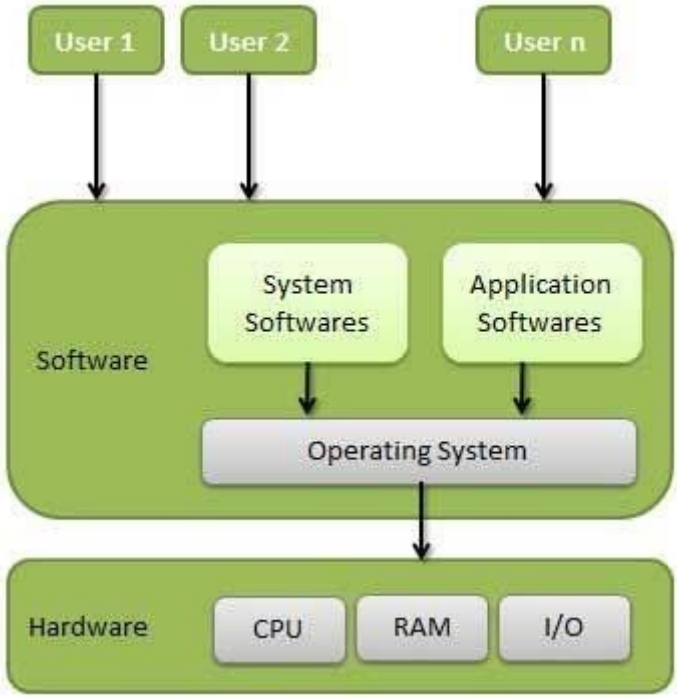
Introduction to Operating System

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.



Operating System Services:

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

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

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process. A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use)

IO operations

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 −

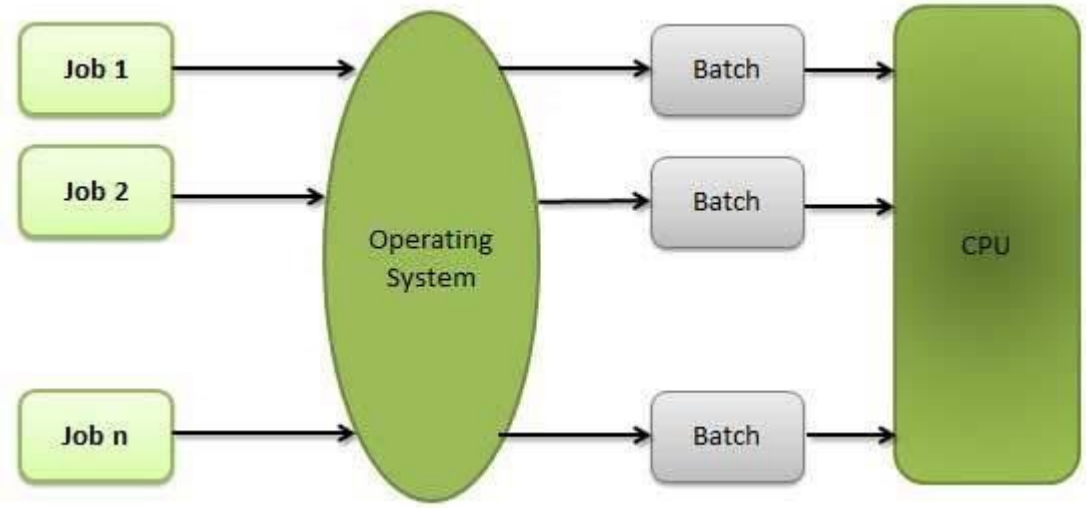
- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

Types of operating system:

a) Batch processing

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts. An operating system does the following activities related to batch processing −

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.



Advantages

- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention.

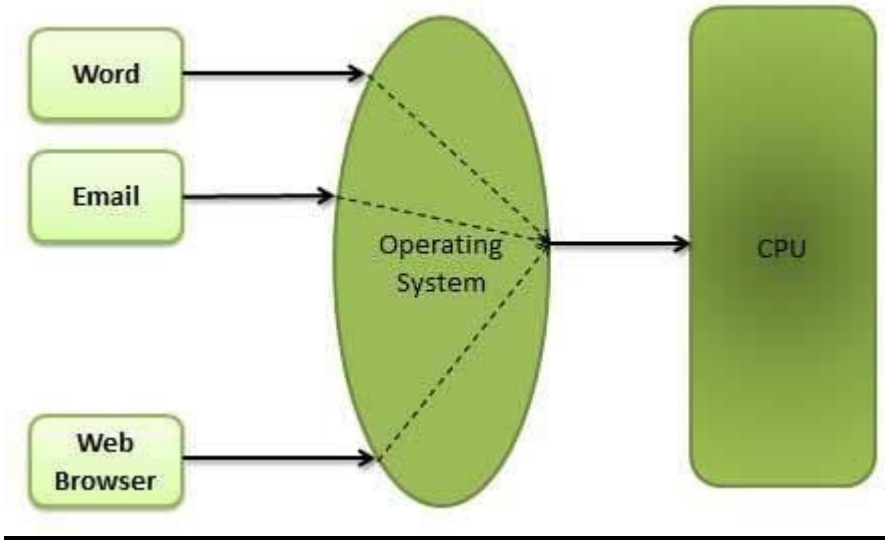
Disadvantages

- Difficult to debug program.
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

b) Multitasking/Timeshared

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking −

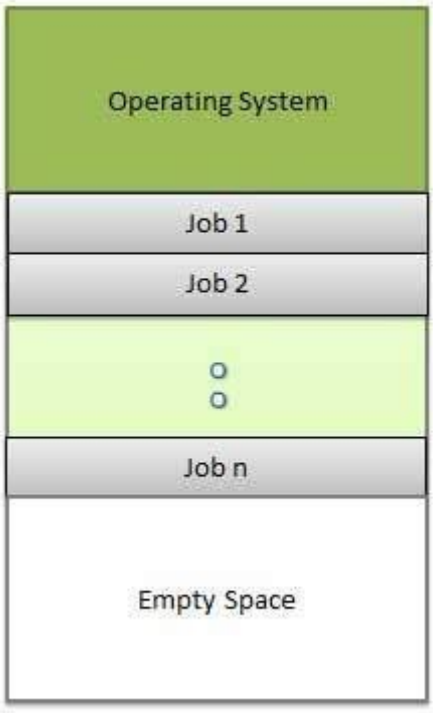
- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.



c) Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as **multiprogramming**. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.



An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.

- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

d) Real Time System

Real-time systems are usually dedicated, embedded systems. An operating system does the following activities related to real-time system activity.

- In such systems, Operating Systems typically read from and react to sensor data.
- The Operating system must guarantee response to events within fixed periods of time to ensure correct performance.

e) Distributed Environment

A distributed environment refers to multiple independent CPUs or processors in a computer system. An operating system does the following activities related to distributed environment.

- The OS distributes computation logics among several physical processors.
- The processors do not share memory or a clock. Instead, each processor has its own local memory.
- The OS manages the communications between the processors. They communicate with each other through various communication lines.

Generations of Operating Systems:

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems. These can be described as follows

The First Generation (1945 - 1955): Vacuum Tubes and Plugboards

Digital computers were not constructed until the Second World War. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

The Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The Batch System was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced multiprogramming. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

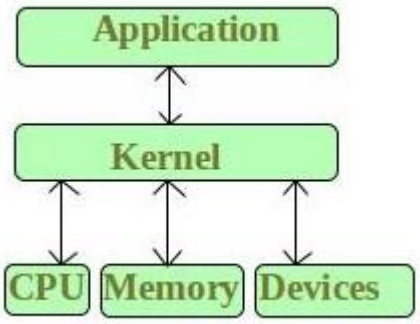
System calls and System Programs

System calls provide an interface between the process and the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls.

Microkernel in Operating Systems

Kernel is the core part of an operating system which manages system resources. It also acts like a bridge between application and hardware of the computer. It is one of the first programs loaded on start-up (after the Bootloader).



Microkernel is one of the classification of the kernel. Being a kernel it manages all system resources. But in a microkernel, the user services and kernel services are implemented in different address space. The user services are kept in user address space, and kernel services are kept under kernel address space, thus also reduces the size of kernel and size of operating system as well.

Microkernel Architecture : Since kernel is the core part of the operating system, so it is meant for handling the most important services only. Thus in this architecture only the most important services are inside kernel and rest of the OS services are present inside system application program. Thus users are able to interact with those not-so important services within the system application. And the microkernel is solely responsible for the most important services of operating system they are named as follows:

- Inter process-Communication
- Memory Management
- CPU-Scheduling

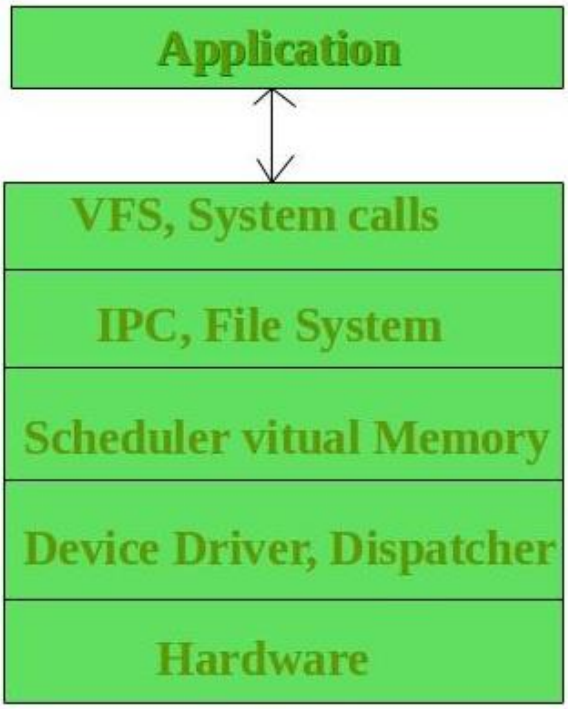
Advantages of Microkernel –

- The architecture of this kernel is small and isolated hence it can function better.
- Expansion of the system is easier, it is simply added in the system application without disturbing the kernel.

Monolithic Kernel

Monolithic Kernel is another classification of Kernel. Like microkernel this one also manages system resources between application and hardware, but user services and kernel services are implemented under same address space. It increases the size of the kernel, thus increases size

of operating system as well. This kernel provides CPU scheduling, memory management, file management and other operating system functions through system calls. As both services are implemented under same address space, this makes operating system execution faster. Below is the diagrammatic representation of Monolithic Kernel.



Monolithic Kernel

Advantages of Monolithic Kernel –

- One of the major advantage of having monolithic kernel is that it provides CPU scheduling, memory management, file management and other operating system functions through system calls.
- The other one is that it is a single large process running entirely in a single address space.
- It is a single static binary file. Example of some Monolithic Kernel based OSs are: Unix, Linux, Open VMS, XTS-400, z/TPF.

Disadvantages of Monolithic Kernel –

- One of the major disadvantage of monolithic kernel is that, if anyone service fails it leads to entire system failure.
- If user has to add any new service. User needs to modify entire operating system.

Key differences between Monolithic Kernel and Microkernel –

Basis for Comparison	Microkernel	Monolithic Kernel
Size	Microkernel is smaller in size	It is larger than microkernel
Execution	Slow Execution	Fast Execution
Extendible	It is easily extendible	It is hard to extend
Security	If a service crashes, it does effects on working on the microkernel	If a service crashes, the whole system crashes in monolithic kernel.
Code	To write a microkernel more code is required	To write a monolithic kernel less code is required
Example	QNX, Symbian, L4Linux etc.	Linux,BSDs(FreeBSD,OpenBSD,NetBSD)etc.

Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example, Virtual Box.

When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques.

Advantages:

1. There are no protection problems because each virtual machine is completely isolated from all other virtual machines.
2. Virtual machine can provide an instruction set architecture that differs from real computers.
3. Easy maintenance, availability and convenient recovery.

Disadvantages:

1. When multiple virtual machines are simultaneously running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.
2. Virtual machines are not as efficient as a real one when accessing the hardware.

UNIX system calls

UNIX system calls are used to manage the file system, control processes and to provide inter-process communication. The UNIX system interface consists of about 80 system calls (as UNIX evolves this number will increase). The following table lists some of the more important system call:

GENERAL CLASS	SPECIFIC CLASS	SYSTEM CALL
File Structure Related Calls	Creating a Channel	creat()
		open()
		close()
	Input/Output	read()
		write()
	Random Access	lseek()
	Channel Duplication	dup()
	Aliasing and Removing Files	link()
		unlink()
	File Status	stat()
		fstat()
	Access Control	access()
		chmod()
		chown()
		umask()
	Device Control	ioctl()
Process Related Calls	Process Creation and Termination	exec()
		fork()
		wait()
		exit()
	Process Owner and Group	getuid()
		geteuid()
		getgid()
		getegid()
	Process Identity	getpid()
		getppid()
	Process Control	signal()
		kill()
		alarm()
	Change Working Directory	chdir()

Shell Interpreter:

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

A shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt:

The prompt, $, which is called command prompt, is issued by the shell. While the prompt is displayed, you can type a command.

The shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Following is a simple example of **date** command which displays current date and time:

$date
Thu Jun 25 08:30:19 MST 2009

You can customize your command prompt using environment variable PS1 explained in Environment tutorial.

Shell Types:

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
2. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

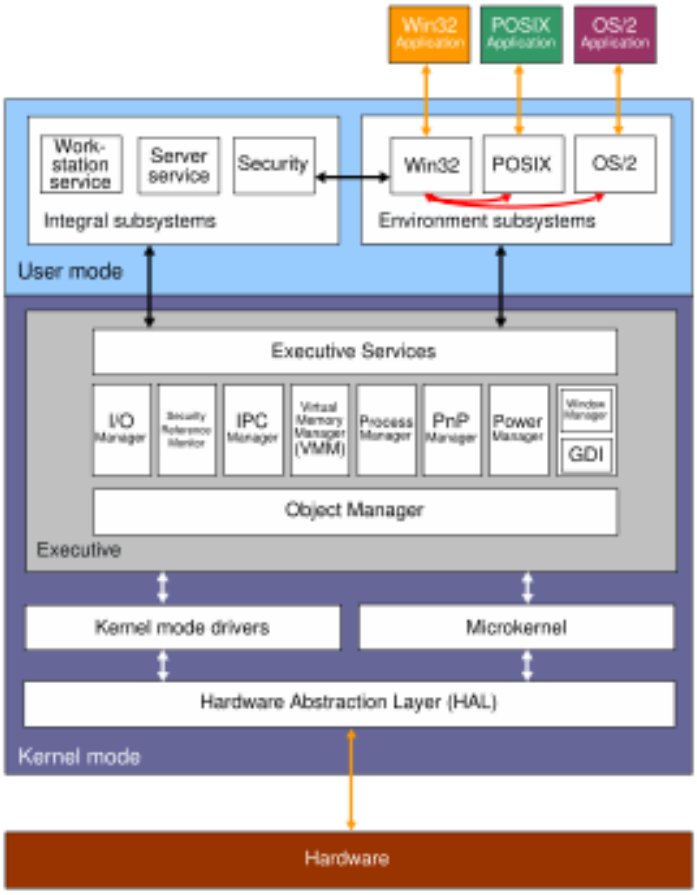
- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow:

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

Architecture of Windows NT

Windows Vista, Windows Server 2003, Windows XP, Windows 2000 and Windows NT are all part of the Windows NT family (NT-based) of Microsoft operating systems. They are all preemptive, operating systems, which have been designed to work with either uniprocessor- or symmetrical multi processor (SMP)-based Intel x86 computers. To process input/output (I/O) requests it uses packet-driven I/O which utilises I/O request packets (IRPs) and asynchronous I/O. The architecture of Windows NT is highly modular, and consists of two main layers: a user mode and a kernel mode.



User mode

The user mode is made up of subsystems which can pass I/O requests to the appropriate kernel mode drivers via the I/O manager. Two subsystems make up the user mode layer of Windows 2000: the *Environment subsystem* and the *Integral subsystem*.

The environment subsystem was designed to run applications written for many different types of operating systems. None of the environment subsystems can directly access hardware, and must request access to memory resources through the Virtual Memory Manager that runs in kernel mode. Currently, there are three main environment subsystems: the Win32 subsystem, an OS/2 subsystem and a POSIX subsystem.

The integral subsystem looks after operating system specific functions on behalf of the environment subsystem. It consists of a *security subsystem*, a *workstation service* and a *server service*

Kernel mode

Windows 2000 kernel mode has full access to the hardware and system resources of the computer and runs code in a protected memory area. It controls access to scheduling, thread prioritisation, memory management and the interaction with hardware. The kernel mode stops user mode services and applications from accessing critical areas of the operating system that they should not have access to as user mode processes ask the kernel mode to perform such operations on its behalf.

Kernel mode consists of *executive services*, which is itself made up on many modules that do specific tasks, *kernel drivers*, a *kernel* and a *Hardware Abstraction Layer*, or HAL.

NTFS (New Technology File System)

NTFS is the file system that the Windows NT operating system uses for storing and retrieving files on a hard disk. NTFS is the Windows NT equivalent of the Windows 95 file allocation table (FAT) and the OS/2 High Performance File System (HPFS). However, NTFS offers a number of improvements over FAT and HPFS in terms of performance, extendibility, and security.

Features of NTFS include:

- Use of a b-tree directory scheme to keep track of file clusters
- Information about a file's clusters and other data is stored with each cluster, not just a governing table (as FAT is)
- Support for very large files (up to 2 to the 64th power or approximately 16 billion bytes in size)
- An access control list (ACL) that lets a server administrator control who can access specific files
- Integrated file compression
- Support for names based on Unicode
- Support for long file names as well as "8 by 3" names
- Data security on both removable and fixed disks

How NTFS Works

When a hard disk is formatted, it is divided into partitions. Within each partition, the operating system keeps track of all the files that are stored by that operating system. Each file is actually stored on the hard disk in one or more clusters or disk spaces of a predefined uniform size. Using NTFS, the sizes of clusters range from 512 bytes to 64 kilobytes. Windows NT provides a recommended default cluster size for any given drive size. For example, for a 4 GB (gigabyte) drive, the default cluster size is 4 KB (kilobytes).

When a file is created using NTFS, a record about the file is created in a special file, the Master File Table (MFT). The record is used to locate a file's possibly scattered clusters. NTFS tries to find contiguous storage space that will hold the entire file.

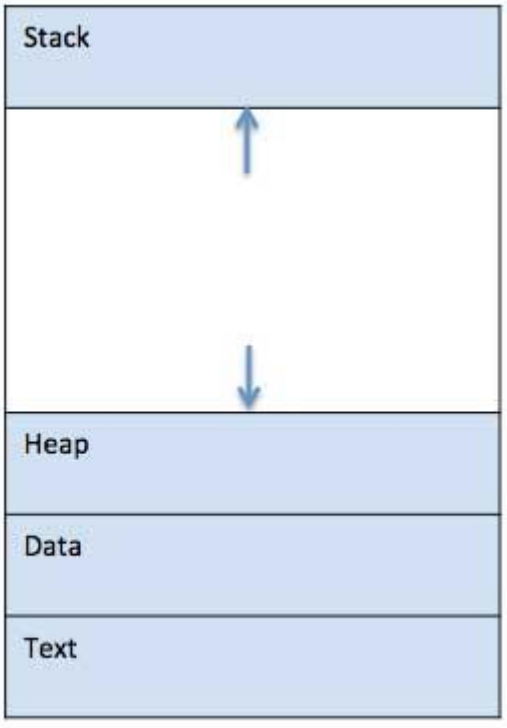
UNIT 2

Process Management

Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion. We write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



Stack The process Stack contains the temporary data such as method/function parameters, return address and local variables

Heap This is dynamically allocated memory to a process during its run time.

Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers

Data This section contains the global and static variables.

Process States

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized. In general, a process can have one of the following five states at a time.

- **Start** This is the initial state when a process is first started or created.
- **Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
- **Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
- **Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
- **Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

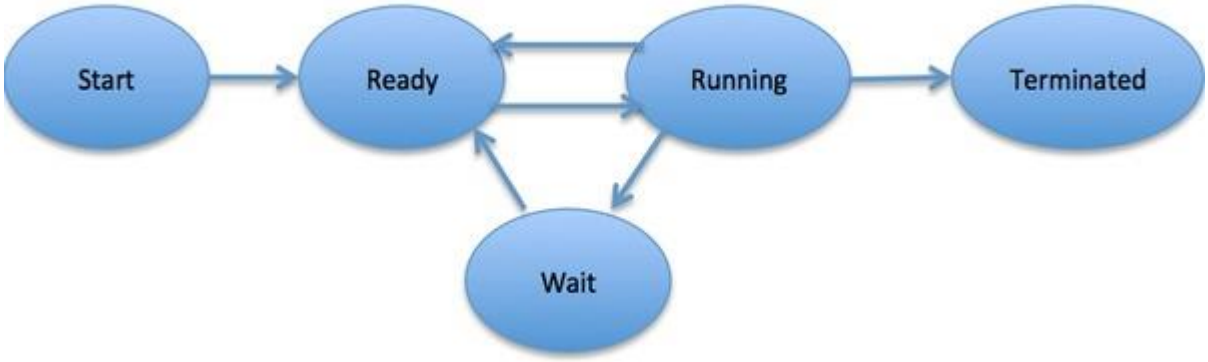


Fig: State Transaction Diagram

Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process.

Process State

The current state of the process i.e., whether it is ready, running, waiting, or whatever.

Process privileges

This is required to allow/disallow access to system resources.

Process ID

Unique identification for each of the process in the operating system.

Pointer

A pointer to parent process.

Program Counter

Program Counter is a pointer to the address of the next instruction to be executed for this process.

CPU registers

Various CPU registers where process need to be stored for execution for running state.

CPU Scheduling Information

Process priority and other scheduling information which is required to schedule the process.

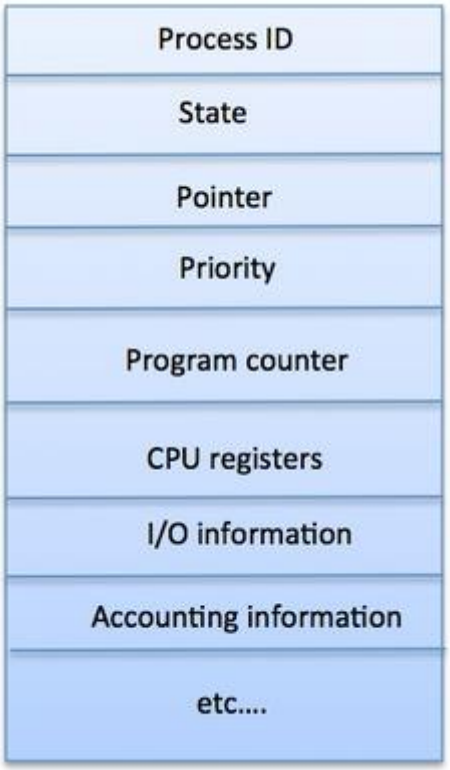
Memory management information

This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.

IO status information

This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

What is Context Switching?

A context switching is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

It is one of the essential features of the multitasking operating system. The processes are switched so fastly that it gives an illusion to the user that all the processes are being executed at the same time.

Context switching can happen due to the following reasons:

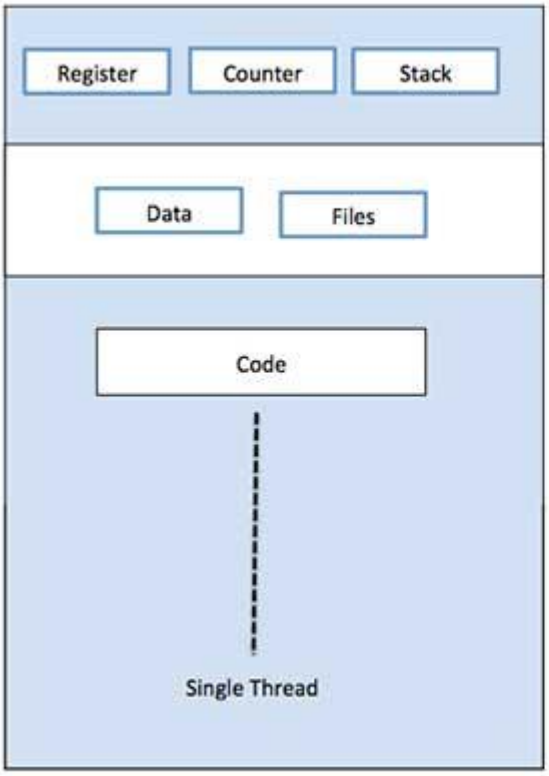
- When a process of high priority comes in the ready state. In this case, the execution of the running process should be stopped and the higher priority process should be given the CPU for execution.
- When an interruption occurs then the process in the running state should be stopped and the CPU should handle the interrupt before doing something else.
- When a transition between the user mode and kernel mode is required then you have to perform the context switching.

What is Thread?

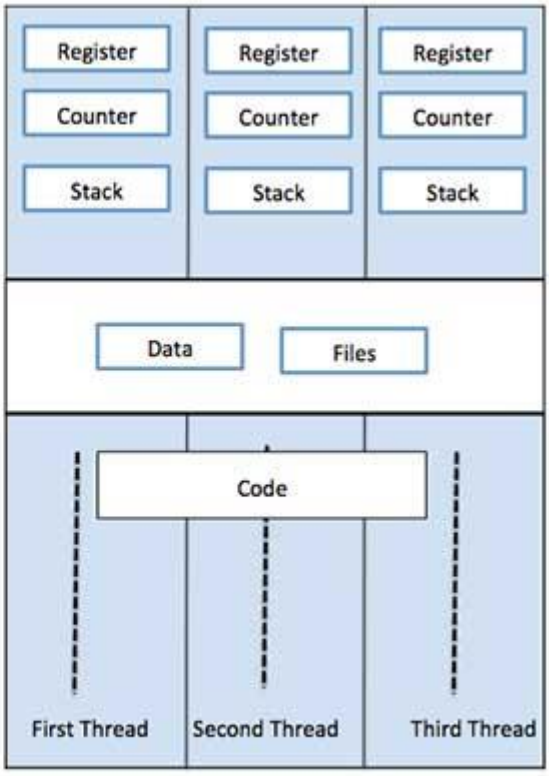
A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history. A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight 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 a separate flow of control. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

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, a 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 Thread

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

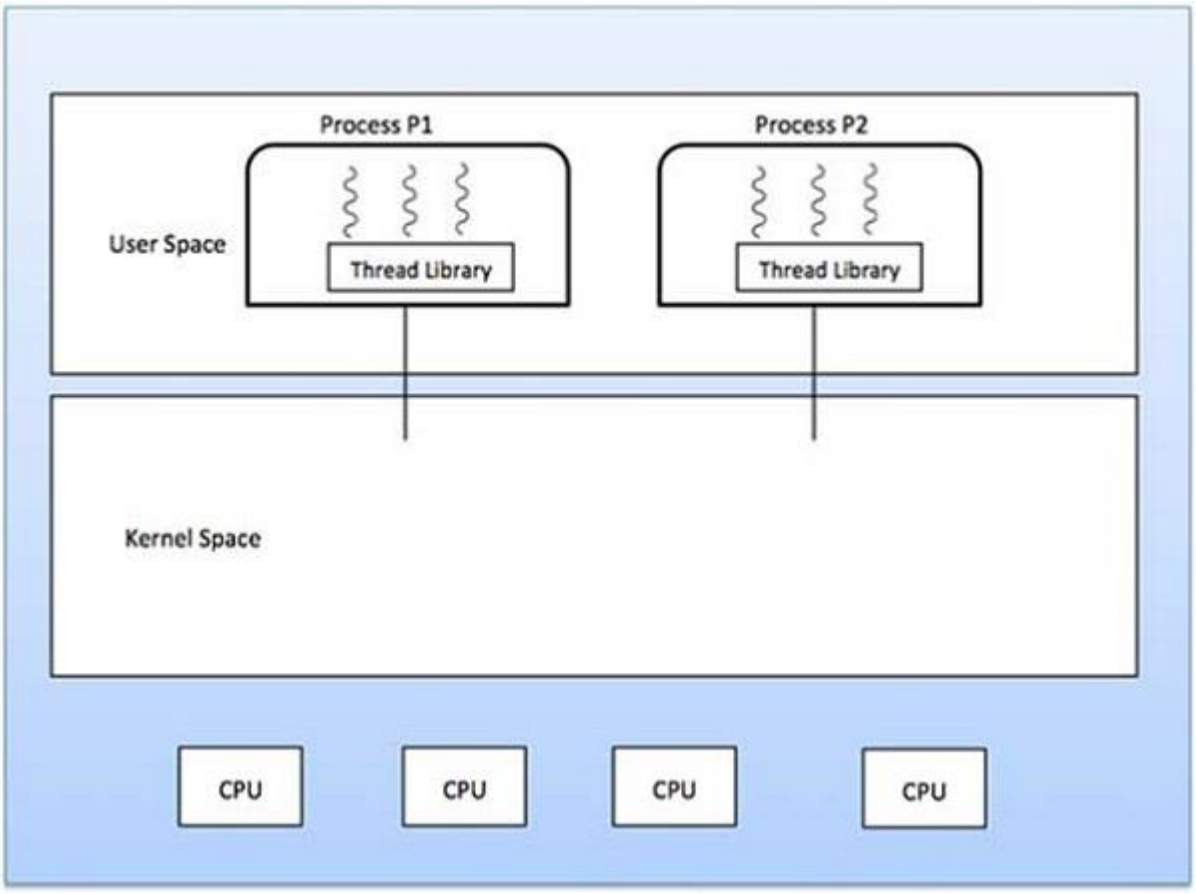
Types of Thread

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, the 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 starts with a single 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 individuals 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 be multithreaded.

Disadvantages

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

Multithreading Models

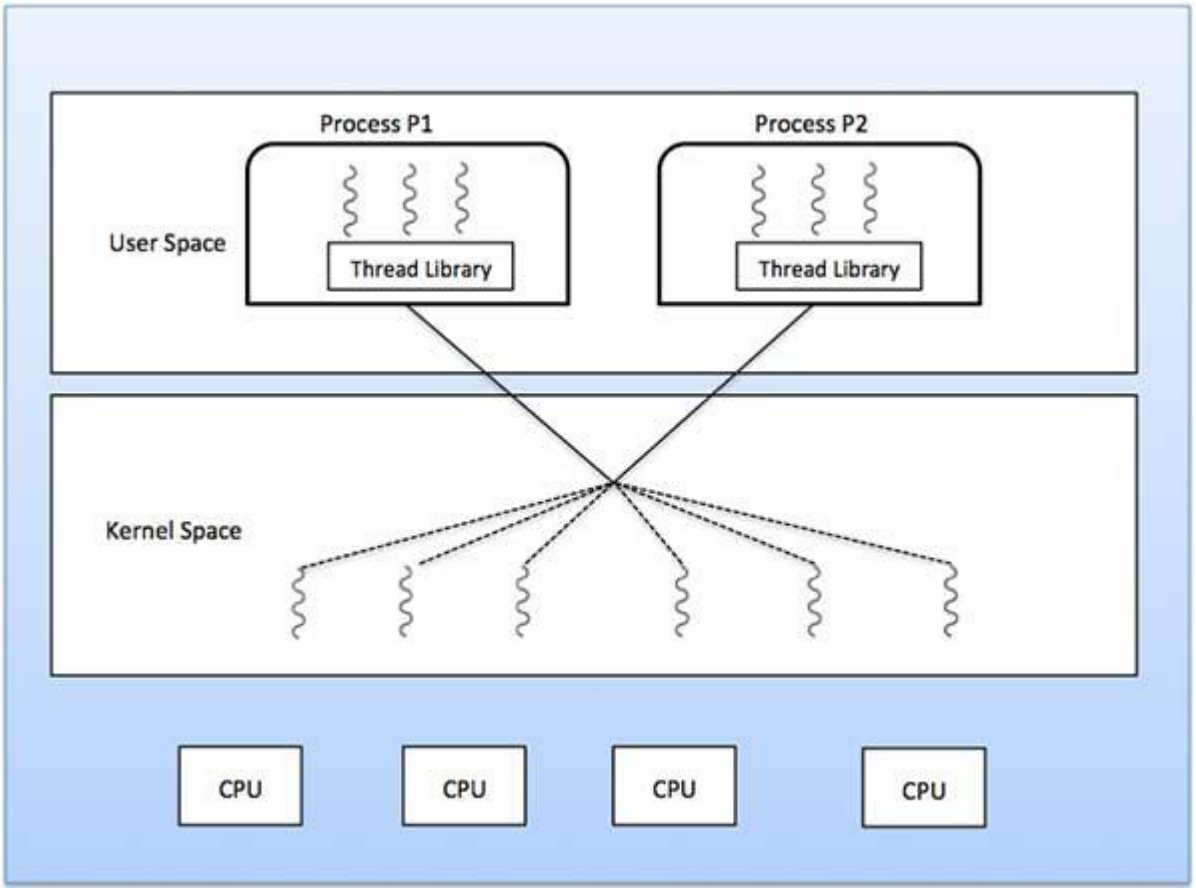
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the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

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

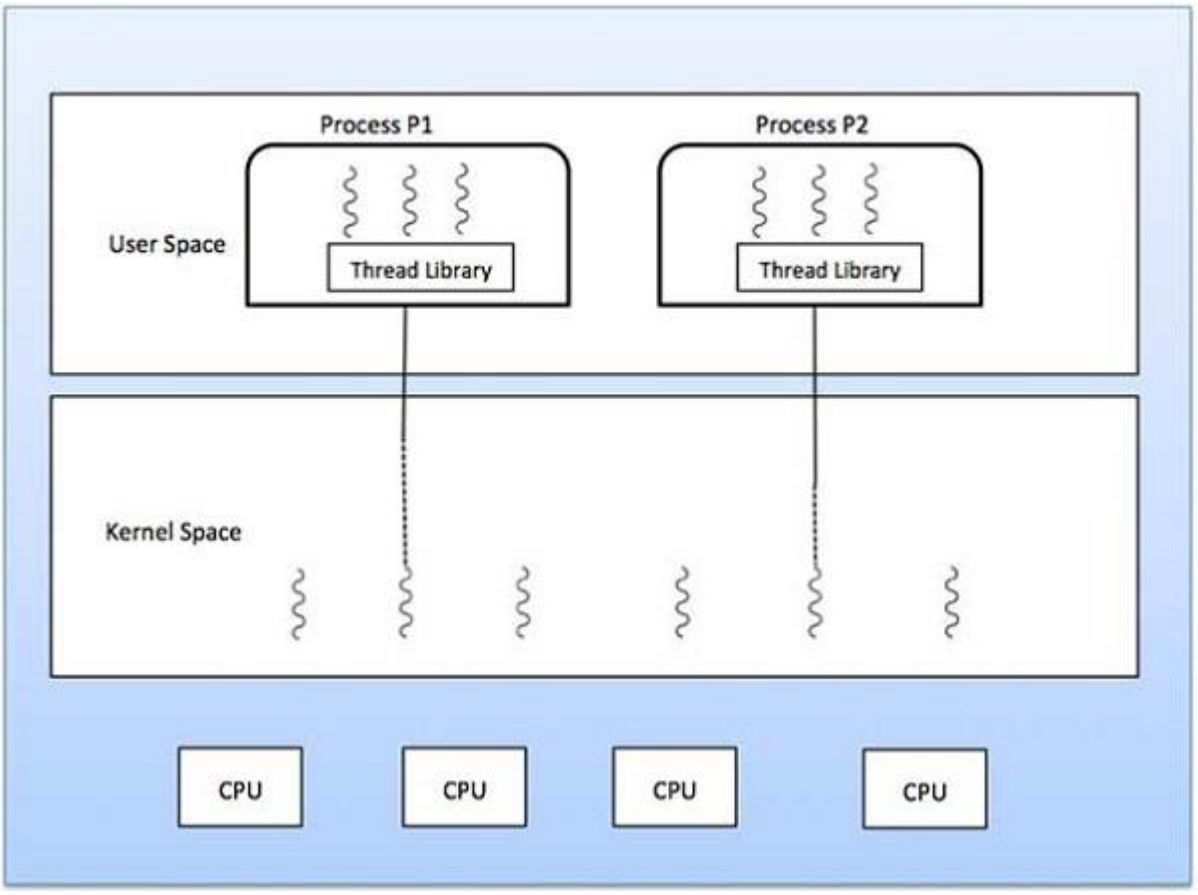
The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



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 by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

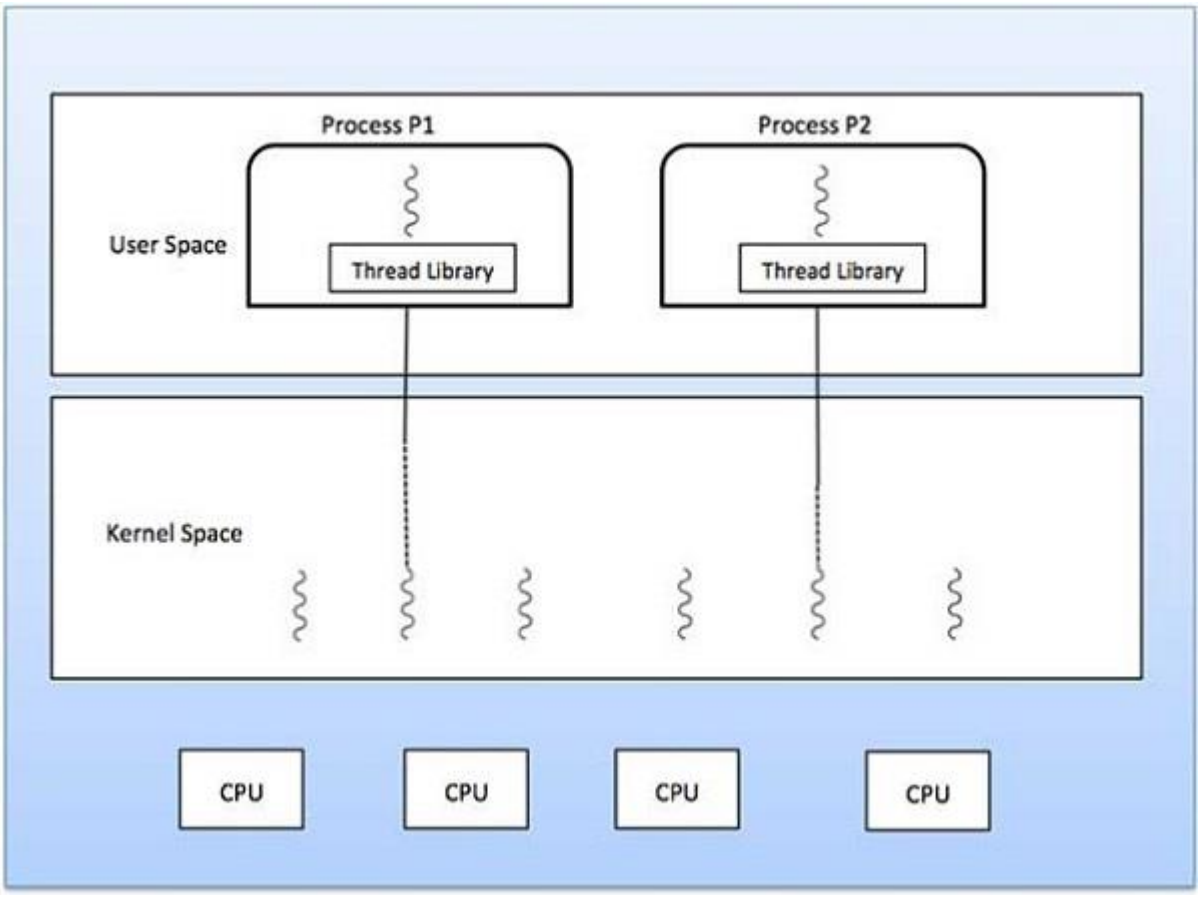
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



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 also 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 applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded

What is CPU Scheduling?

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc,

thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**

Non-Preemptive Scheduling:

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive Scheduling:

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Schedulers

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

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

Long Term Scheduler

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

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

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

Short Term Scheduler

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

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

Medium Term Scheduler

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

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

Comparison among Scheduler

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

Scheduling Criteria

There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

- **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
- **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.
- **Turnaround time** - Time required for a particular process to complete, from submission time to completion. (Wall clock time.)
- **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
- **Response time** - The time taken in an interactive program from the issuance of a command to the commence of a response to that command.

In general one wants to optimize the average value of a criteria (Maximize CPU utilization and throughput, and minimize all the others.) However some times one wants to do something different, such as to minimize the maximum response time.

Scheduling algorithms

There are various process scheduling algorithms which are given by-

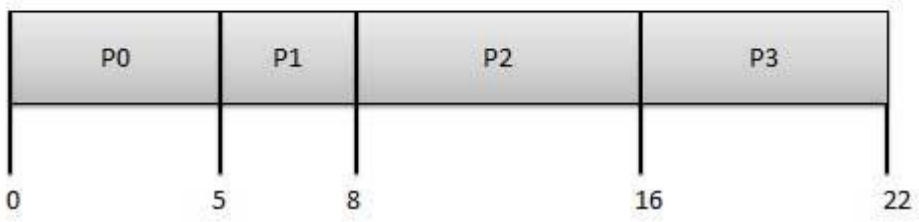
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Priority Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time/ Burst Time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- 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 served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Multiple-processor scheduling

In multiple-processor scheduling multiple CPU's are available and hence Load Sharing becomes possible. However multiple processor scheduling is more complex as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

UNIT 3

Memory Management

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

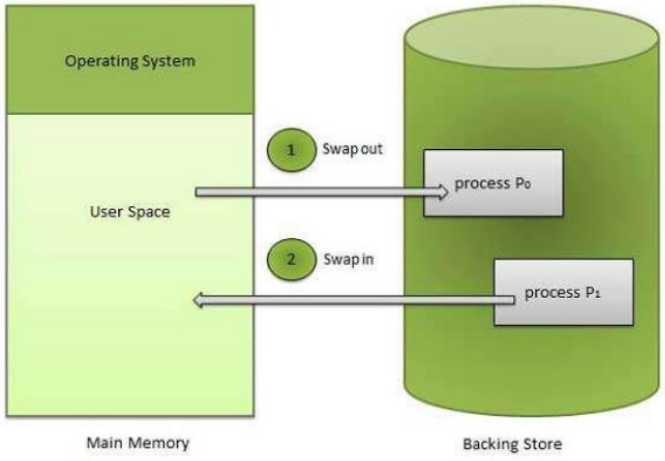
Logical and Physical Address Space

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address. Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space. The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory. Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Memory Allocation

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

1. **First Fit:**
   The first hole that is big enough is allocated to program.
2. **Best Fit:**
   The smallest hole that is big enough is allocated to program.
3. **Worst Fit:**
   The largest hole that is big enough is allocated to program.

Contiguous Memory Allocation:

The main memory is divided into two partitions: at one partition the operating system resides and at other the user processes reside. The Contiguous memory allocation is one of the methods of memory allocation. In contiguous memory allocation, when a process requests for the memory, a single contiguous section of memory blocks is assigned to the process according to its requirement. The contiguous memory allocation can be achieved by dividing the memory into the fixed-sized partition and allocate each partition to a single process only.

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

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

1. **Fixed Partitioning:**
   This is the oldest and simplest technique used to put more than one processes in the main memory. In this partitioning, number of partitions in RAM are fixed but size of each partition may or may not be same. As it is contiguous allocation, hence no spanning is allowed. Here partition are made before execution or during system configure.

Advantages of Fixed Partitioning –

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

Disadvantages of Fixed Partitioning –

i. **Internal Fragmentation:** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process

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

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

External fragmentation: Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used. External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

Internal fragmentation: Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Compaction

Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes. Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time. In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.

Non-Contiguous Memory Allocation

The Non-contiguous memory allocation allows a process to acquire the several memory blocks at the different location in the memory according to its requirement**.** The non-contiguous memory allocation reduces the memory wastage caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.

Comparison Chart

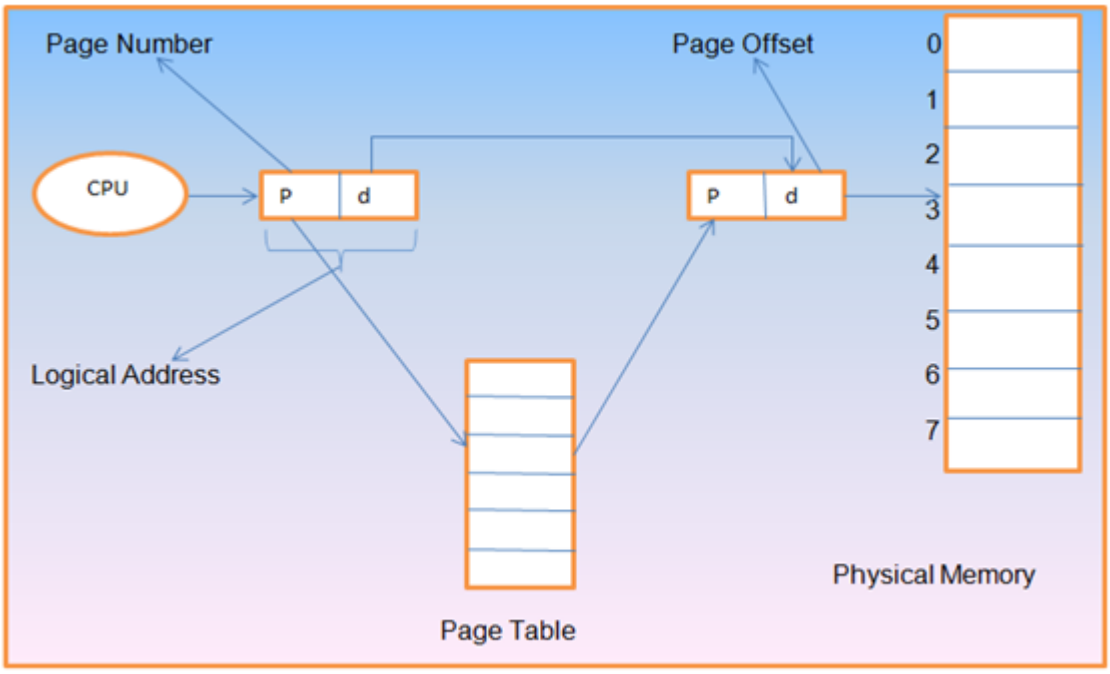
Basis the Comparison	Contiguous Memory Allocation	Non-contiguous Memory Allocation
Basic	Allocates consecutive blocks of memory to a process.	Allocates separate blocks of memory to a process.
Overheads	Contiguous memory allocation does not have the overhead of	Non-contiguous memory allocation has overhead of address translation while execution of a process.

	address translation while execution of a process.	
Execution rate	A process executes faster in contiguous memory allocation	A process executes quite slower comparatively in non-contiguous memory allocation.
Solution	The memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only.	Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available.

Paging with hardware implementation of page table.

Paging is a memory-management technique that provides the non contiguous address space in main memory. Paging avoids external fragmentation. In this technique physical memory is broken into fixed-sized blocks called frames and logical memory is divided into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from the backing-store.

The address generated by CPU is called logical address and it is divided into two parts a page number (p) and a page offset (d). The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. The combination of base address and page offset is used to map the page in physical memory address. Hardware decides the page size.



What is Paging Protection?

The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

Advantages of Paging

Here, are advantages of using Paging method:

- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.

Disadvantages of Paging

Here, are drawback/ cons of Paging:

- May cause Internal fragmentation
- Complex memory management algorithm.
- Page tables consume additional memory.
- Multi-level paging may lead to memory reference overhead.

Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory. Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size. A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

Advantages of a Segmentation

Here, are pros/benefits of Segmentation

- Offer protection within the segments
- You can achieve sharing by segments referencing multiple processes.
- Not offers internal fragmentation
- Segment tables use lesser memory than paging

Disadvantages of Segmentation

Here are cons/drawback of Segmentation

- In segmentation method, processes are loaded/ removed from the main memory. Therefore, the free memory space is separated into small pieces which may create a problem of external fragmentation
- Costly memory management algorithm

Segmented Paging

Pure segmentation is not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques. In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number → It points to the appropriate Segment Number.

Page Number → It Points to the exact page within the segment

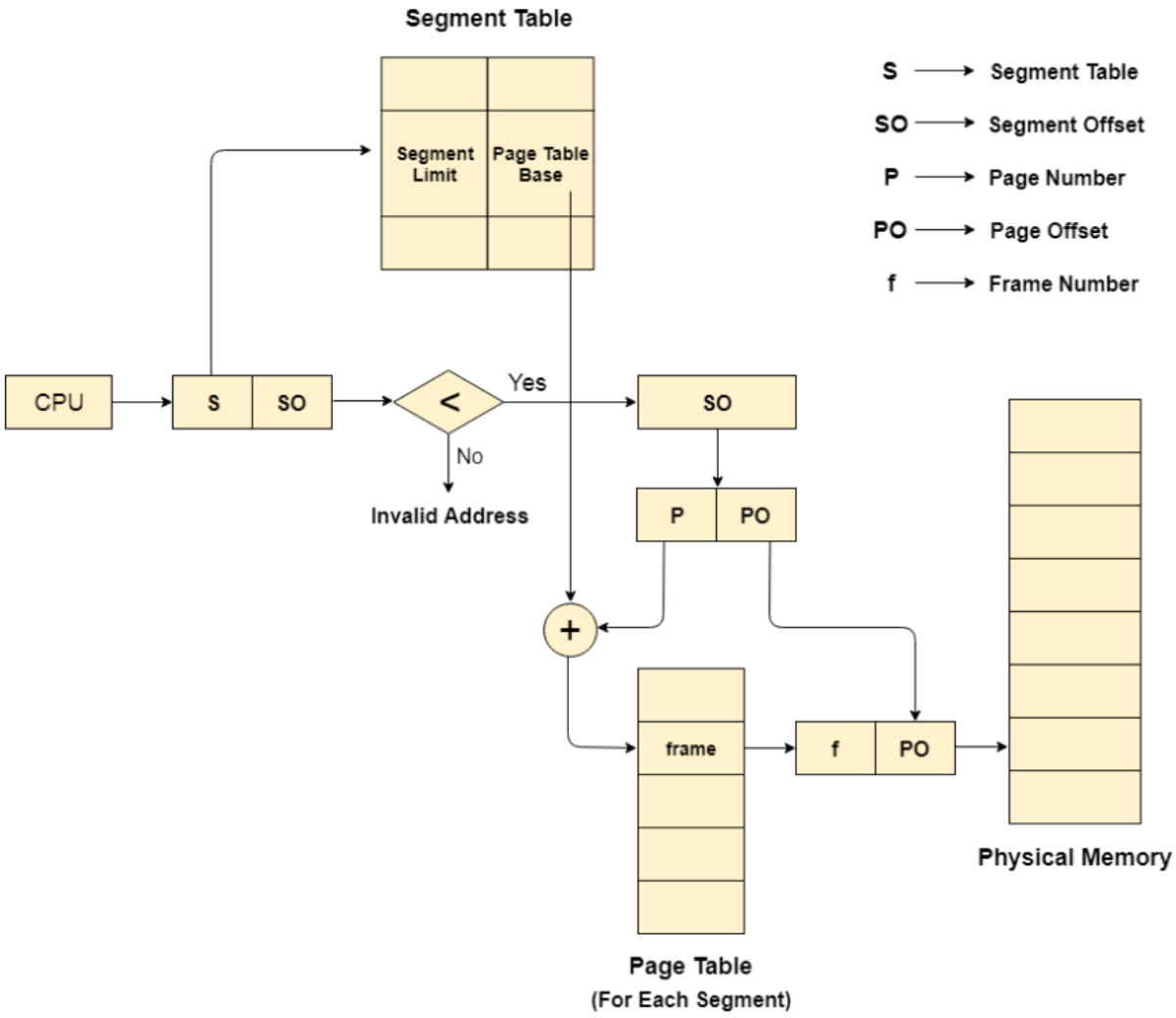
Page Offset → Used as an offset within the page frame

Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.

Translation of logical address to physical address

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.


Segment Table
Segment Limit
Page Table Base
S → Segment Table
SO → Segment Offset
P → Page Number
PO → Page Offset
f → Frame Number
CPU
S
SO
<
Yes
No
Invalid Address
SO
P
PO
+
frame
f
PO
Physical Memory
Page Table
(For Each Segment)

Virtual Memory

Virtual Memory is a memory management scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process. Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory. By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works? In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory. Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

Demand Paging

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory. A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced.

Page Replacement Algorithms

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. **Optimal Page Replacement algorithm** → this algorithms replaces the page which will not be referred for so long in future. Although it cannot be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. **Least recent used (LRU) page replacement algorithm** → this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

Numerical on Optimal, LRU and FIFO

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm
2. FIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm

Optimal Page Replacement Algorithm

Request	**4**	**7**	**6**	**1**	**7**	**6**	**1**	**2**	**7**	**2**
Frame3			6	6	6	6	6	2	2	2
Frame2		7	7	7	7	7	7	7	7	7
Frame1	4	4	4	1	1	1	1	1	1	1
Hit/Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU Page Replacement Algorithm

Request	**4**	**7**	**6**	**1**	**7**	**6**	**1**	**2**	**7**	**2**
Frame3			6	6	6	6	6	6	7	7
Frame2		7	7	7	7	7	7	2	2	2
Frame1	4	4	4	1	1	1	1	1	1	1
Hit/Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame3			6	6	6	6	6	6	7	7
Frame2		7	7	7	7	7	7	2	2	2
Frame1	4	4	4	1	1	1	1	1	1	1
Hit/Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

Belady's Anomaly

In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames. This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady'sAnomaly.

Paging VS Segmentation

Sr No.	Paging	Segmentation
1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
5	Paging is closer to Operating System	Segmentation is closer to User
6	It suffers from internal fragmentation	It suffers from external fragmentation
7	There is no external fragmentation	There is no external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information
10	Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.

Q1. Explain Demand Segmentation.

Ans. Operating system uses demand segmentation where there is insufficient hardware available to implement 'Demand Paging'. The segment table has a valid bit to specify if the segment is already in physical memory or not. If a segment is not in physical memory then segment fault results, which traps the operating system and brings the needed segment into physical memory, much like a page fault.

Demand segmentation allows for pages that are often referenced with each other to be brought into memory together, this decreases the number of page faults.

UNIT 4

Process synchronization & Deadlocks

Introduction

When two or more process cooperates with each other, their order of execution must be preserved otherwise there can be conflicts in their execution and inappropriate outputs can be produced. A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process. Such processes need to be synchronized so that their order of execution can be guaranteed. The procedure involved in preserving the appropriate order of execution of cooperative processes is known as Process Synchronization. There are various synchronization mechanisms that are used to synchronize the processes.

Race Condition

A Race Condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.

Critical Section

The regions of a program that try to access shared resources and may cause race conditions are called critical section. To avoid race condition among the processes, we need to assure that only one process at a time can execute within the critical section.

The Critical Section Problem

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any I/O device. The critical section cannot be executed by more than one process at the same time. The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise. The critical section problem can be solved it the following conditions can be satisfied.

- **Mutual Exclusion:** By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.

- **Progress:** Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.
- **Bounded Waiting**: We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.
- **Architectural Neutrality:** Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Introduction to semaphore

Dijkstra proposed an approach which involves storing all the wake-up calls. Dijkstra states that, instead of giving the wake-up calls directly to the consumer, producer can store the wake-up call in a variable. Any of the consumers can read it whenever it needs to do so.

Semaphore is the variables which stores the entire wake up calls that are being transferred from producer to consumer. It is a variable on which read, modify and update happens automatically in kernel mode.

Semaphore cannot be implemented in the user mode because race condition may always arise when two or more processes try to access the variable simultaneously. It always needs support from the operating system to be implemented.

Semaphore can be divided into two categories.

1. Counting Semaphore
2. Binary Semaphore or Mutex

Counting Semaphore

Counting semaphore can be used when we need to have more than one process in the critical section at the same time. The value of counting semaphore at any point of time indicates the maximum number of processes that can enter in the critical section at the same time.

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

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

Problem: A Counting Semaphore was initialized to 12,then 10P (wait) and 4V (Signal) operations were computed on this semaphore. What is the result?

Solution:

1. S = 12 (initial)
2. 10 p (wait) :
3. SS = S -10 = 12 - 10 = 2
4. then 4 V :
5. SS = S + 4 =2 + 4 = 6

Hence, the final value of counting semaphore is 6.

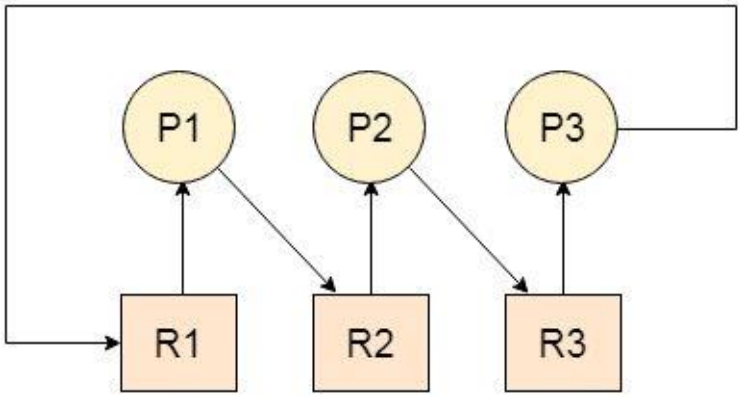
Binary Semaphore or Mutex

In counting semaphore, Mutual exclusion was not provided. However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slots available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

Introduction to Deadlock

Every process needs some resources to complete its execution. A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

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



After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution. In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.

Difference between Starvation and Deadlock

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.

4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Necessary conditions for Deadlocks

1. **Mutual Exclusion**

   A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. **Hold and Wait**

   A process waits for some resources while holding another resource at the same time.

3. **No preemption**

   The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait**

   All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

Strategies for handling Deadlock

1. Deadlock Ignorance

Deadlock Ignorance is the most widely used approach among all the mechanism. This is being used by many operating systems mainly for end user uses. In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff.

There is always a tradeoff between Correctness and performance. The operating systems like Windows and Linux mainly focus upon performance. However, the performance of the system decreases if it uses deadlock handling mechanism all the time if deadlock happens 1 out of 100 times then it is completely unnecessary to use the deadlock handling mechanism all the time.

In these types of systems, the user has to simply restart the computer in the case of deadlock. Windows and Linux are mainly using this approach

2. Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

1. Mutual Exclusion: If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource. However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

2. Hold and Wait: Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

3. No Preemption: Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

4. Circular Wait: To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

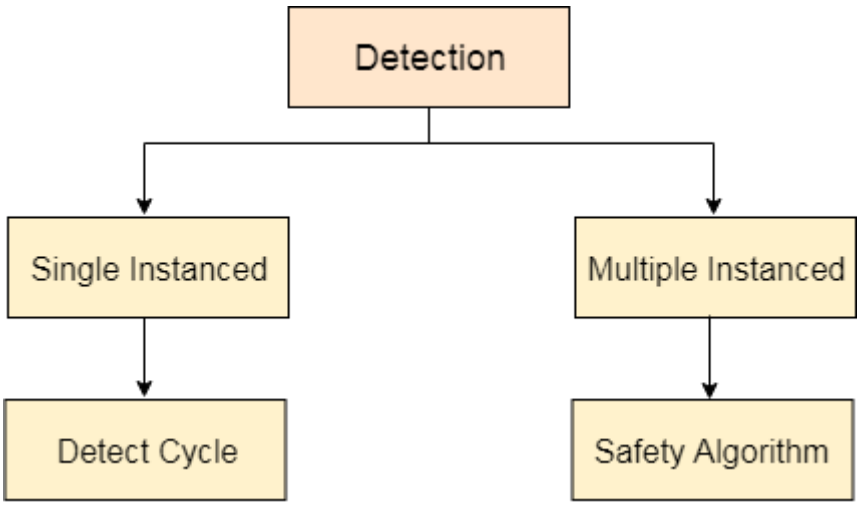
3. Deadlock avoidance

In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step. In simple words, The OS reviews each allocation so that the allocation doesn't cause the deadlock in the system.

4. Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix. In order to recover the system from deadlocks, either OS considers resources or processes.

For Resource

Preempt the **resource** we can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state System passes through various states to get into the deadlock state. The operating system can roll back the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

For Process

Kill a process: Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process: This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

I/O Hardware

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc.

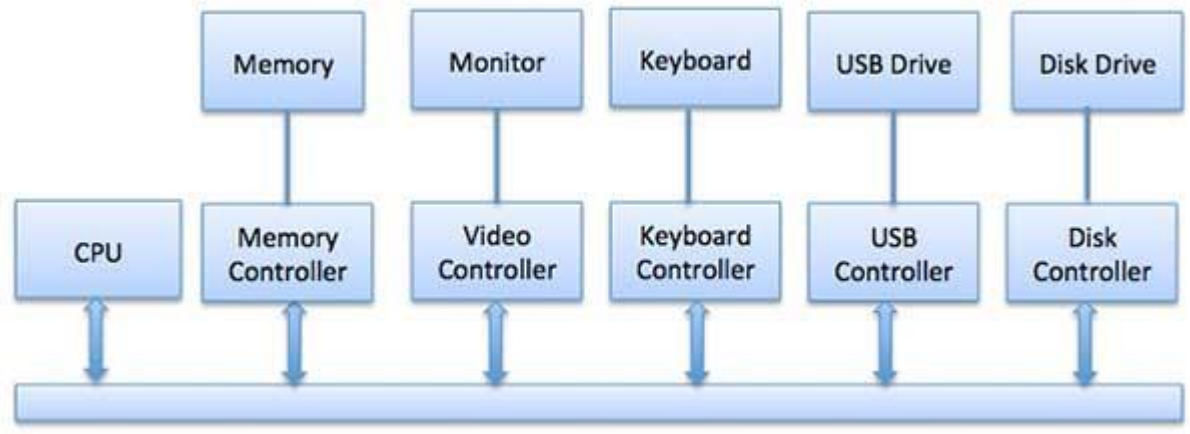
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

- **Synchronous I/O** − In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** − I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

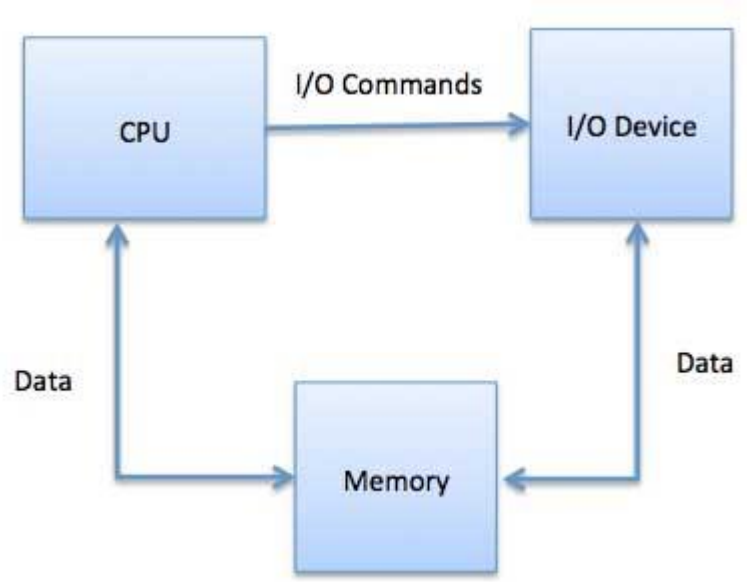
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

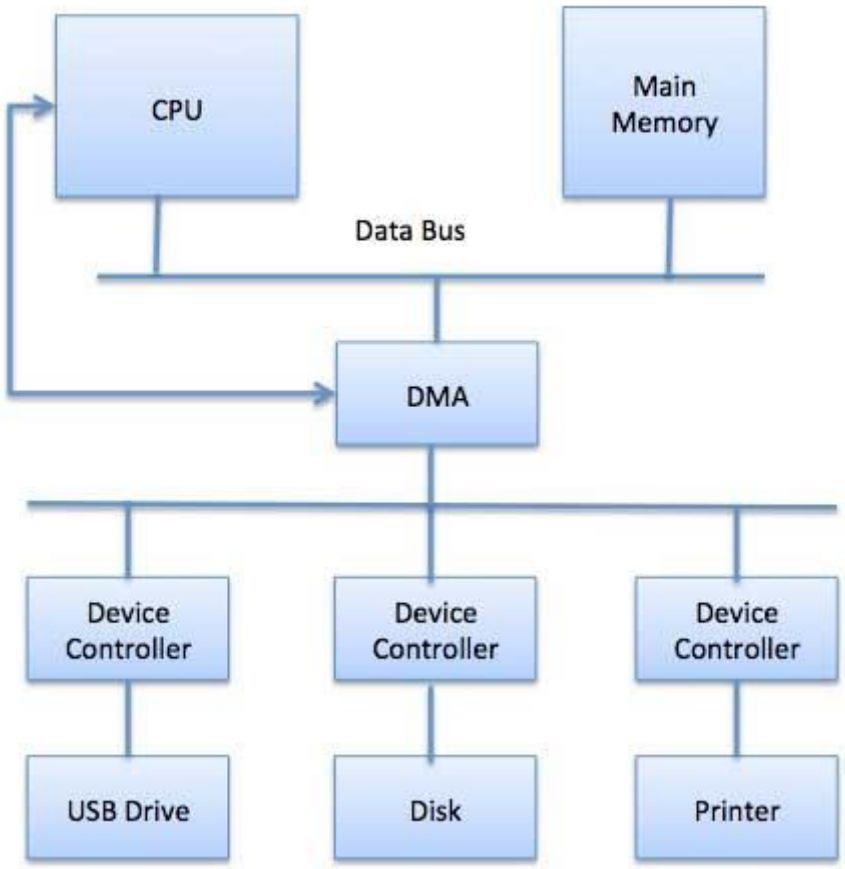
Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main

memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows −

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.

6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as polling and interrupts. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, it saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

Application I/O Interface

- I/O system calls encapsulate device behaviours in generic classes.
- Device-driver layer hides differences among I/O controllers from kernel.
- New devices talking already-implemented protocols need no extra work.
- Each OS has its own I/O subsystem structures and device driver frameworks.
- Devices vary in many dimensions.

  i) Character-stream or block.
  ii) Sequential or random-access.
  iii) Synchronous or asynchronous (or both)
  iv) Sharable or dedicated
  v) Speed of operation.
  vi) read-write, read only, or write only.

I/O Protection

- The I/O system must protect against either accidental or deliberate erroneous I/O.
- User applications are not allowed to perform I/O in user mode - All I/O requests are handled through system calls that must be performed in kernel mode.
- Memory mapped areas and I/O ports must be protected by the memory management system, but access to these areas cannot be totally denied to user programs. Instead the memory protection system restricts access so that only one process at a time can access particular parts of memory, such as the portion of the screen memory corresponding to a particular window.

Transforming I/O Requests to Hardware Operations

- Users request data using file names, which must ultimately be mapped to specific blocks of data from a specific device managed by a specific device driver.
- DOS uses the colon separator to specify a particular device (e.g. C:, LPT:, etc.)
- UNIX uses a ***mount table*** to map filename prefixes (e.g. /usr) to specific mounted devices. Where multiple entries in the mount table match different prefixes of the filename the one that matches the longest prefix is chosen. (e.g. /usr/home instead of /usr where both exist in the mount table and both match the desired file.)
- UNIX uses special ***device files,*** usually located in /dev, to represent and access physical devices directly.
    - Each device file has a major and minor number associated with it, stored and displayed where the file size would normally go.
    - The major number is an index into a table of device drivers, and indicates which device driver handles this device. (E.g. the disk drive handler.)
    - The minor number is a parameter passed to the device driver, and indicates which specific device is to be accessed, out of the many which may be handled by a particular device driver. (e.g. a particular disk drive or partition.)
- A series of lookup tables and mappings makes the access of different devices flexible, and somewhat transparent to users.

Kernel I/O Subsystem

Kernels provide many services related to I/O. Several services—scheduling, buffering, caching, spooling, device reservation, and error handling'—are provided by the kernel's I /O subsystem and build on the hardware and device driver infrastructure. The I/O subsystem is also responsible for protecting itself from errant processes and malicious users.

I/O Scheduling: To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which applications issue system calls rarely is the best choice. Scheduling can improve overall system performance, can share device access fairly among processes, and can reduce the average waiting time.

Buffering: A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons. One reason is to cope with a speed mismatch between the producer and consumer of a data stream. A second use of buffering is to adapt between devices that have different data-transfer sizes. Such disparities are especially common in computer networking, where buffers are used widely for fragmentation and reassembly of messages. At the sending side, a large message is fragmented into small network packets. The packets are sent over the network, and the receiving side places

them in a reassembly buffer to form an image of the source data. A third use of buffering is to support copy semantics for application I/O.

Caching: A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere.

Spooling and Device Reservation: A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In others, it is handled by an in-kernel thread.
Another way to deal with concurrent device access is to provide explicit facilities for coordination. Some operating systems (including VMS) provide support for exclusive device access by enabling a process to allocate an idle device and to deallocate that device when it is no longer needed.

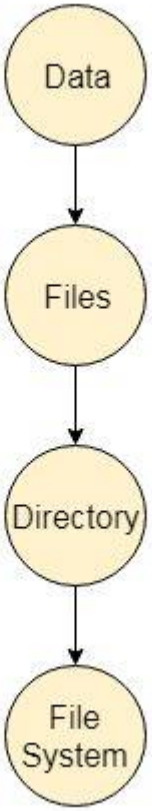
Error Handling: An operating system that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual result of each minor mechanical glitch. Devices and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded, or for "permanent" reasons, as when a disk controller becomes defective. Operating systems can often compensate effectively for transient failures.

File Management

What is a File.

A file can be defined as a data structure which stores the sequence of records. Files are stored in a file system, which may exist on a disk or in the main memory. Files can be simple (plain text) or complex (specially-formatted).

The collection of files is known as Directory. The collection of directories at the different levels, is known as File System.



Attributes of the File

1. Name

Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

2. Identifier

Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** a video file can have the extension **.mp4.**

3. Type

In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.

4. Location

In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

5. Size

The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.

6. Protection

The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.

7. Time and Date

Every file carries a time stamp which contains the time and date on which the file is last modified.

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

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

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

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 access
- 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 by 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.

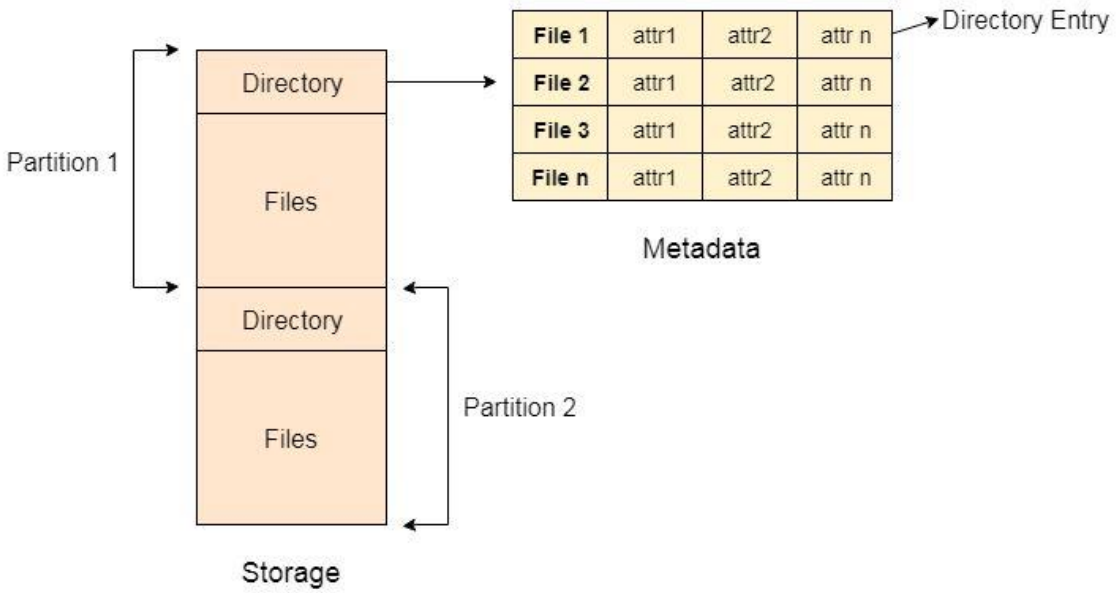
Indexed sequential access

- This mechanism is built up on base 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.

Directory Structure

What is a directory?

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes. To get the benefit of different file systems on the different operating systems, a hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks. Each partition must have at least one directory in which, all the files of the partition can be listed.

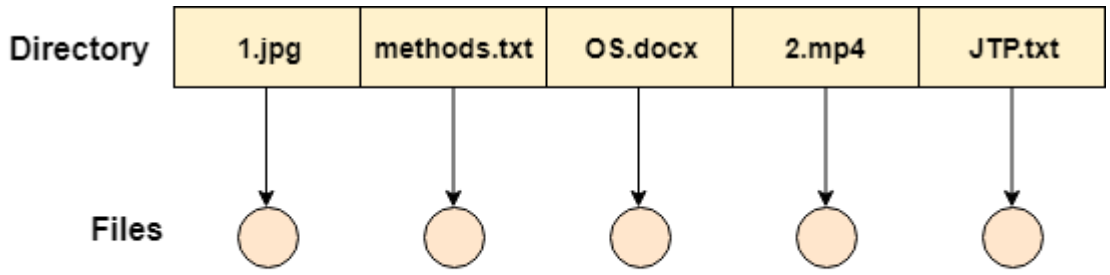


A directory can be viewed as a file which contains the Meta data of the bunch of files. Every Directory supports a number of common operations on the file:

1. File Creation
2. Search for the file
3. File deletion
4. Renaming the file
5. Traversing Files
6. Listing of files

Single Level Directory

The simplest method is to have one big list of all the files on the disk. The entire system will contain only one directory which is supposed to mention all the files present in the file system. The directory contains one entry per each file present on the file system.This type of directories can be used for a simple system.



Single Level Directory

Advantages

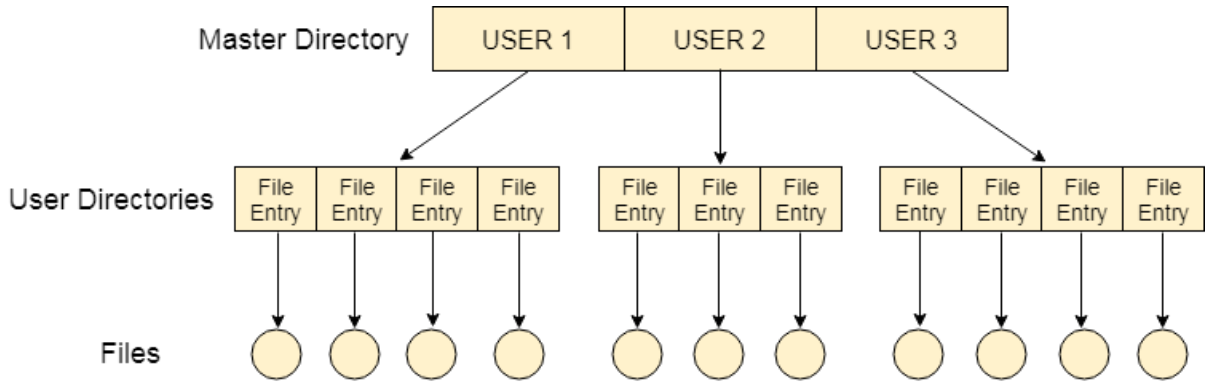
1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.
5. Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

Two Level Directory

In two level directory systems, we can create a separate directory for each user. There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file. The system doesn't let a user to enter in the other user's directory without permission.



Two Level Directory

Characteristics of two level directory system

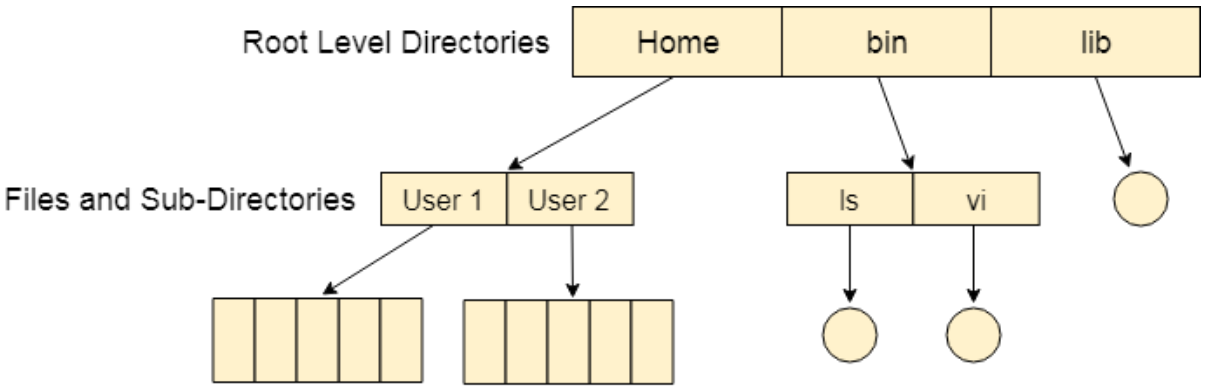
1. Each files has a path name as ***/User-name/directory-name/***
2. Different users can have the same file name.
3. Searching becomes more efficient as only one user's list needs to be traversed.
4. The same kind of files cannot be grouped into a single directory for a particular user.

Tree Structured Directory

In Tree structured directory system, any directory entry can either be a file or sub directory. Tree structured directory system overcomes the drawbacks of two level directory system. The similar kind of files can now be grouped in one directory.

Each user has its own directory and it cannot enter in the other user's directory. However, the user has the permission to read the root's data but he cannot write or modify this. Only administrator of the system has the complete access of root directory.

Searching is more efficient in this directory structure. The concept of current working directory is used. A file can be accessed by two types of path, either relative or absolute.



The Structured Directory System

Allocation Methods

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed. There are following methods which can be used for allocation.

- Contiguous Allocation.
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

Advantages

1. It is simple to implement.
2. We will get Excellent read performance.

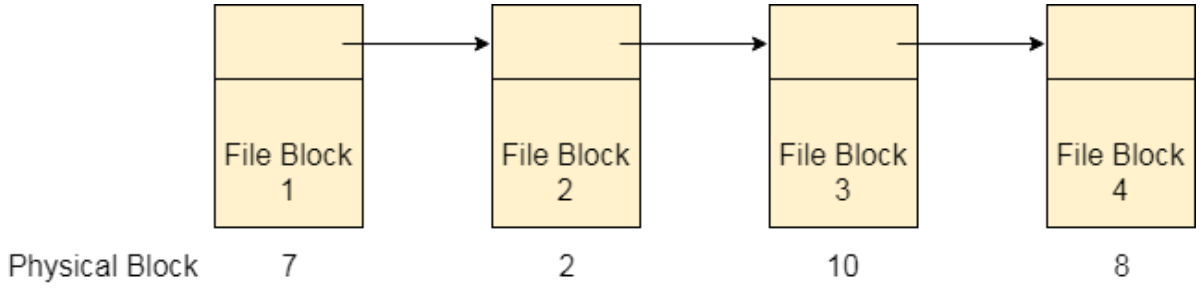
3. Supports Random Access into files.

Disadvantages

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

Linked List Allocation

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



Linked List Allocation

Advantages

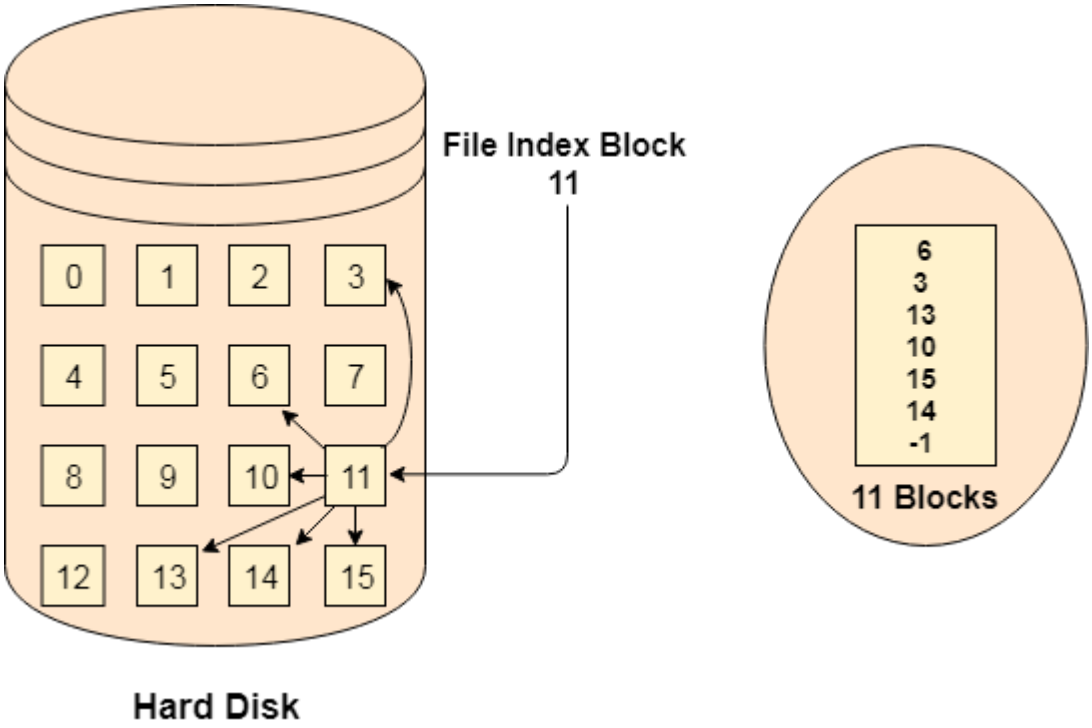
1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

Disadvantages

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block

Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



Advantages

1. Supports direct access
2. A bad data block causes the lost of only that block.

Disadvantages

1. A bad index block could cause the lost of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.
4. More pointer overhead.

Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

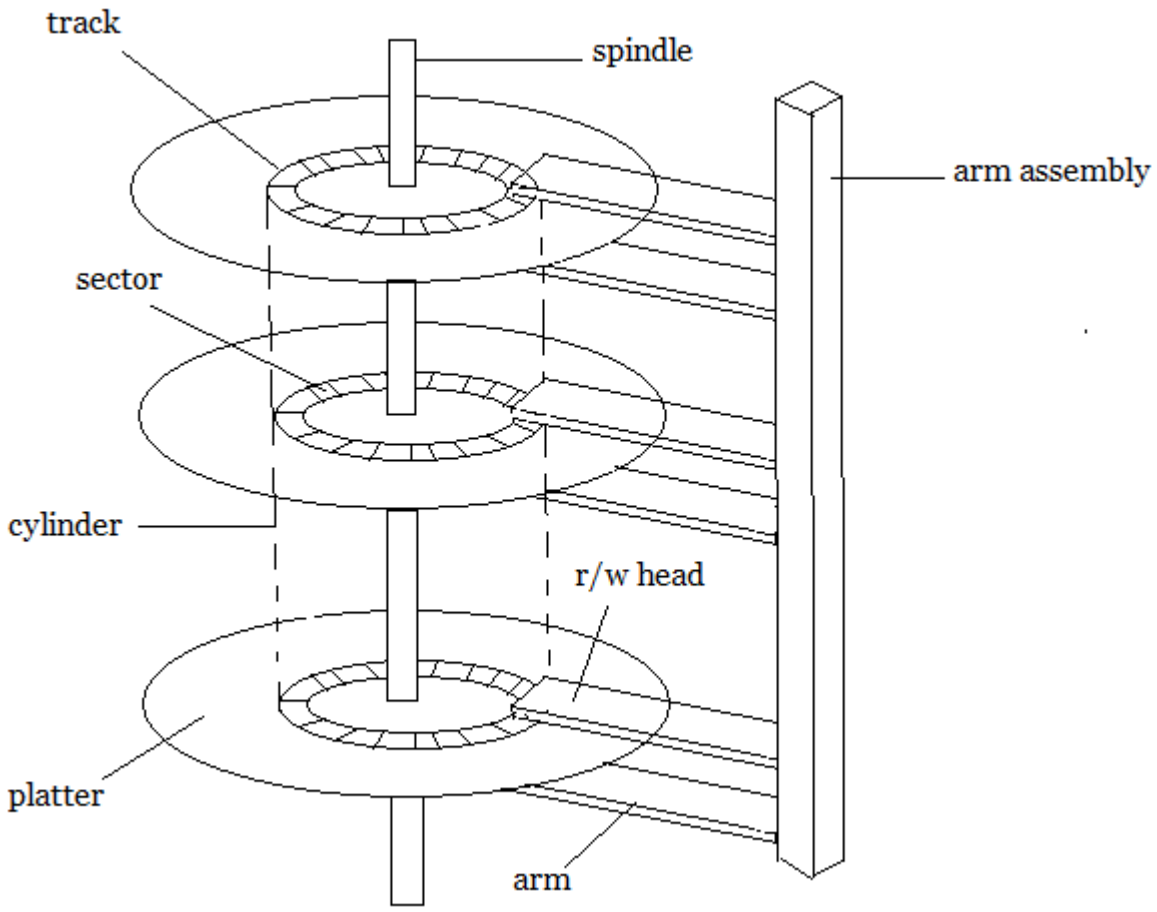


Fig. Structure of a magnetic disk

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track.

Rotational latency is defined as the time taken by the arm to reach the required sector in the track.

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- FCFS scheduling algorithm

- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

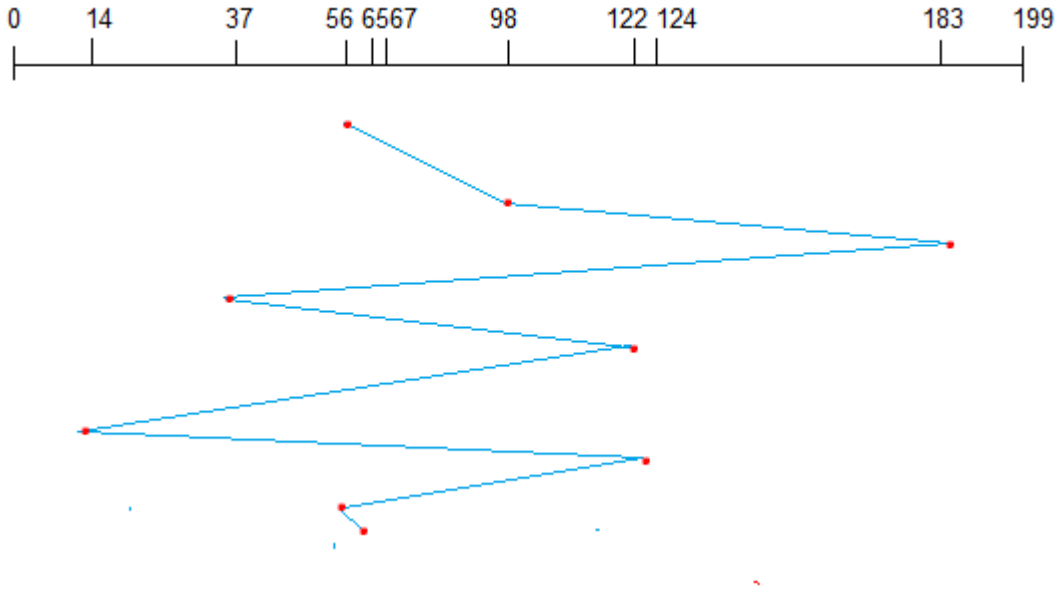
Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

An example of FCFS where the queue has the following requests with cylinder numbers as follows:

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in the given order in the queue i.e., **56→98→183→...→67**.



SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total

seek time as compared to FCFS. It allows the head to move to the closest track in the service queue.

Disadvantages

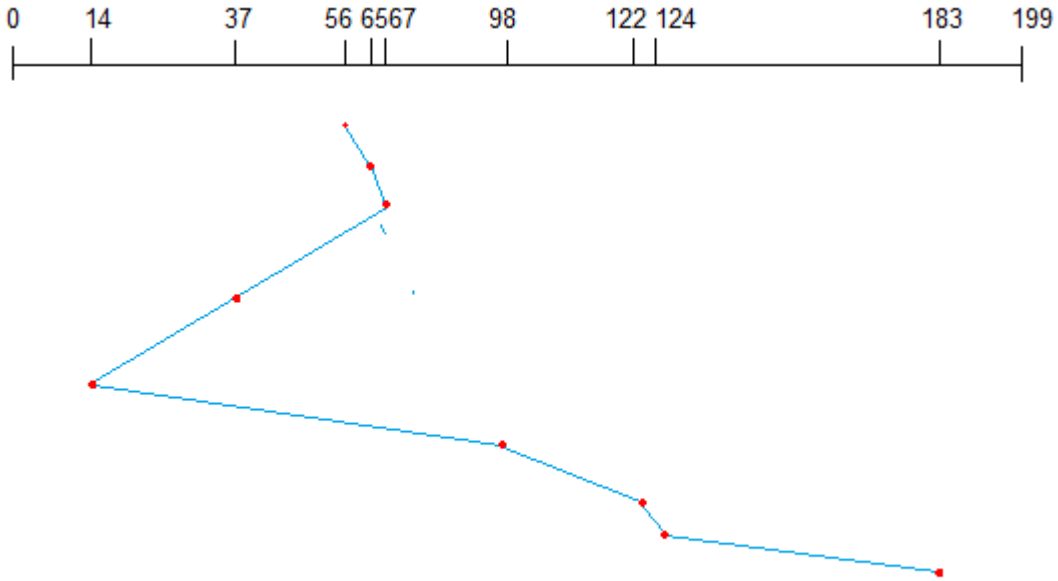
- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

Example

Consider the following disk request

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The next closest cylinder to **56** is **65**, and then the next nearest one is **67**, then **37**, **14**, so on.



Scan Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path.

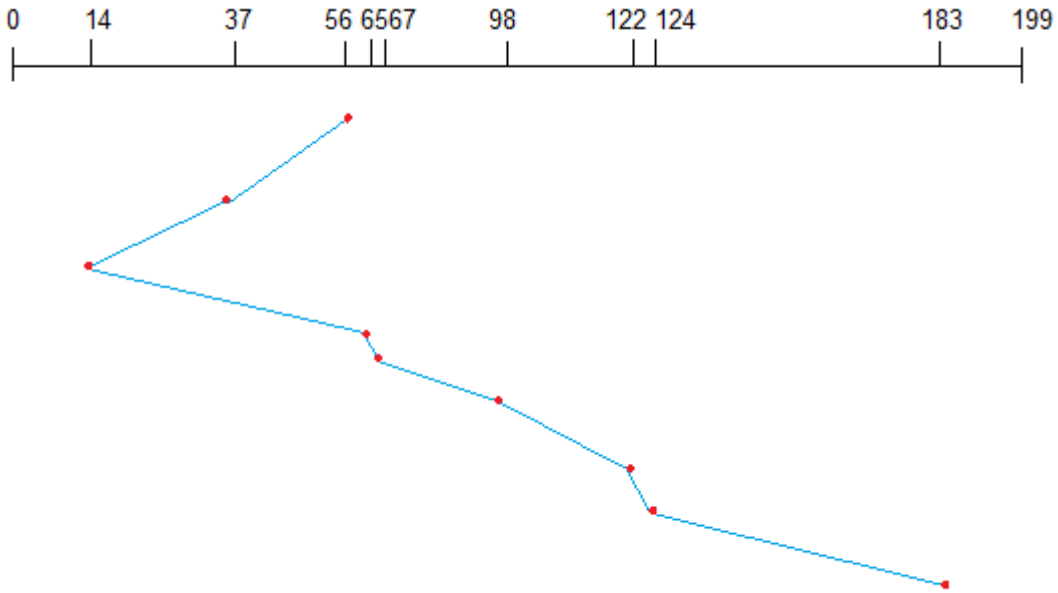
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

Consider the following disk request sequence

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in backward direction and accesses **37** and **14**. Then it goes in the opposite direction and accesses the cylinders as they come in the path.



C-SCAN algorithm

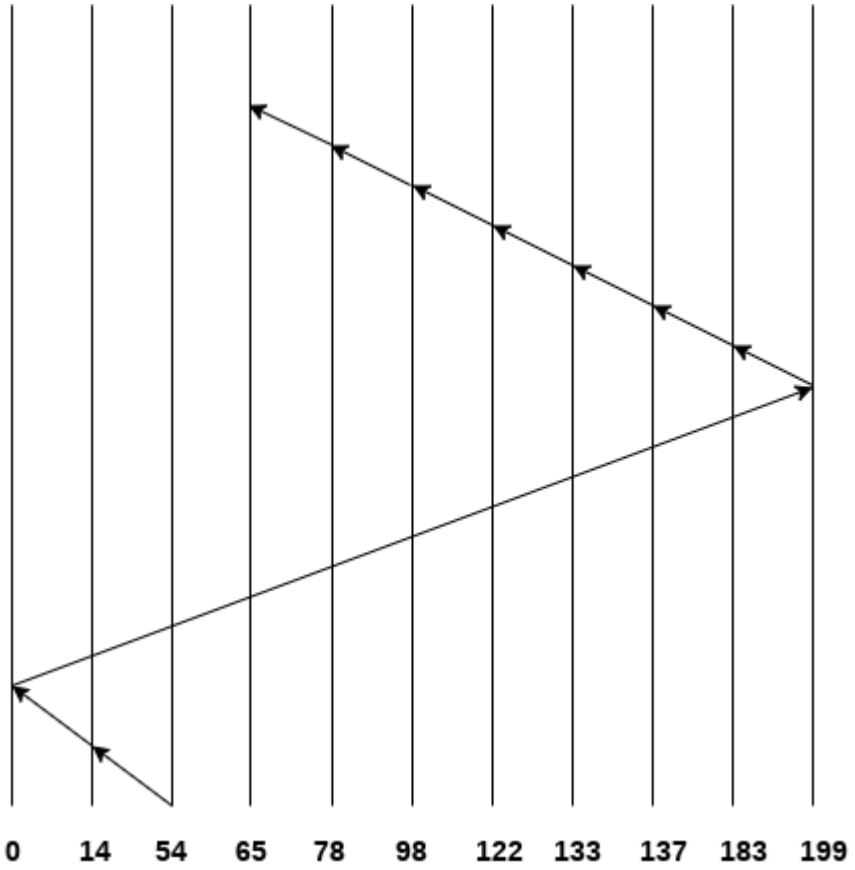
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

Look Scheduling

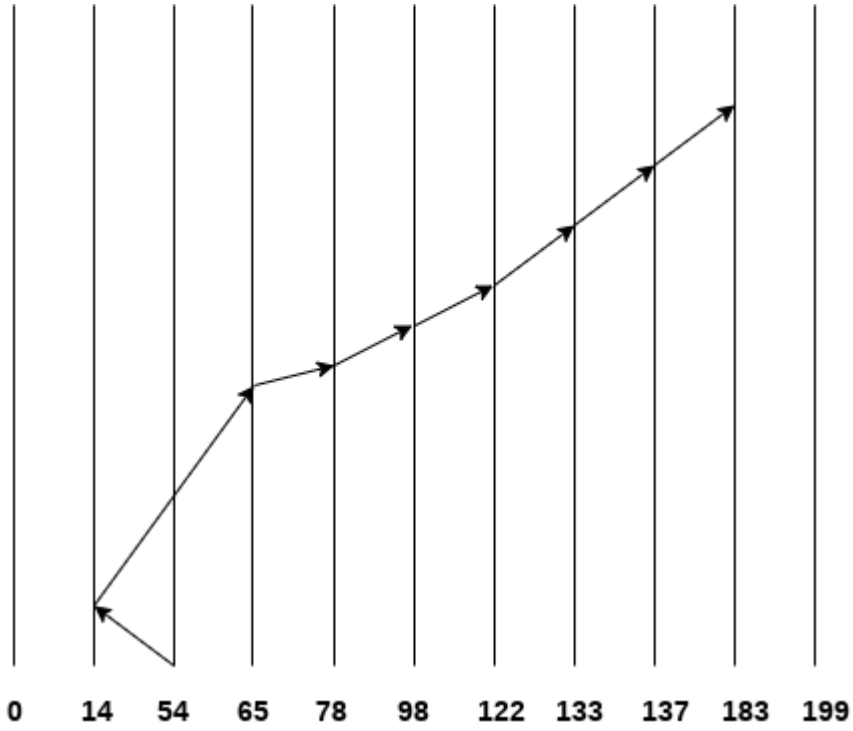
It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

C Look Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

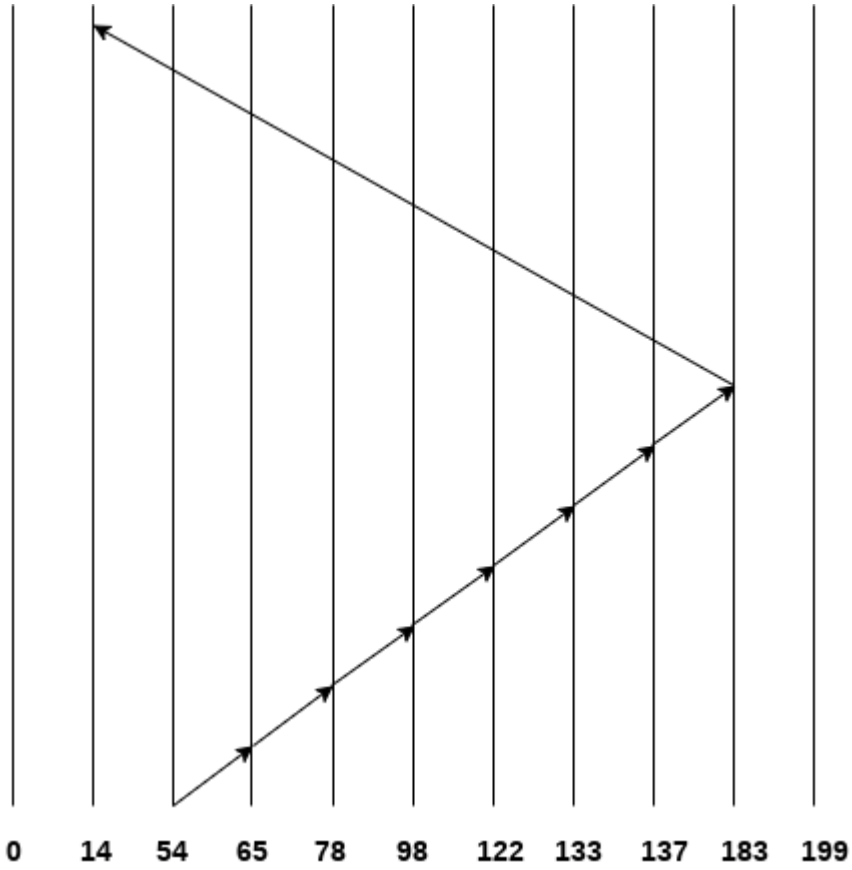
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

For any confusion visit https://www.youtube.com/watch?v=aKmuGwHj3Cw

Introduction to distributed file system (DFS)

A distributed file system (DFS) is a file system with data stored on a server. The data is accessed and processed as if it was stored on the local client machine. The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way. The server allows the client users to share files and store data just like they are storing the information locally. However, the servers have full control over the data and give access control to the clients.

One process involved in implementing the DFS is giving access control and storage management controls to the client system in a centralized way, managed by the servers. Transparency is one of the core processes in DFS, so files are accessed, stored, and managed on the local client machines while the process itself is actually held on the servers. This transparency brings convenience to the end user on a client machine because the network file system efficiently manages all the processes. Generally, a DFS is used in a LAN, but it can be used in a WAN or over the Internet. A DFS allows efficient and well-managed data and storage sharing options on a network compared to other options. Another option for users in network-based computing is a shared disk file system. A shared disk file system puts the access control on the client's systems so the data is inaccessible when the client system goes offline. DFS is fault-tolerant and the data is accessible even if some of the network nodes are offline. A DFS

makes it possible to restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.