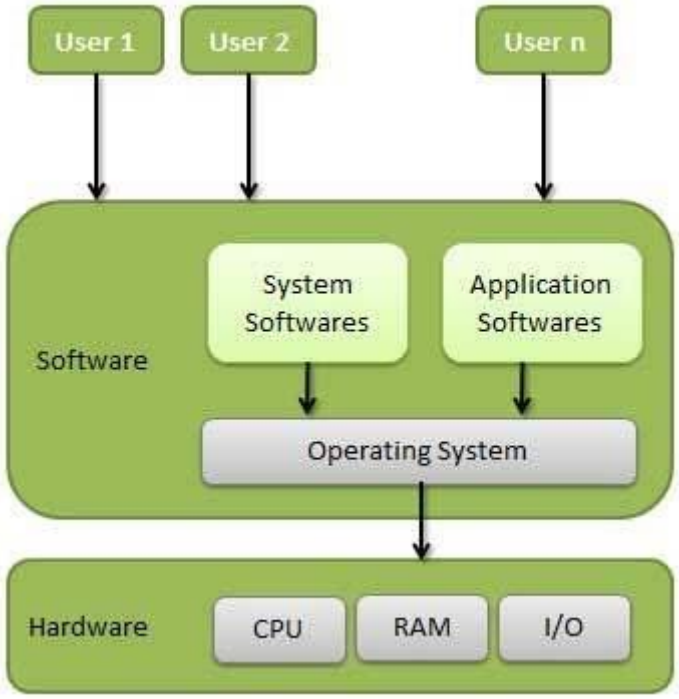
UNIT 1

Fundamentals of Operating System

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.



Need of Operating System:

- **OS as a platform for Application programs:** Operating system provides a platform, on top of which, other programs, called application programs can run. These application programs help the users to perform a specific task easily
- **Managing Input-Output unit:** Operating System also allows the computer to manage its own resources such as memory, monitor, keyboard, printer etc. Management of these resources is required for an effective utilization.
- **Consistent user interface:** Operating System provides templates, UI components to make the working of a computer, really easy for the user.
- **Multitasking:** Operating System manages memory and allow multiple programs to run in their own space and even communicate with each other through shared memory.

Multitasking gives users a good experience as they can perform several tasks on a computer at a time.

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.

Early Systems

The evolution of operating systems is directly dependent on the development of computer systems and how users use them.

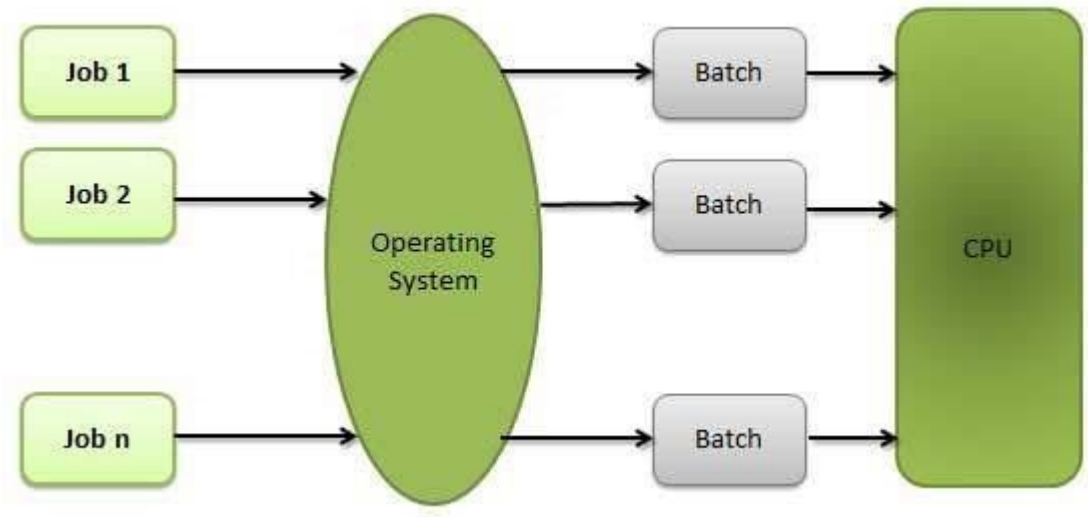
- 1945: **ENIAC**, Moore School of Engineering, University of Pennsylvania.
- 1949: **EDSAC** and **EDVAC**
- 1949: **BINAC** - a successor to the ENIAC
- 1951: **UNIVAC** by Remington
- 1952: **IBM** 701
- 1956: The interrupt
- 1954-1957: **FORTRAN** was developed

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.

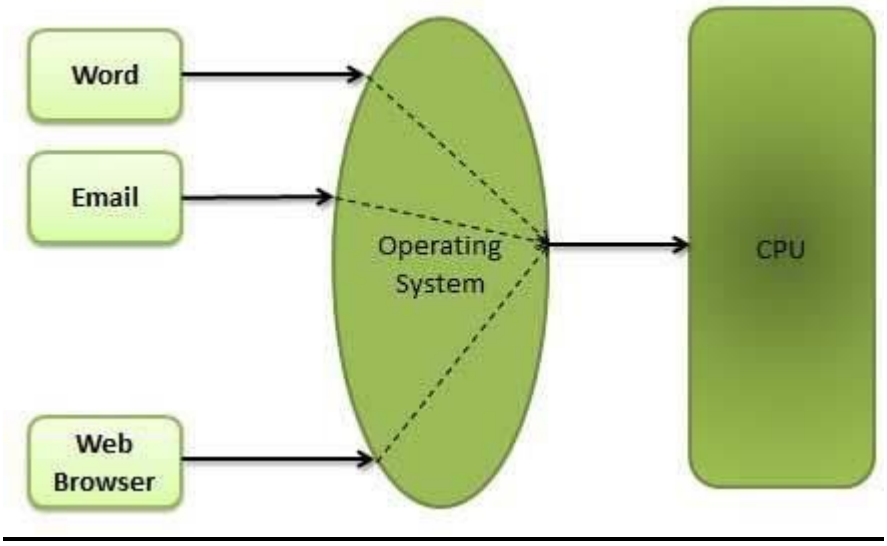
Disadvantages

- 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 −

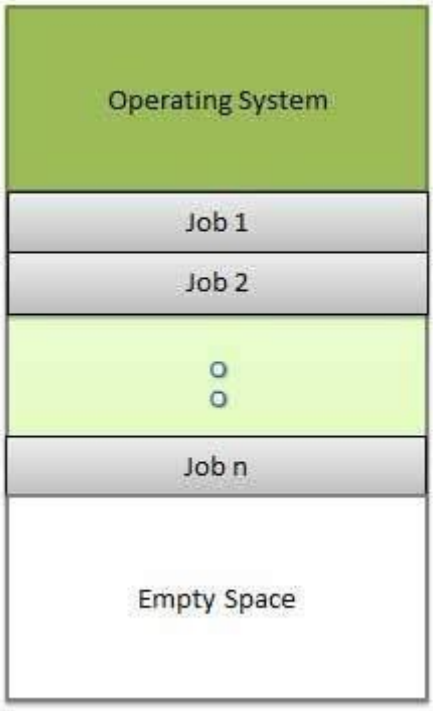
- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- 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 ensures 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.

……

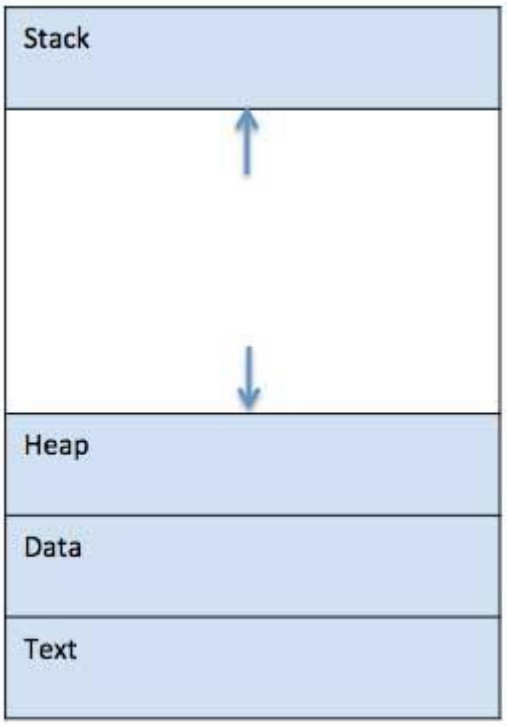
UNIT 2

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.

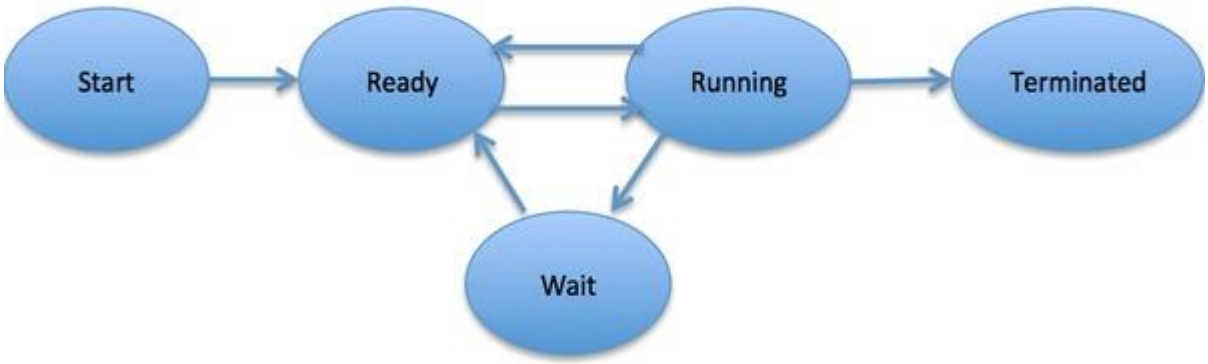


Fig: State Transaction Diagram

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.

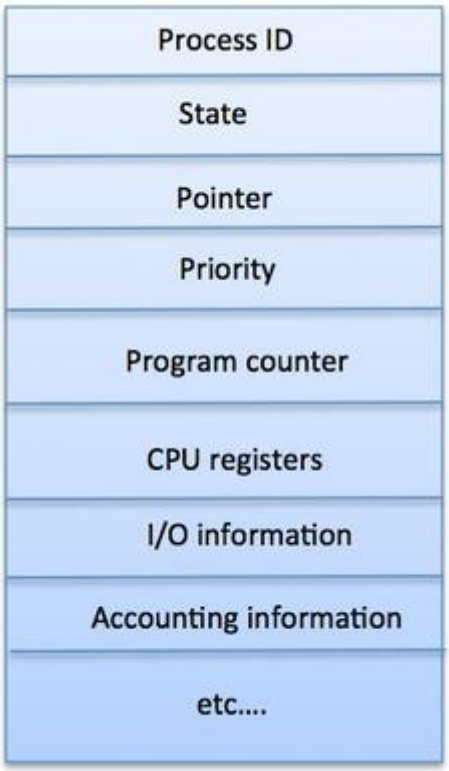
Memory management information

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 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.

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-

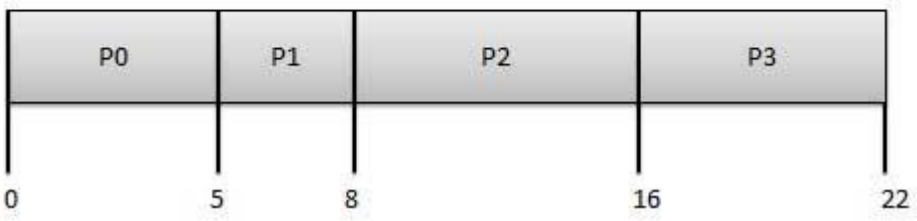
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Priority Scheduling

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.

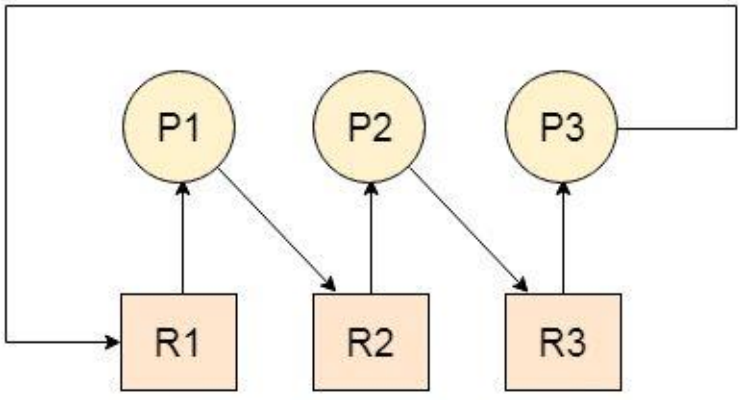
Multiple-processor scheduling

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.

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. 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.

The idea behind the approach is very simple that we have to fail one of the four conditions but there can be a big argument on its physical implementation in the system.

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.

UNIT 3

Process synchronization

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.

Sleep and Wake (Producer Consumer problem)

The concept of sleep and wake is very simple. If the critical section is not empty then the process will go and sleep. It will be waked up by the other process which is currently executing inside the critical section so that the process can get inside the critical section. In producer consumer problem, let us say there are two processes, one process writes something while the other process reads that. The process which is writing something is called **producer** while the process which is reading is called **consumer**.

The producer produces the item and inserts it into the buffer. The value of the global variable count got increased at each insertion. If the buffer is filled completely and no slot is available then the producer will sleep, otherwise it keep inserting.

On the consumer's end, the value of count got decreased by 1 at each consumption. If the buffer is empty at any point of time then the consumer will sleep otherwise, it keeps consuming the items and decreasing the value of count by 1.

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.

Classical Problems of Synchronization

Semaphore can be used in other synchronization problems besides Mutual Exclusion. Some of the classical problem depicting flaws of process synchronization in systems are given below:

1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem
3. The Readers Writers Problem

Bounded Buffer Problem

- This problem is generalised in terms of the Producer Consumer problem, where a finite buffer pool is used to exchange messages between producer and consumer processes.

  Because the buffer pool has a maximum size, this problem is often called the Bounded buffer problem.

- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

Dining Philosophers Problem

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.
- Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

The Readers Writers Problem

- In this problem there are some processes(called **readers**) that only read the shared data, and never change it, and there are other processes(called **writers**) who may change the data in addition to reading, or instead of reading it.
- There are various type of readers-writers problem, most centred on relative priorities of readers and writers.

Inter Process Communication (IPC)

Inter Process Communication (IPC) is a mechanism that involves communication of one process with another process. This usually occurs only in one system.

Communication can be of two types −

- Between related processes initiating from only one process, such as parent and child processes.
- Between unrelated processes, or two or more different processes.

Following are some important terms associated with IPC:

Pipes − Communication between two related processes. The mechanism is half duplex meaning the first process communicates with the second process. To achieve a full duplex i.e., for the second process to communicate with the first process another pipe is required.

FIFO − Communication between two unrelated processes. FIFO is a full duplex, meaning the first process can communicate with the second process and vice versa at the same time.

Message Queues − Communication between two or more processes with full duplex capacity. The processes will communicate with each other by posting a message and retrieving it out of the queue. Once retrieved, the message is no longer available in the queue.

Shared Memory − Communication between two or more processes is achieved through a shared piece of memory among all processes. The shared memory needs to be protected from each other by synchronizing access to all the processes.

Semaphores − Semaphores are meant for synchronizing access to multiple processes. When one process wants to access the memory (for reading or writing), it needs to be locked (or protected) and released when the access is removed. This needs to be repeated by all the processes to secure data.

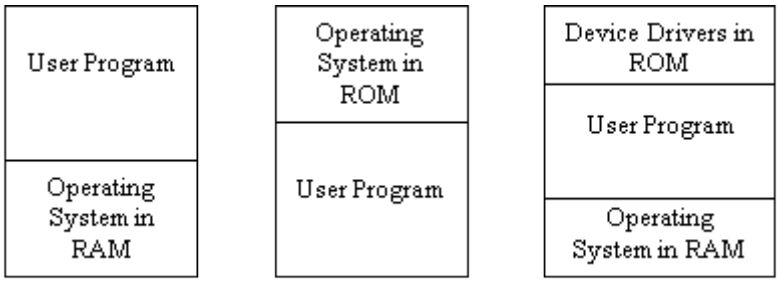
Signals − Signal is a mechanism to communication between multiple processes by way of signaling. This means a source process will send a signal (recognized by number) and the destination process will handle it accordingly.

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.

Single user operating system

The simplest possible memory management scheme is to run one program at a time sharing the memory with the operating system. Some alternatives for this scheme are shown below.



i ii iii

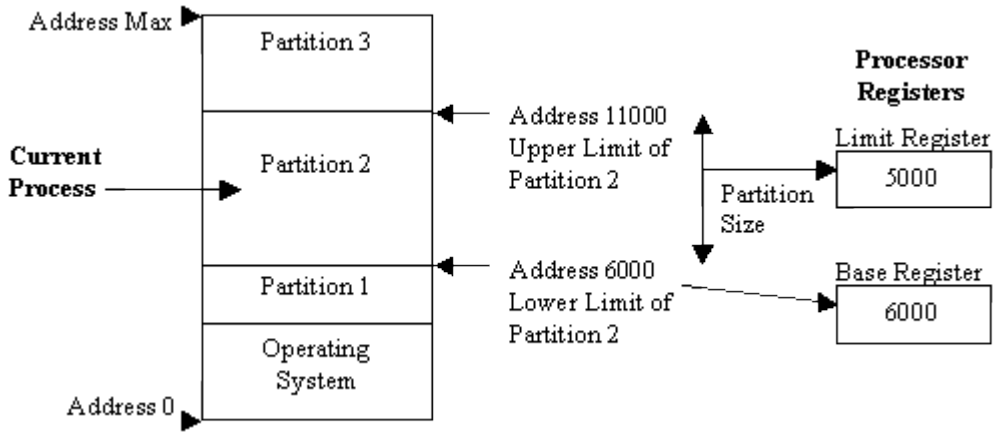
The first model was used on very early computers. The second model is used on some palmtop computers and embedded systems (systems with a specific application purpose as opposed to general purpose computing environments.) The third model is used by MS-DOS. With this model, when a user typed a command at the prompt, the operating system loaded the program from disk into memory and executed it. When it finished, control returned to the operating system which again presented the command prompt of the command interpreter. The next command would load a new program into memory, overwriting the previous one.

The single-user Operating systems can run only one process at a time and hence yields very poor utilisation. The computer system's resources are largely idle, waiting for the activity of a single user, and no background activities can take place.

Multi-user operating system

Multi-user systems allow a number of tasks to be initiated by different users logged in to the system. Multi-programming can significantly improve device utilisation. While one process is using the CPU, another may be carrying out a disk event or a network event. With a greater number of processes for the schedulers to choose from, it should be possible to keep each of the resources much busier. Multi-tasking and multi-user systems introduce some problems for the memory manager. One is that the memory space of one process needs to be protected from that of other processes. Otherwise malicious users or faulty processes could interfere with the memory space of correct processes.

Access time to memory is vitally important, so the protection features and the dynamic relocation features are built into the hardware design for speed, rather than being implemented in software.A very simple organisation of memory for holding a number of processes is given below:-



Partitioning in Operating System:

There are two Memory Management Techniques: Contiguous, and Non-Contiguous. 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.

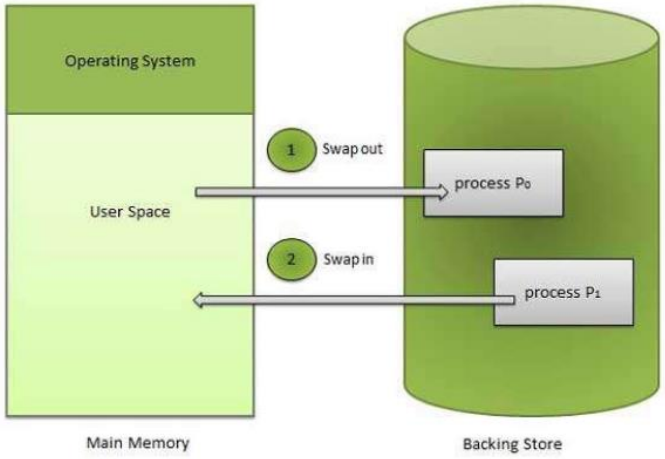
Internal fragmentation: Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

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.

Swapping

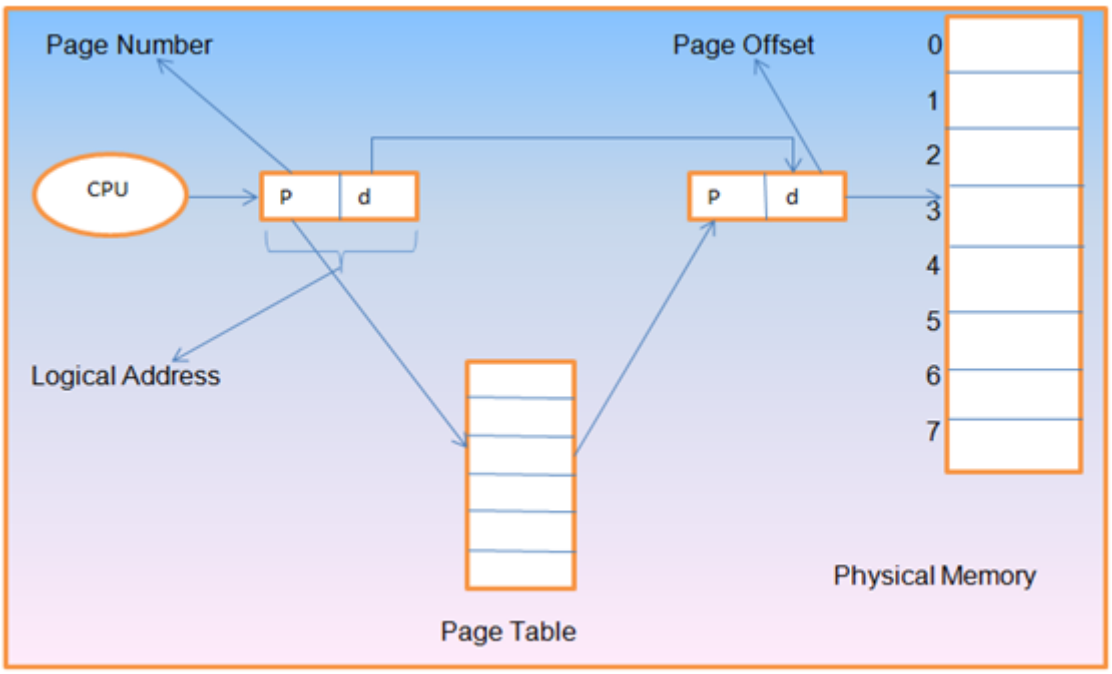
Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory. Though performance is usually 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.

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.



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

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.

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.

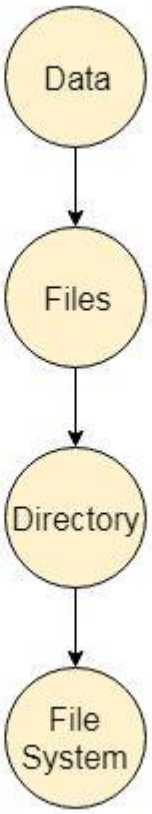
UNIT 1V

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.

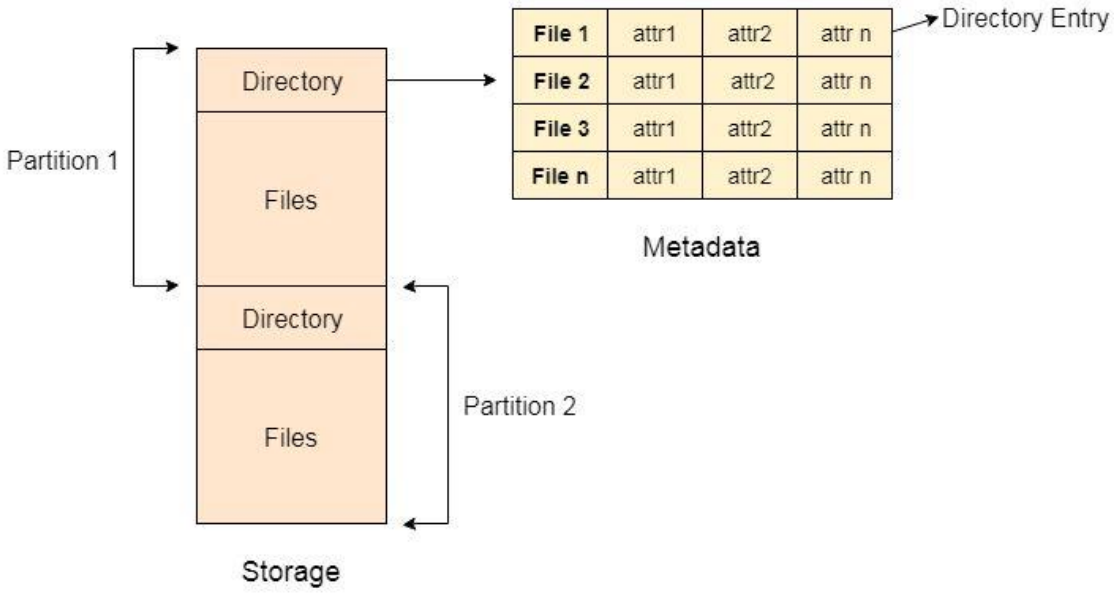
Indexed sequential access

- 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.

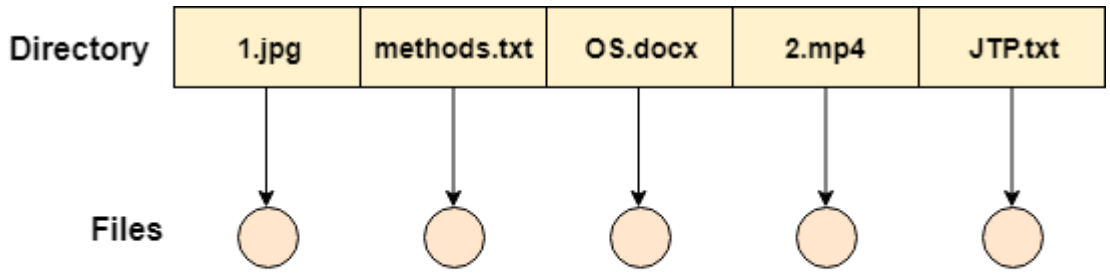


A directory can be viewed as a file which contains the Meta data of the bunch of files. Every Directory supports a number of common operations on the file:

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

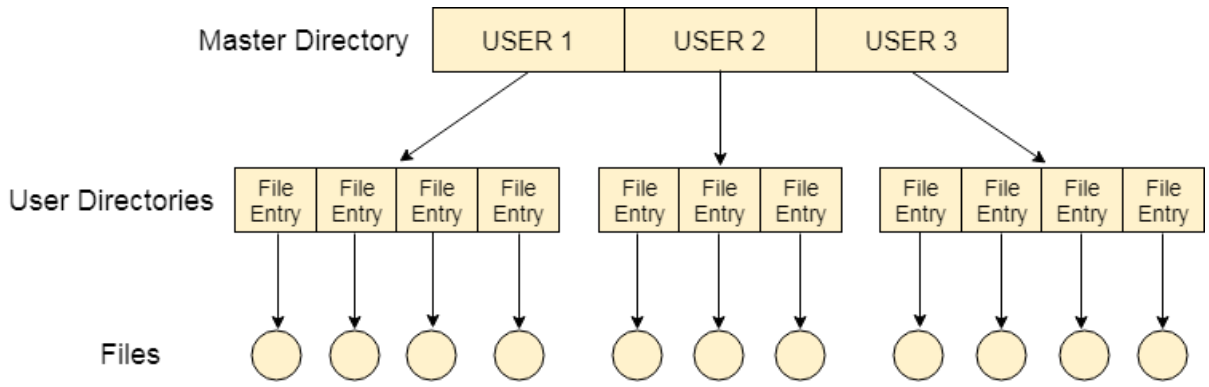
1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

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

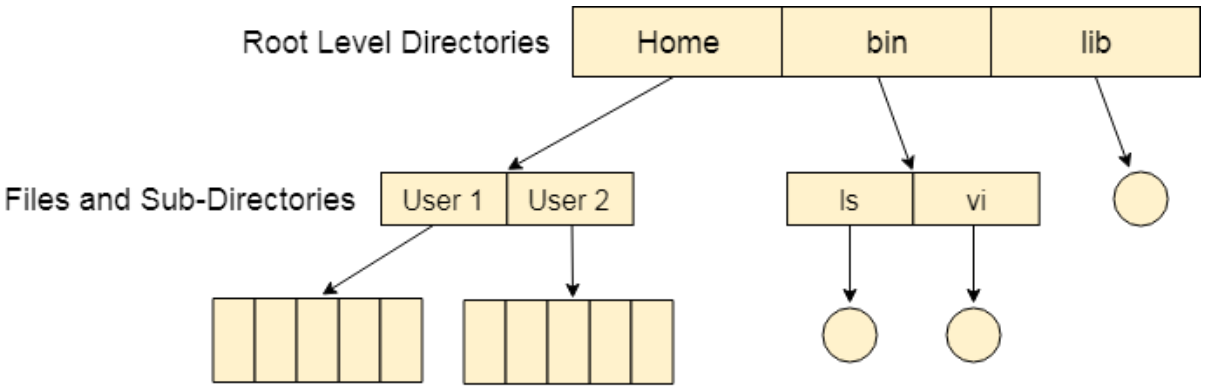
1. Each files has a path name as ***/User-name/directory-name/***
2. Different users can have the same file name.
3. Searching becomes more efficient as only one user's list needs to be traversed.
4. The same kind of files cannot be grouped into a single directory for a particular user.

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

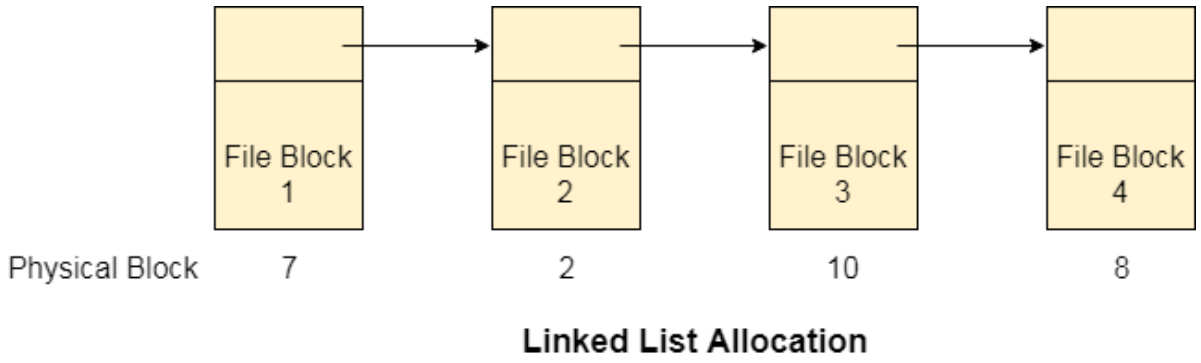
1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

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.

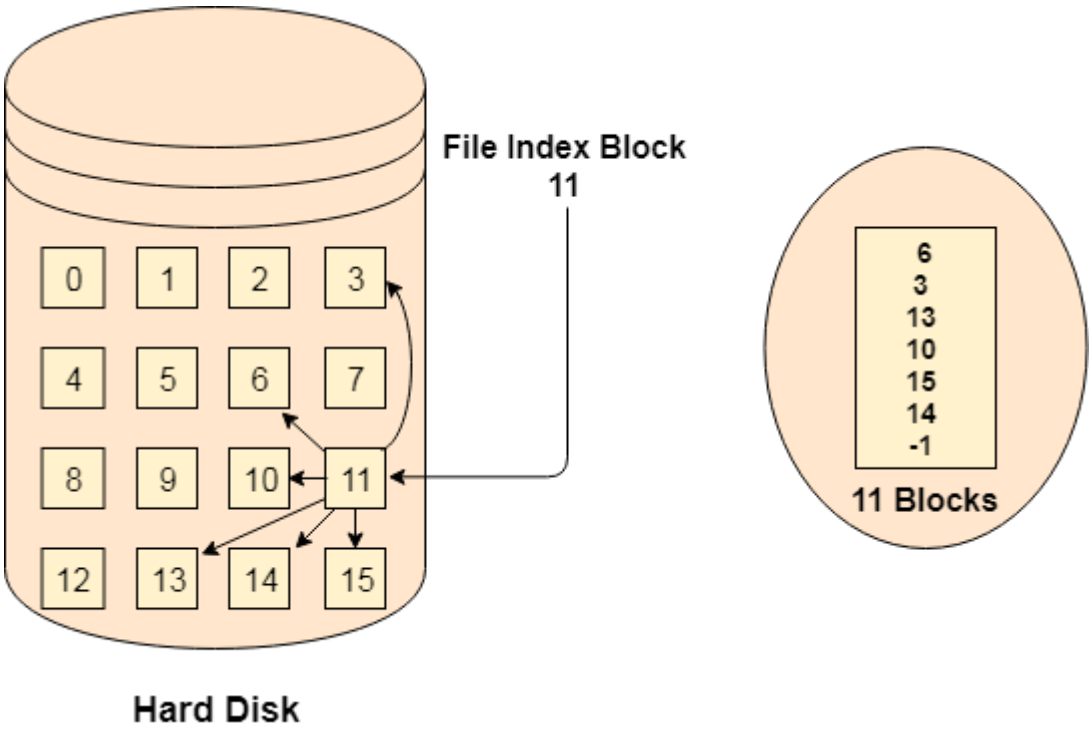
Disadvantages

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.

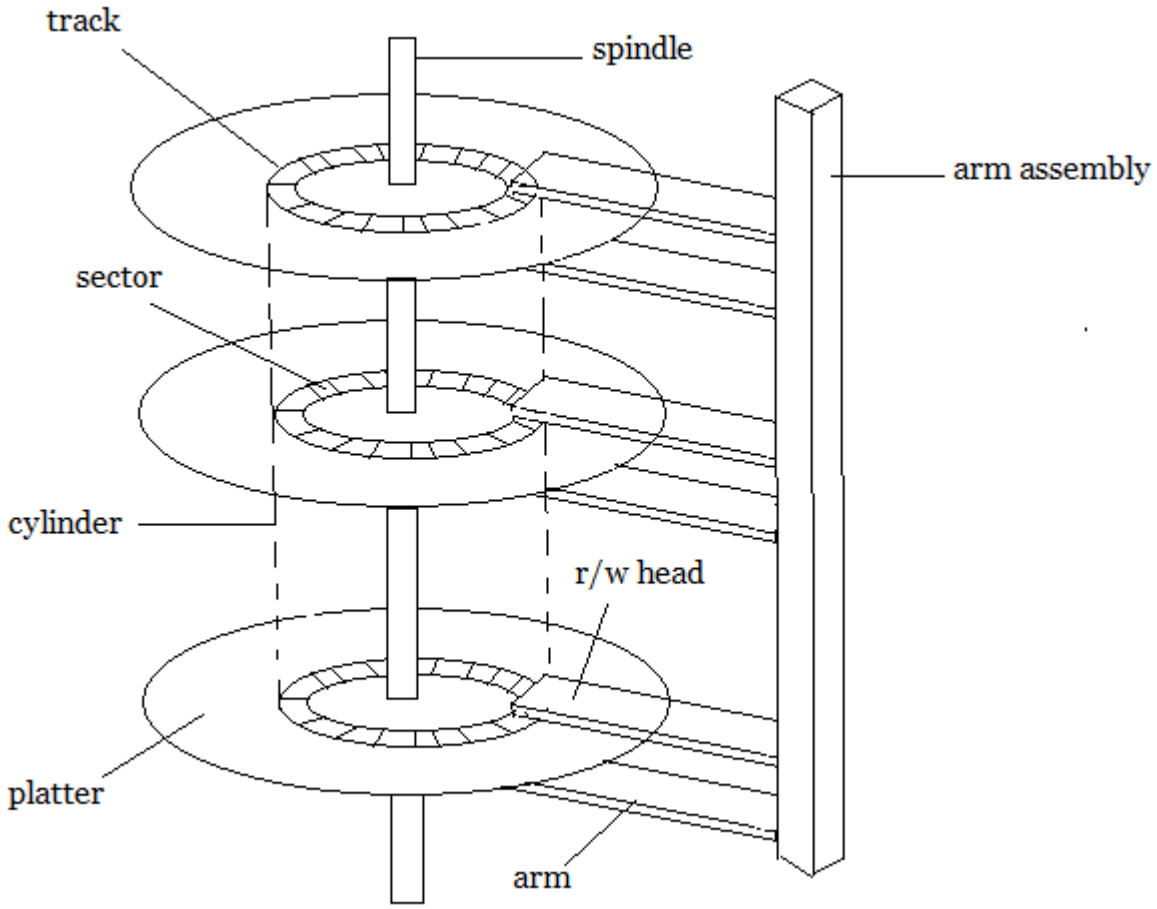


Fig. Structure of a magnetic disk

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.

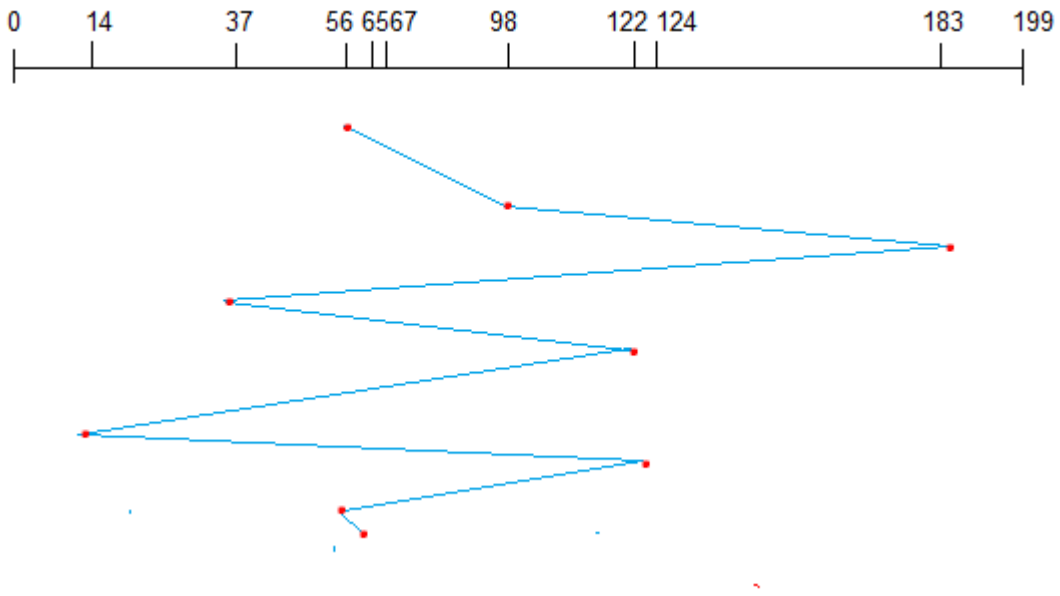
Disadvantages

- 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

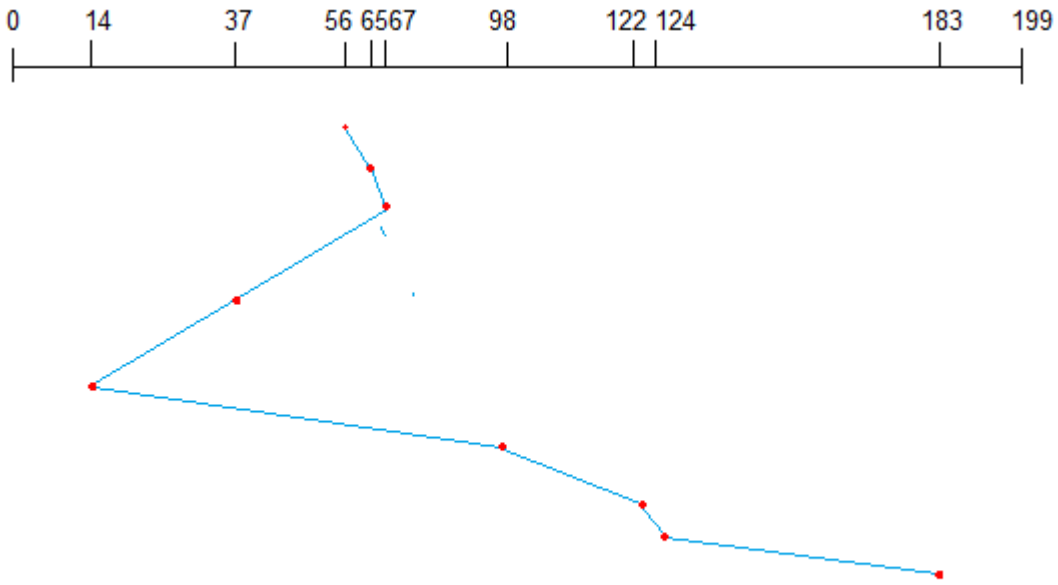
- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

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.

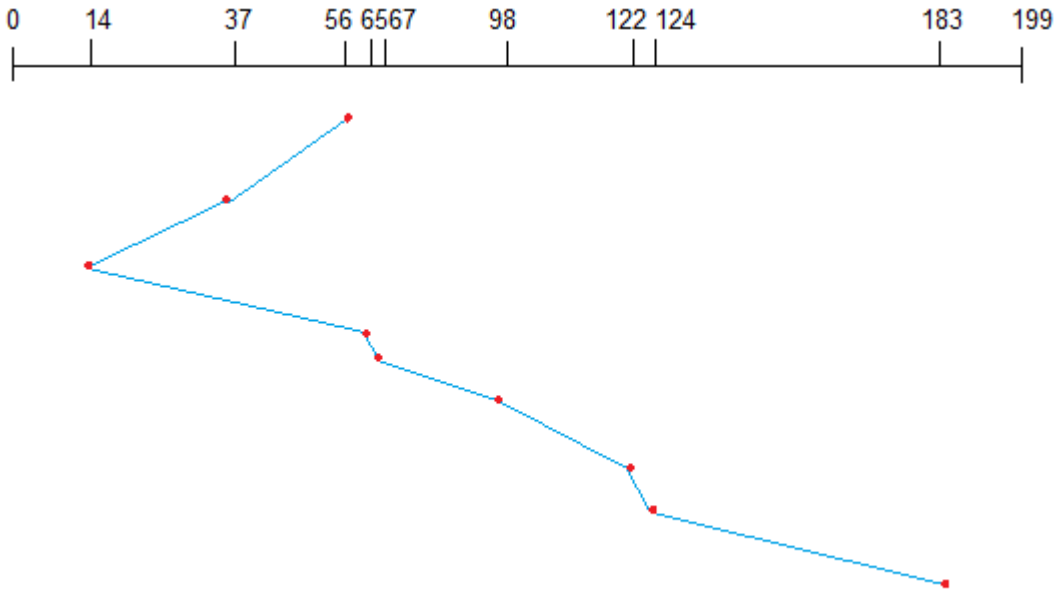
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

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

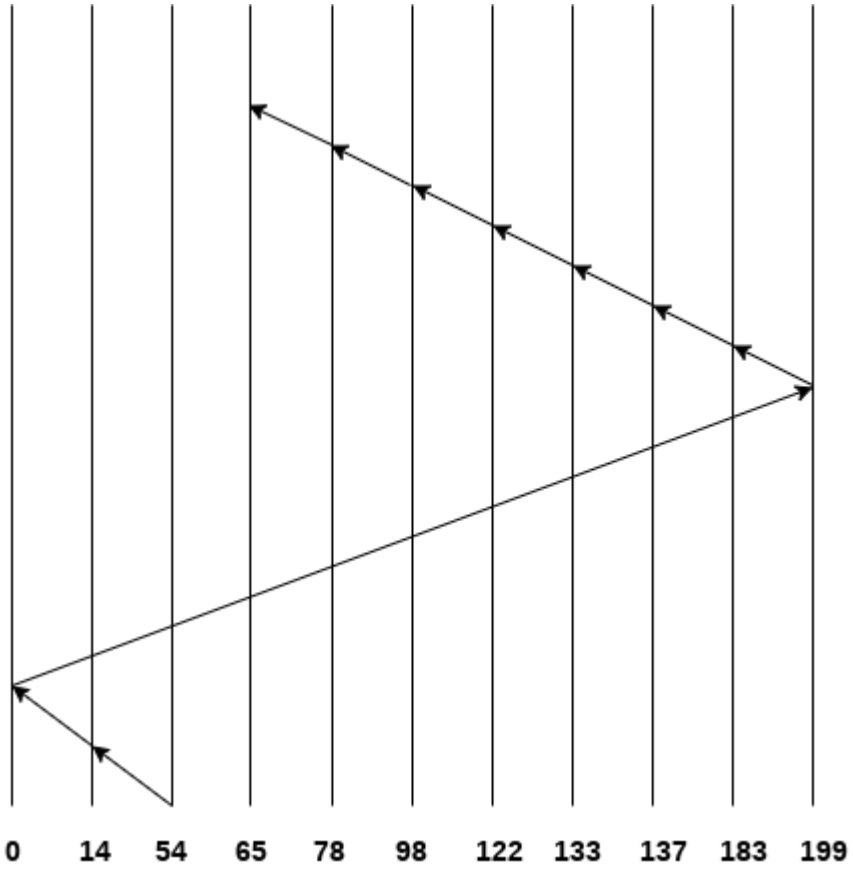
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

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

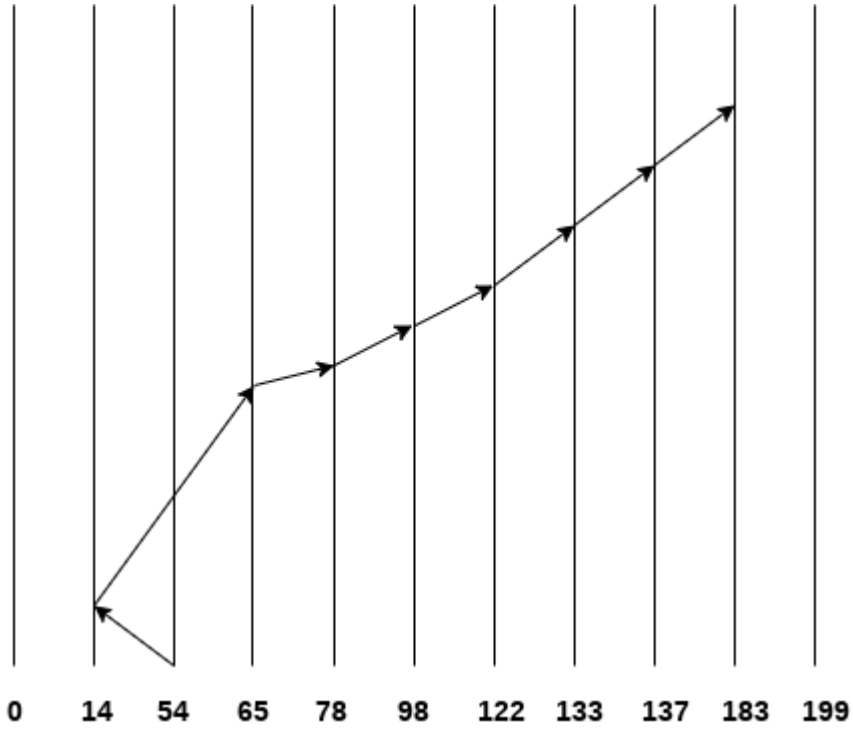
It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to 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.

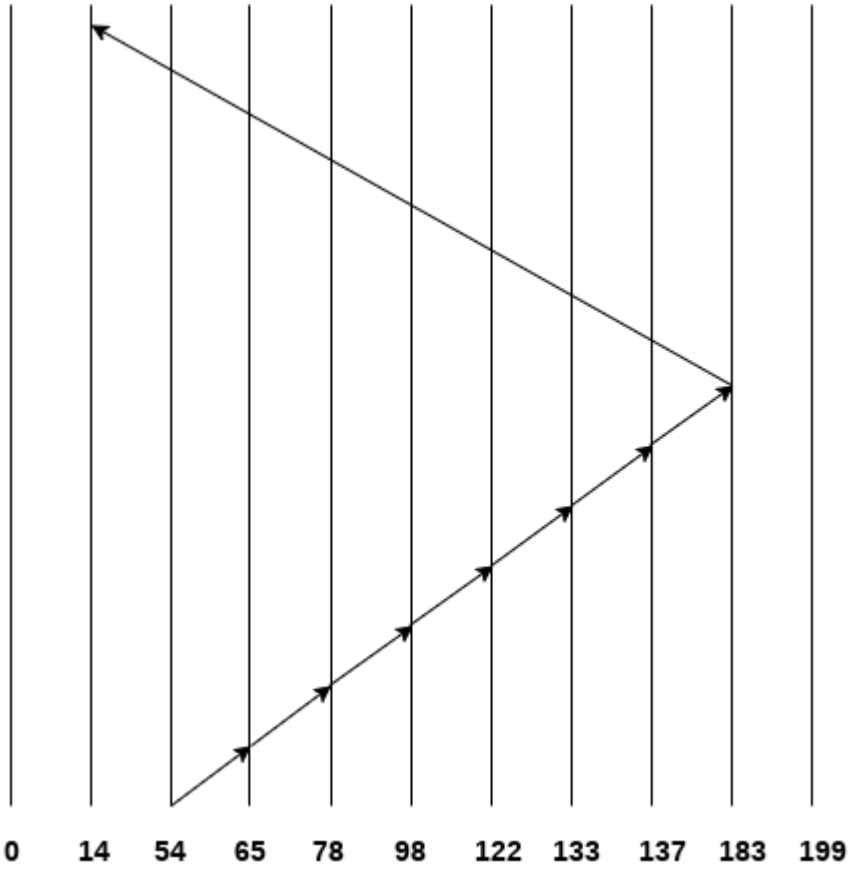
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

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

---

****

****

****