAN switch allows or prohibits access between the hosts and the storage. As one example, if a host is running low on disk space, the SAN can be configured to allocate more storage to that host

.

**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.

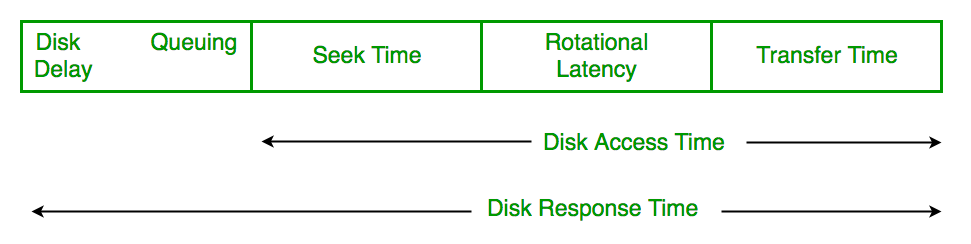
Disk scheduling is important because:

* 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:

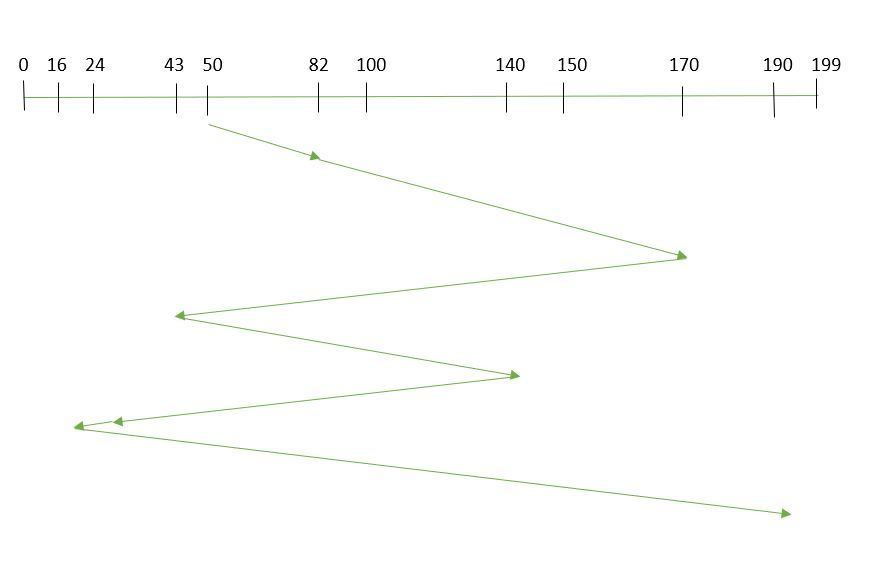
* **Seek Time:**Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
* **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk 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**

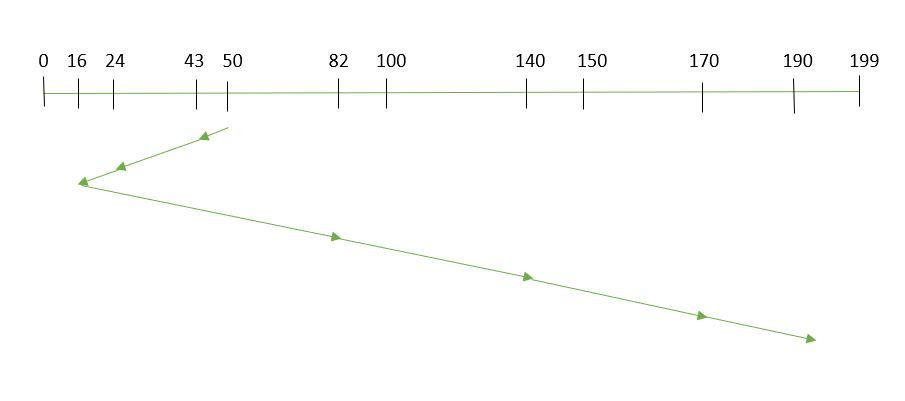
**FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.Let us understand this with the help of an example.  
  
**Example:**Suppose the order of request is- (82,170,43,140,24,16,190)  
And current position of Read/Write head is : 50  
  
So, total seek time:  
=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)  
=642

Advantages:

* Every request gets a fair chance
* No indefinite postponement

Disadvantages:

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

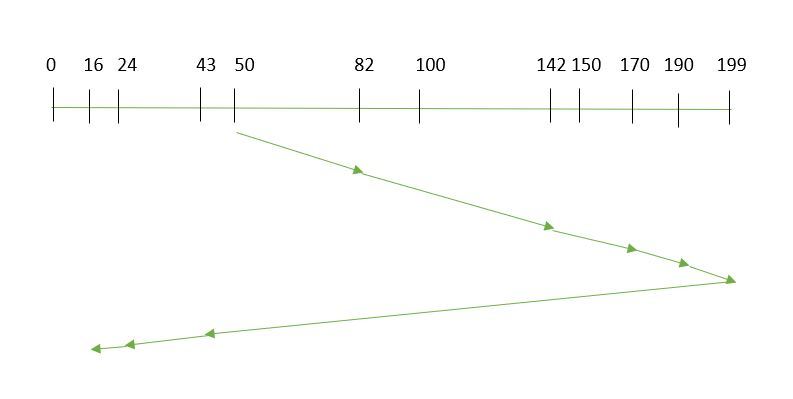
**SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.Let us understand this with the help of an example.  
  
**Example:**Suppose the order of request is- (82,170,43,140,24,16,190)  
And current position of Read/Write head is : 50  
  
So, total seek time:  
=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-40)+(190-170)  
=208

Advantages:

* Average Response Time decreases
* Throughput increases

Disadvantages:

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

**SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.  
  
**Example:**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:  
=(82-50)+(140-82)+(170-140)+(190-170)+(190-43)+(43-24)+(24-16)

=314

Advantages:

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

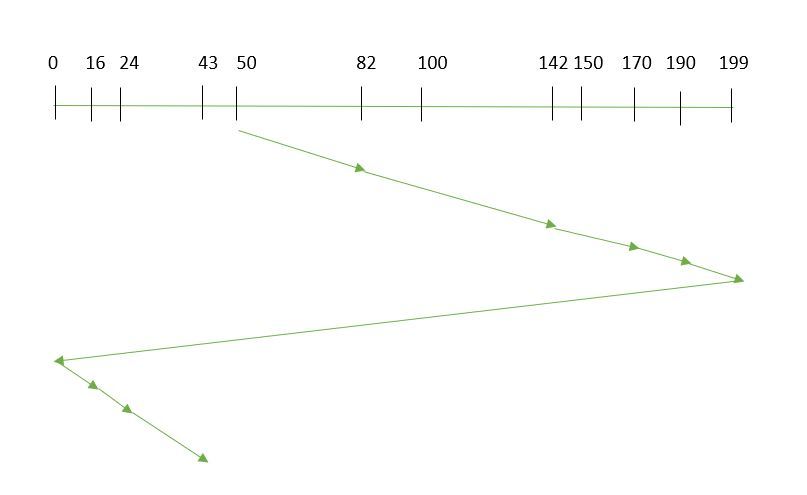
Disadvantages:

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

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

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

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

****

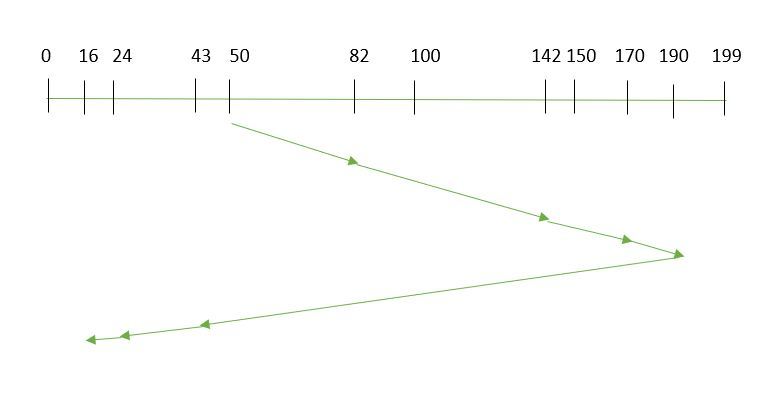
Seek time is calculated as:

=(82-50)+(140-82)+(170-140)+(190+170)+(199-190)+(16-0)+(24-16)+(43-24)

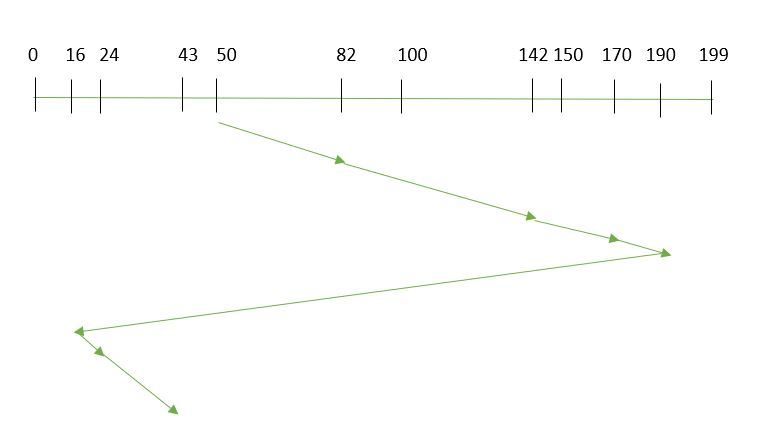
=192

Advantages:

* Provides more uniform wait time compared to SCAN

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

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

=314  
=**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:**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 above  
=(82-50)+(140-82)+(170-140)+(190-170)+(190-16)+(24-16)+(43-24)

=341

**Swap-Space Management**

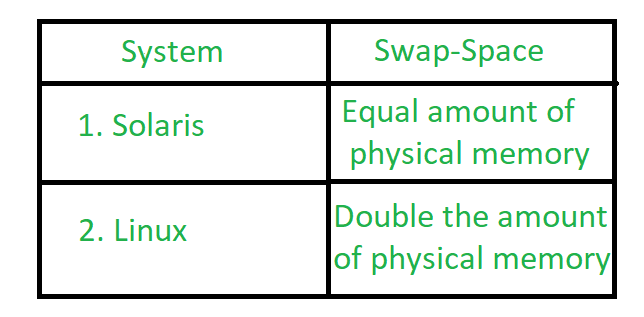
**Swapping** is a memory management technique used in multi-programming to increase the number of processes sharing the CPU. It is a technique of removing a process from main memory and storing it into secondary memory, and then bringing it back into main memory for continued execution. This action of moving a process out from main memory to secondary memory is called **Swap Out** and the action of moving a process out from secondary memory to main memory is called **Swap In**.

**Swap-Space :**The area on the disk where the swapped out processes are stored is called swap space.

**Swap-Space Management :**Swap-Swap management is another low-level task of the operating system. Disk space is used as an extension of main memory by virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decrease system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput

**Swap-Space Use :**Swap-space is used by the different operating-systems in various ways. The systems which are implementing swapping may use swap space to hold the entire process which may include image, code and data segments. Paging systems may simply store pages that have been pushed out of the main memory. The need for swap space on a system can vary from a megabytes to gigabytes but it also depends on the amount of physical memory, the virtual memory it is backing and the way in which it is using the virtual memory.

It is safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort the processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it does not harm other.Following table shows different system using amount of swap space:



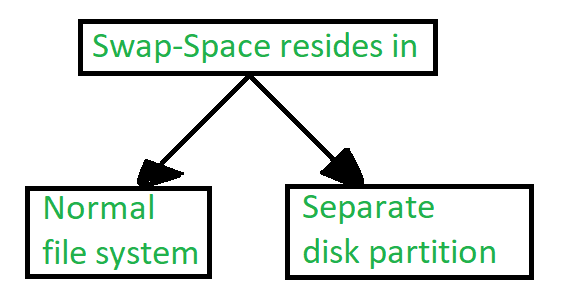
**Figure –** Different systems using amount of swap-space

**Explanation of above table :**

Solaris, setting swap space equal to the amount by which virtual memory exceeds page-able physical memory. In the past Linux has suggested setting swap space to double the amount of physical memory. Today, this limitation is gone, and most Linux systems use considerably less swap space.

Including Linux, some operating systems; allow the use of multiple swap spaces, including both files and dedicated swap partitions. The swap spaces are placed on the disk so the load which is on the I/O by the paging and swapping will spread over the system’s bandwidth.

**Swap-Space Location :**

****

**Figure –** Location of swap-space

A swap space can reside in one of the two places –

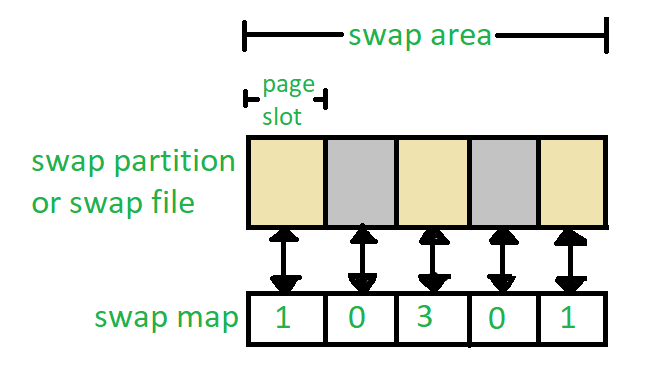
1. Normal file system
2. Separate disk partition

Let, if the swap-space is simply a large file within the file system. To create it, name it and allocate its space, normal **file-system** routines can be used. This approach, though easy to implement, is inefficient. Navigating the directory structures and the disk-allocation data structures takes time and extra disk access. During reading or writing of a process image, **external fragmentation** can greatly increase swapping times by forcing multiple seeks.

There is also an alternative to create the swap space which is in a separate **raw partition**. There is no presence of any file system in this place. Rather, a swap space storage manager is used to allocate and deallocate the blocks. from the raw partition. It uses the algorithms for speed rather than storage efficiency, because we know the access time of swap space is shorter than the file system. **Internal fragmentation** increases, but it is acceptable, because the lifespan of the swap space is shorter than the files in the file system. Raw partition approach creates a fixed amount of swap space in case of the **disk partitioning**.

Some operating systems are flexible and can swap both in raw partitions and in the file system space, example: **Linux**.

**Swap-Space Management: An Example –**



**Figure –** Data structure for swapping on Linux system

Each swap area consists of 4-KB **page slots**, which are used to hold the swapped pages. Associated with each swap area is a **swap-map-** an array of integers counters, each corresponding to a page slot in the swap area. If the value of the counter is 0 it means the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page. For example, a value 3 indicates that the swapped page is mapped to the 3 different processes.

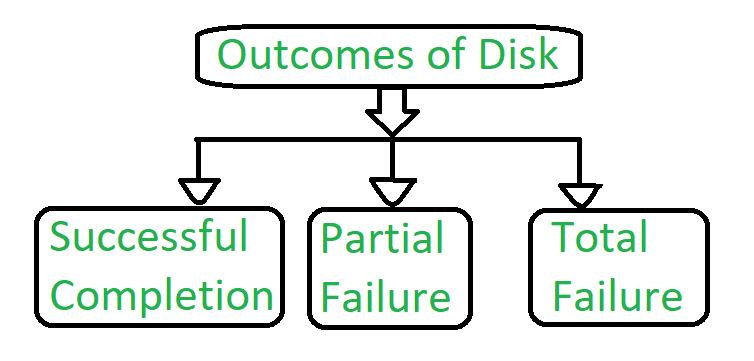
**Stable-Storage Implementation in Operating system**

By definition, information residing in the **Stable-Storage** is never lost. Even if the disk and CPU have some errors, it will never lose any data.

**Stable-Storage Implementation**

To achieve such storage, we need to replicate the required information on multiple storage devices with independent failure modes. The writing of an update should be coordinated in such a way that it would not delete all the copies of the state and that, when we are recovering from a failure, we can force all the copies to a consistent and correct value, even if another failure occurs during the recovery. In these, we discuss how to meet these needs.

The disk write operation results to one of the following outcome:

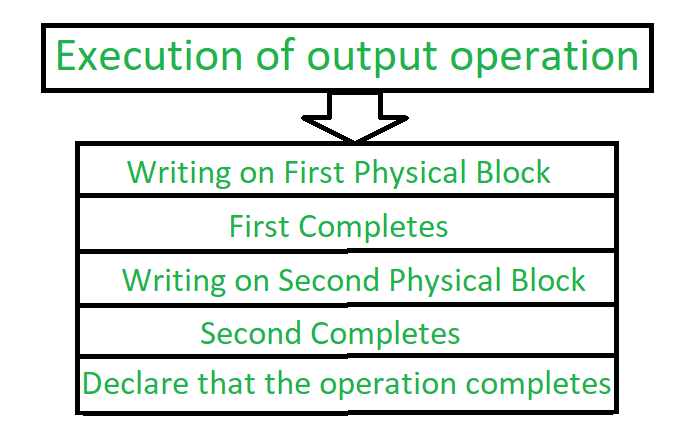


**Figure –** Outcomes of Disk

**Successful completion –**The data will be written correctly on disk.  
**Partial Failure –**In this case, failure occurred in the middle of the data transfer, so that only some sectors were written with the new data, and the sector which was written during the failure may have been corrupted.

**Total Failure –**The failure occurred before the disk write started, so the previous data values on the disk remain intact.While writing a block somehow if failure occurs, the system’s first work is to detect the failure and then invoke a recovery process to restore the consistent state. To do that, the system must contain two physical blocks for each logical block.

An output operation is executed as follows:



**Figure –** Process of execution of output operation

1. Write the information onto the first physical block.
2. When the first write completes successfully, write the same operation onto the second physical block.
3. When both the operation declares successfully, declare the operation as complete.

During the recovery from a failure each of the physical blocks is examined. If both are the same and no detectable error exists, then no further action is necessary. If one block contains detectable errors then we replace its content with the value of the other block. If neither block contains the detectable error, but the block differs in content, then we replace the content of first block with the content of the second block.This procedure of the recovery gives us an conclusion that either the write to stable content succeeds successfully or it results in no change.

This procedure will be extended if we want an arbitrarily large number of copies of each block of the stable storage. With the usage of a large number of copies, the chances of failure reduces. Generally, it is usually reasonable to simulate stable storage with only two copies.. The data that is present in the stable storage is guaranteed to be safe unless a failure destroys all the copies.

Because waiting for disk writes to complete is time consuming, many storage arrays add NVRAM as a cache. Since the memory is no-volatile it can be trusted to store the data en route to the disks. In this way it is considered as a part of the stable storage. Writing to the stable storage is much faster than to disk, so performance is greatly improved.

### Tertiary-Storage Structure

* Primary storage refers to computer memory chips; Secondary storage refers to fixed-disk storage systems ( hard drives ); And ***Tertiary Storage*** refers to ***removable media,*** such as tape drives, CDs, DVDs, and to a lesser extent floppies, thumb drives, and other detachable devices.
* Tertiary storage is typically characterized by large capacity, low cost per MB, and slow access times, although there are exceptions in any of these categories.
* Tertiary storage is typically used for backups and for long-term archival storage of completed work. Another common use for tertiary storage is to swap large little-used files ( or groups of files ) off of the hard drive, and then swap them back in as needed in a fashion similar to secondary storage providing swap space for primary storage.

**Tertiary-Storage Devices**

#### Removable Disks

Removable magnetic disks ( e.g. floppies ) can be nearly as fast as hard drives, but are at greater risk for damage due to scratches. Variations of removable magnetic disks up to a GB or more in capacity have been developed. ( Hot-swappable hard drives? )

A ***magneto-optical*** disk uses a magnetic disk covered in a clear plastic coating that protects the surface.

The heads sit a considerable distance away from the magnetic surface, and as a result do not have enough magnetic strength to switch bits ***at normal room temperature.***

For writing, a laser is used to heat up a specific spot on the disk, to a temperature at which the weak magnetic field of the write head is able to flip the bits.

For reading, a laser is shined at the disk, and the ***Kerr effect*** causes the polarization of the light to become rotated either clockwise or counter-clockwise depending on the orientation of the magnetic field.

***Optical disks*** do not use magnetism at all, but instead use special materials that can be altered ( by lasers ) to have relatively light or dark spots.

The most common examples of these disks are ***re-writable*** CD-RWs and DVD-RWs.

An alternative to the disks described above are ***Write-Once Read-Many, WORM*** drives.

The original version of WORM drives involved a thin layer of aluminum sandwiched between two protective layers of glass or plastic.

Holes were burned in the aluminum to write bits.

Because the holes could not be filled back in, there was no way to re-write to the disk. ( Although data could be erased by burning more holes. )

WORM drives have important legal ramifications for data that must be stored for a very long time and must be provable in court as unaltered since it was originally written. ( Such as long-term storage of medical records. )

Modern CD-R and DVD-R disks are examples of WORM drives that use organic polymer inks instead of an aluminum layer.

Read-only disks are similar to WORM disks, except the bits are pressed onto the disk at the factory, rather than being burned on one by one

# **Protection Mechanisms**

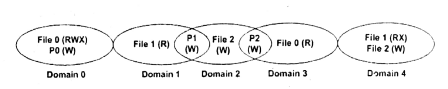
Protection plays a very crucial role in a multiuser environment, where several users will be making concurrent use of the computer resources such as CPU, memory etc. It is the duty of the operating system to provide a mechanism that protects each process from others.

All the items that require protection in a multiuser environment are listed as objects and those that want to access these objects are known as subjects. The operating system grants different 'access rights' to different subjects.

These rights may include read, write, execute, append, delete etc.

### 1. Domain

A domain is a combination of different objects and a set of different 'access rights' that can be granted to different subjects to operate on each of these objects. An operating system maintains several such domains with different combinations of access rights. The user processes can execute in one of those domains and can access the objects in that domain according to the access rights given to those objects.



**Protection domain**

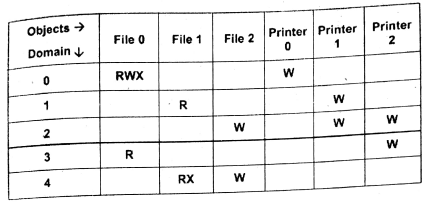
A user process executing in domain 0 has access to read from, write into and execute the file 0 and can write to printer P0. Similarly, the process executing in domain 1 has access to read from file 1. The printer P1 is common to both domain 1 and domain 2. The processes executing in domain 1 and domain 2 both can have access to printer P1

In matrix form, the above image can be represented as shown in the below image.

During the execution of a process, it may become necessary for it to access an object, which is in another domain. If it has a right to access that object it switches to the new domain and accesses that file. This process is known as domain switching.

### Implementation of Access Matrix

The access matrix can be implemented by using either access control lists or capability lists.



**Protection matrix**

In ACL, the data is stored by column by the operating system. The information about the users and their access rights for each file is maintained by the operating system. The empty entries are discarded.

In capability lists, the access control matrix is sliced horizontally by a row. This implies that the operating system will have to maintain for each user a list of all the objects that the user can access and the ways in which he can access them. A combination of ACL and capability list techniques may also be used to design protection mechanisms.

### Encryption

It is one of the most powerful and important tools of protection. The process of encryption involves two steps: encryption of the original data into some other form about which nothing is known to the third person and decryption of the data into the original form from the encrypted form.

The most commonly used methods to achieve encryption are: transposition ciphers and substitution ciphers.

In transposition ciphers, the letters in the original message are not changed; only the order in which they are contained in the original message gets changed. For example, consider that the message 'it is raining' needs to be encrypted. It will become 'gniniar si ti' in the encrypted form using a particular form of transposition ciphers algorithm.

The set of characters in the encrypted form will be different from the original ones if we use substitution ciphers. Every letter may be replaced by its previous alphabet, for instance. Now the message 'it is raining' would become, after encryption, 'hs hr qzhmhmf'.

It is very easy to implement these ciphers for characters. The varied forms of these algorithms can be used to encrypt bit streams. For instance, a predetermined bit stream may be added to the bits in the original stream at a particular position to obtain the encrypted message. The same bit of steam is subtracted at the destination so that the original stream is obtained. This addition and subtraction may be accomplished with the help of simple adder and subtractor circuits.

The key idea behind the encryption schemes is that the encryption process must be restorable. Means, once we encrypt the original message to a different form, there should be a way to restore it to the original form

**OS security issues:**

## **Intruders**

In the computer security world, people who want to cause some trouble for their fun or for their own commercial profit are called intruders.

Basically, intruders are of the following two types:

* Active intruders
* Passive intruders

Now, let's describe briefly about the above two different types of intruders.

## **Active Intruders**

Active intruders are malicious.

Active intruders always want to make some unauthorized access to the other's system to change, modify, or sometimes delete the data.

## **Passive Intruders**

Passive intruders are less malicious than active one.

Passive intruders want to read the files they aren't authorized to read.

Some of the most common types of ***violations*** include:

***Breach of Confidentiality -*** Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.

***Breach of Integrity -*** Unauthorized ***modification*** of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.

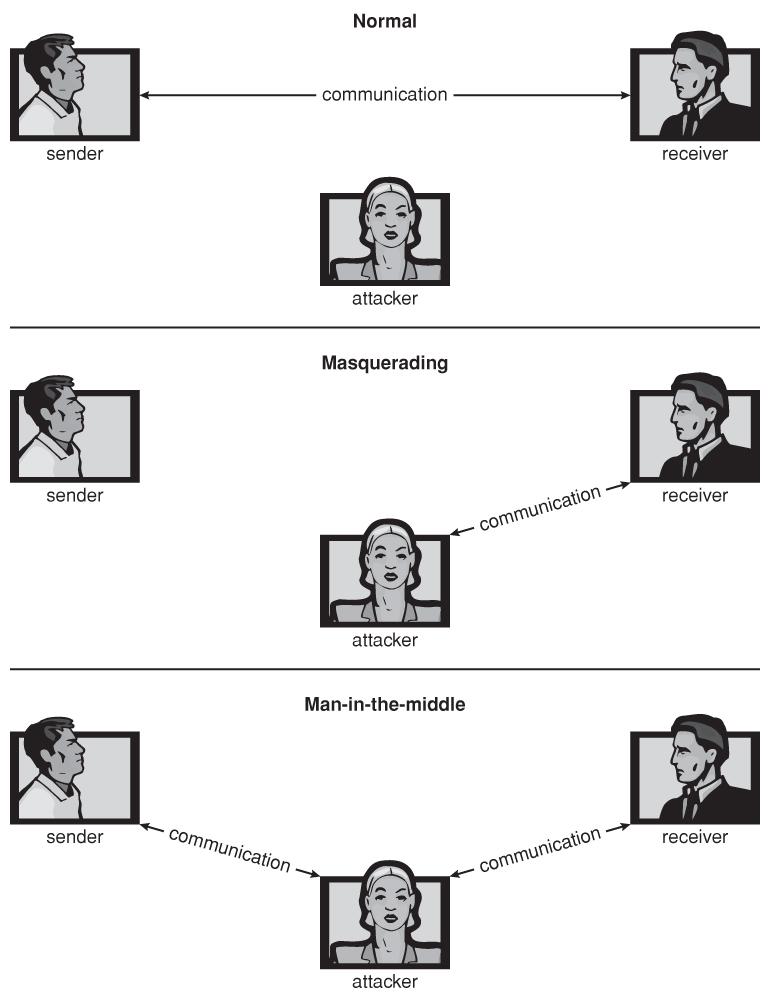
***Breach of Availability -*** Unauthorized ***destruction*** of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.

***Theft of Service -*** Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.

***Denial of Service, DOS -*** Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.

One common attack is ***masquerading,*** in which the attacker pretends to be a trusted third party. A variation of this is the ***man-in-the-middle,*** in which the attacker masquerades as both ends of the conversation to two targets.

A ***replay attack*** involves repeating a valid transmission. Sometimes this can be the entire attack, ( such as repeating a request for a money transfer ), or other times the content of the original message is replaced with malicious content.



**Figure 15.1 - Standard security attacks.**

There are four levels at which a system must be protected:

**Physical -** The easiest way to steal data is to pocket the backup tapes. Also, access to the root console will often give the user special privileges, such as rebooting the system as root from removable media. Even general access to terminals in a computer room offers some opportunities for an attacker, although today's modern high-speed networking environment provides more and more opportunities for remote attacks.

**Human -** There is some concern that the humans who are allowed access to a system are trustworthy, and that they cannot be coerced into breaching security. However more and more attacks today are made via ***social engineering,*** which basically means fooling trustworthy people into accidentally breaching security.

**Phishing** involves sending an innocent-looking e-mail or web site designed to fool people into revealing confidential information. E.g. spam e-mails pretending to be from e-Bay, PayPal, or any of a number of banks or credit-card companies.

**Dumpster Diving** involves searching the trash or other locations for passwords that are written down. ( Note: Passwords that are too hard to remember, or which must be changed frequently are more likely to be written down somewhere close to the user's station. )

**Password Cracking** involves divining users' passwords, either by watching them type in their passwords, knowing something about them like their pet's names, or simply trying all words in common dictionaries. ( Note: "Good" passwords should involve a minimum number of characters, include non-alphabetical characters, and not appear in any dictionary ( in any language ), and should be changed frequently. Note also that it is proper etiquette to look away from the keyboard while someone else is entering their password. )

**Operating System -** The OS must protect itself from security breaches, such as runaway processes ( denial of service ), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.

**Network -** As network communications become ever more important and pervasive in modern computing environments, it becomes ever more important to protect this area of the system. ( Both protecting the network itself from attack, and protecting the local system from attacks coming in through the network. ) This is a growing area of concern as wireless communications and portable devices become more and more prevalent.

**Program Threats**

There are many common threats to modern systems. Only a few are discussed here.

#### Trojan Horse

A ***Trojan Horse*** is a program that secretly performs some maliciousness in addition to its visible actions.

Some Trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with ***viruses,***

One dangerous opening for Trojan horses is long search paths, and in particular paths which include the current directory ( "." ) as part of the path. If a dangerous program having the same name as a legitimate program ( or a common mis-spelling, such as "sl" instead of "ls" ) is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake.

Another classic Trojan Horse is a login emulator, which records a users account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again ( with a proper login prompt ), logs in successfully, and doesn't realize that their information has been stolen.

***Spyware*** is a version of a Trojan Horse that is often included in "free" software downloaded off the Internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. ( This is an example of ***covert channels,*** in which surreptitious communications occur. ) Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user.

#### Trap Door

A ***Trap Door*** is when a designer or a programmer ( or hacker ) deliberately inserts a security hole that they can use later to access the system.

Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door.

A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

#### Logic Bomb

A ***Logic Bomb*** is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.

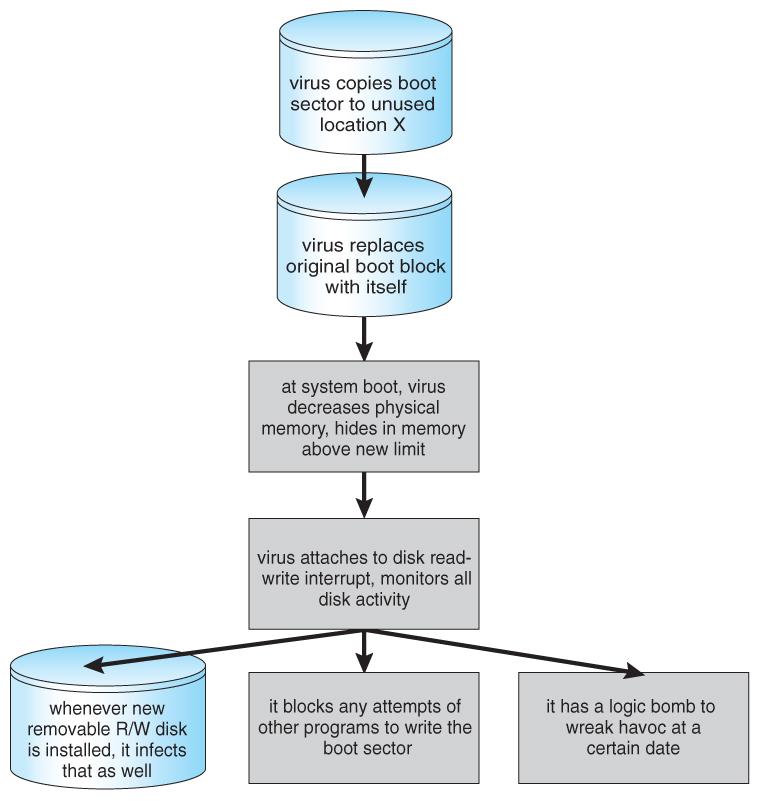
A classic example is the ***Dead-Man Switch***, which is designed to check whether a certain person ( e.g. the author ) is logging in every day, and if they don't log in for a long time ( presumably because they've been fired ), then the logic bomb goes off and either opens up security holes or causes other problems.

#### Stack and Buffer Overflow

This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow.

#### Viruses

* A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself ( by infecting other programs ), and ( eventually ) wreaking havoc.
* Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures ( such as the boot block. )
* Viruses are delivered to systems in a ***virus dropper,*** usually some form of a Trojan Horse, and usually via e-mail or unsafe downloads.
* Viruses take many forms ( see below. ) Figure 15.5 shows typical operation of a boot sector virus:



**Figure 15.5 - A boot-sector computer virus.**

* Some of the forms of viruses include:
  + **File -** A file virus attaches itself to an executable file, causing it to run the virus code first and then jump to the start of the original program. These viruses are termed ***parasitic,*** because they do not leave any new files on the system, and the original program is still fully functional.
  + **Boot -** A boot virus occupies the boot sector, and runs before the OS is loaded. These are also known as ***memory viruses***, because in operation they reside in memory, and do not appear in the file system.
  + **Macro -** These viruses exist as a macro ( script ) that are run automatically by certain macro-capable programs such as MS Word or Excel. These viruses can exist in word processing documents or spreadsheet files.
  + **Source code** viruses look for source code and infect it in order to spread.
  + **Polymorphic** viruses change every time they spread - Not their underlying functionality, but just their ***signature,*** by which virus checkers recognize them.
  + **Encrypted** viruses travel in encrypted form to escape detection. In practice they are self-decrypting, which then allows them to infect other files.
  + **Stealth** viruses try to avoid detection by modifying parts of the system that could be used to detect it. For example the read( ) system call could be modified so that if an infected file is read the infected part gets skipped and the reader would see the original unadulterated file.
  + **Tunneling** viruses attempt to avoid detection by inserting themselves into the interrupt handler chain, or into device drivers.
  + **Multipartite** viruses attack multiple parts of the system, such as files, boot sector, and memory.
  + **Armored** viruses are coded to make them hard for anti-virus researchers to decode and understand. In addition many files associated with viruses are hidden, protected, or given innocuous looking names such as "...".
* In 2004 a virus exploited three bugs in Microsoft products to infect hundreds of Windows servers ( including many trusted sites ) running Microsoft Internet Information Server, which in turn infected any Microsoft Internet Explorer web browser that visited any of the infected server sites. One of the back-door programs it installed was a ***keystroke logger,*** which records users keystrokes, including passwords and other sensitive information.
* There is some debate in the computing community as to whether a ***monoculture,*** in which nearly all systems run the same hardware, operating system, and applications, increases the threat of viruses and the potential for harm caused by them.

### System and Network Threats

Most of the threats described above are termed ***program threats***, because they attack specific programs or are carried and distributed in programs. The threats in this section attack the operating system or the network itself, or leverage those systems to launch their attacks.

#### Worms

A ***worm*** is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system. Worms consume system resources, often blocking out other, legitimate processes.

****