An operating system acts as an interface between the software and different parts of the computer or the computer hardware.

Types of Operating Systems

- **Batch Operating System:** A Batch Operating System is a type of operating system that does not interact with the computer directly. There is an operator who takes similar jobs having the same requirements and groups them into batches.
- **Time-sharing Operating System:** Time-sharing Operating System is a type of operating system that allows many users to share computer resources (maximum utilization of the resources).
- **Distributed Operating System:** Distributed Operating System is a type of operating system that manages a group of different computers and makes appear to be a single computer. These operating systems are designed to operate on a network of computers. They allow multiple users to access shared resources and communicate with each other over the network. Examples include Microsoft Windows Server and various distributions of Linux designed for servers.
- **Network Operating System:** Network Operating System is a type of operating system that runs on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions.
- **Real-time Operating System:** Real-time Operating System is a type of operating system that serves a real-time system and the time interval required to process and respond to inputs is very small. These operating systems are designed to respond to events in real time. They are used in applications that require quick and deterministic responses, such as embedded systems, industrial control systems, and robotics.
- **Multiprocessing Operating System:** Multiprocessor Operating Systems are used in operating systems to boost the performance of multiple CPUs within a single computer system. Multiple CPUs are linked together so that a job can be divided and executed more quickly.

- **Single-User Operating Systems:** Single-User Operating Systems are designed to support a single user at a time. Examples include Microsoft Windows for personal computers and Apple macOS.
- **Multi-User Operating Systems:** Multi-User Operating Systems are designed to support multiple users simultaneously. Examples include Linux and Unix.
- **Embedded Operating Systems:** Embedded Operating Systems are designed to run on devices with limited resources, such as smartphones, wearable devices, and household appliances. Examples include Google's Android and Apple's iOS.
- **Cluster Operating Systems:** Cluster Operating Systems are designed to run on a group of computers, or a cluster, to work together as a single system. They are used for high-performance computing and for applications that require high availability and reliability. Examples include Rocks Cluster Distribution and OpenMPI.

System Call

A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**.

A system call is initiated by the program executing a specific instruction, which triggers a switch to kernel mode, allowing the program to request a service from the OS.

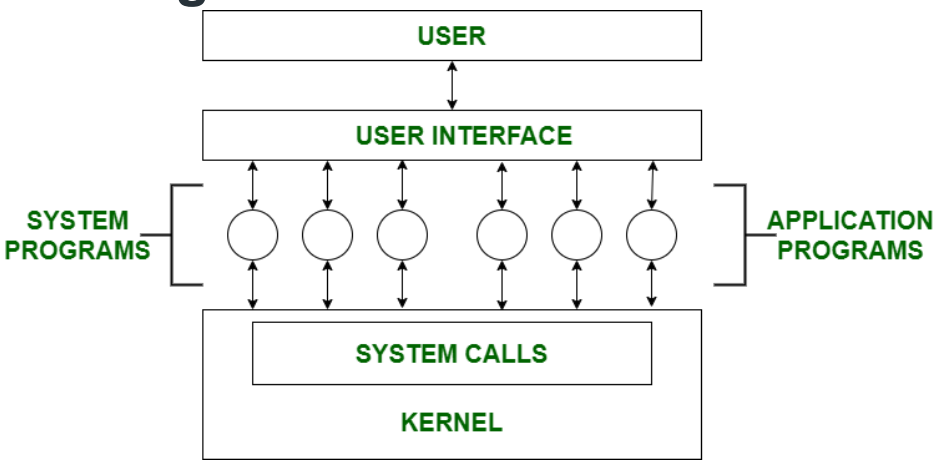
Here is a detailed explanation step by step how system calls work:

- Users need special resources
- The program makes a system call request
- Operating system sees the system call
- The operating system performs the operations
- Operating system give control back to the program

Process Control	CreateProcess() ExitProcess()	Fork() Exit()

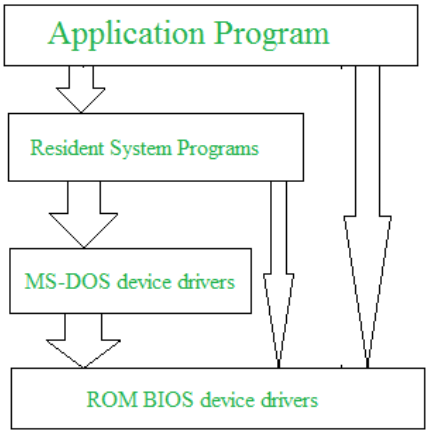
	WaitForSingleObject()	Wait()

System Programs



Operating Systems Structures

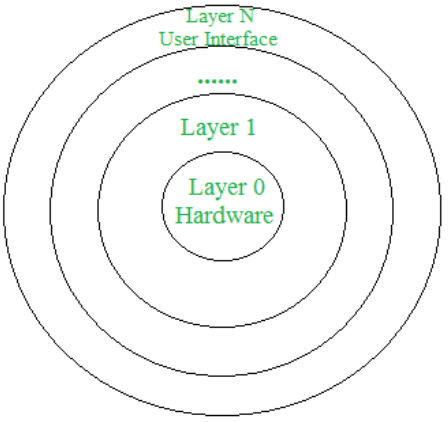
- **Simple/Monolithic Structure**
  do not have well-defined structures and are small, simple, and limited. The interfaces and levels of functionality are not well separated.



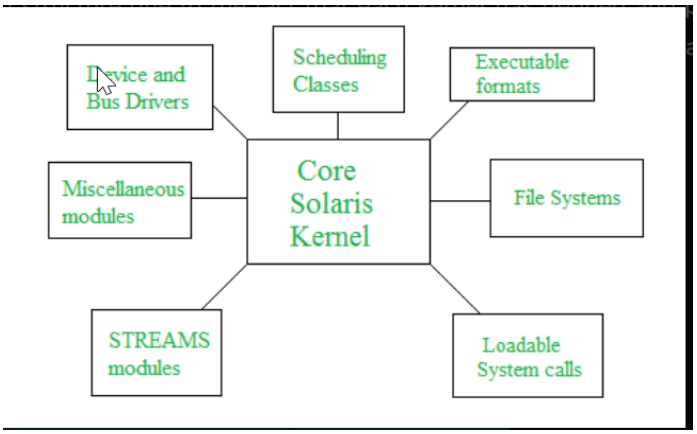
- **Micro-Kernel Structure**
  This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs.
- **Hybrid-Kernel Structure**
  Hybrid-kernel structure is nothing but just a combination of both monolithic-kernel structure and micro-kernel structure.
- **Exo-Kernel Structure**

provide application-level management of hardware resources. By separating resource management from protection, the exokernel architecture aims to enable application-specific customization.

- **Layered Structure**
  In this structure, the OS is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers.



- **Modular Structure**
  It involves designing of a modular kernel. The kernel has only a set of core components and other services are added as dynamically loadable modules to the kernel either during runtime or boot time.



- **Virtual Machines**
  Based on our needs, a virtual machine abstracts the hardware of our personal computer, including the CPU, disc drives, RAM, and NIC (Network Interface Card), into a variety of different execution contexts, giving us the impression that each execution environment is a different computer.

Booting and Dual Booting of Operating System

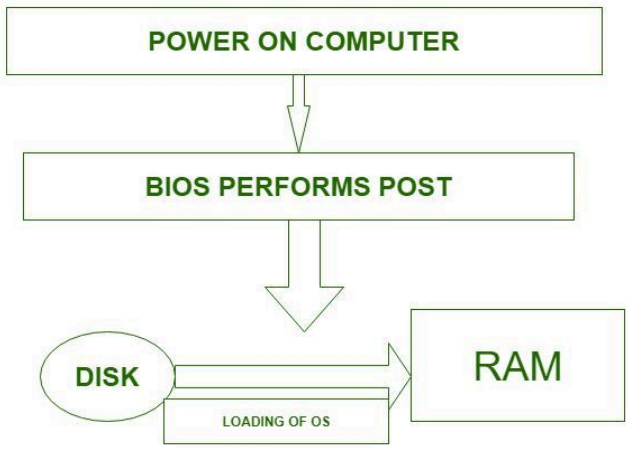
When a computer system is started, there is a mechanism in the system that loads the operating system from the secondary storage into the main memory, or RAM, of the system.

1. Cold or Hard Booting

A state in which a computer is switched on from being switched off

2. Soft or Warm Booting

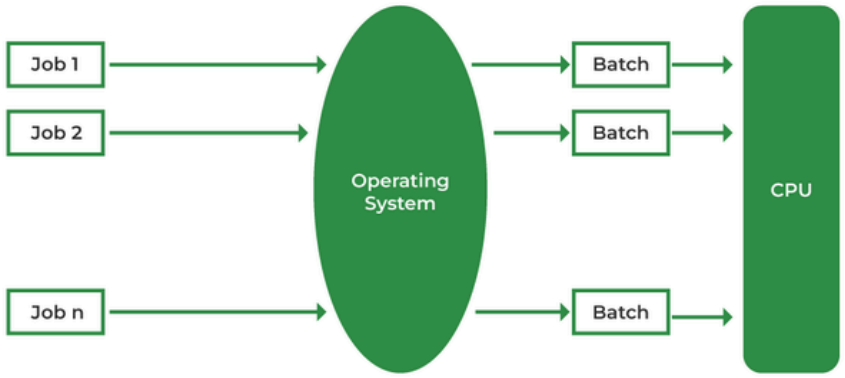
Soft boot or restart method Warm Booting, also called soft boots or restarts, reboots a computer system without shutting it down entirely.



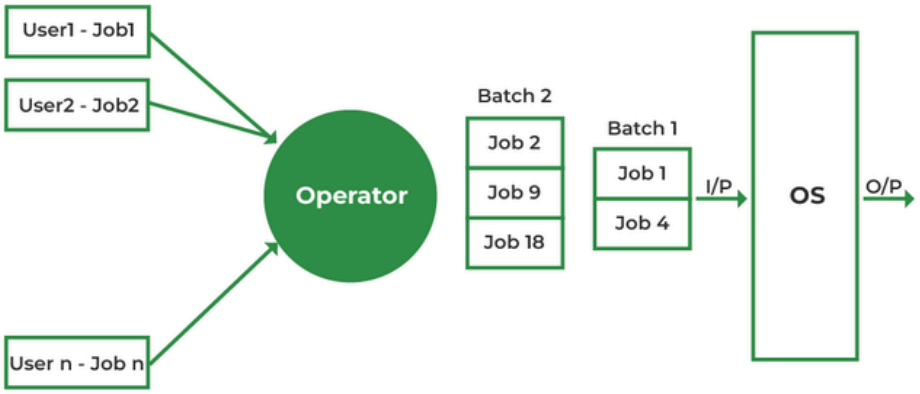
When two operating systems are installed on a computer system, it is called dual booting. A boot loader that understands multiple file systems and multiple operating systems can occupy the boot space. Once loaded, it can boot one of the operating systems available on the disk. The disk can have multiple partitions, each containing a different type of operating system. When a computer system turns on, a boot manager program displays a menu, allowing the user to choose the operating system to use.

Batch Processing Operating System

Jobs having similar requirements are grouped and executed as a group to speed up processing. Users using batch operating systems do not interact with the computer directly. Each user prepares their job using an offline device for example a punch card and submits it to the computer operator.



It contains a command line interface, a library for scheduling tasks, and a user interface for managing tasks.



Some examples of batch-processing operating systems include IBM's z/OS , Unisys MCP , and Burroughs MCP/BCS

Advantages:

Resource Efficiency ,High Throughput , Error Reduction, Simplified Management , Cost Efficiency , Scalability

Disadvantages:

Limited functionality , Security issues , Interruptions , Inefficiency

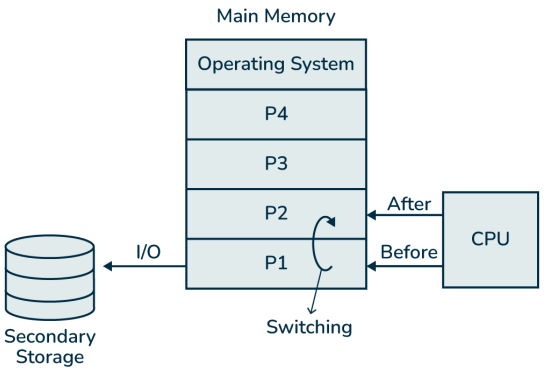
Multiprogramming in Operating System

Multiprogramming means more than one program can be active at the same time.

Features of Multiprogramming

- Need Single CPU for implementation.
- Context switch between process.
- Switching happens when current process undergoes waiting state.
- CPU idle time is reduced.
- High resource utilization.
- High Performance.

In multiprogramming system, multiple programs are to be stored in memory and each program has to be given a specific portion of memory which is known as process. The operating system handles all these process and their states. Before the process undergoes execution, the operating system selects a ready process by checking which one process should udergo execution. When the chosen process undergoes CPU execution, it might be possible that in between process need any input/output operation at that time process goes out of main memory for I/O operation and temporarily stored in secondary storage and CPU switches to next ready process. And when the process which undergoes for I/O operation comes again after completing the work, then CPU switches to this process. This switching is happening so fast and repeatedly that creates an illusion of simultaneous execution.



Some examples of multiprogramming operating systems are

IBM OS/360 , UNIX , VMS , Windows NT , Linux , macOS , HP-UX

Multi-tasking

Refers to the ability of an operating system to execute multiple tasks (or processes) simultaneously by rapidly switching between them.

Designed with user interaction, allowing users to run multiple applications

More complex one. It requires more sophisticated scheduling algorithms and resource management to handle multiple tasks, user inputs, and ensure a smooth experience.

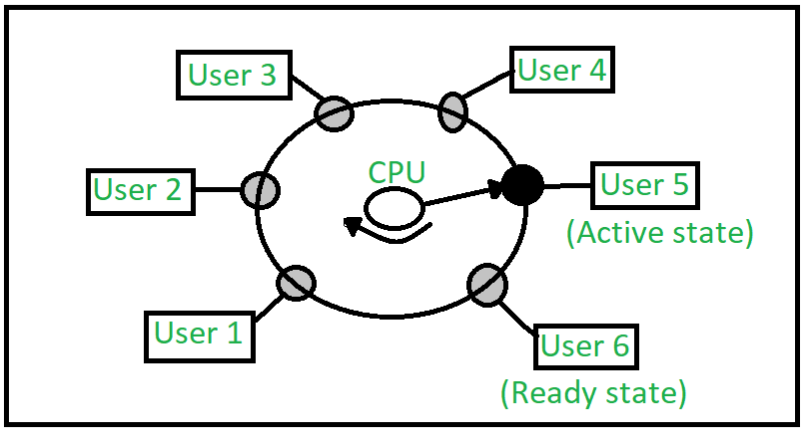
Used in interactive environments where multiple applications need to be run concurrently, particularly in desktop and mobile operating systems.

a multitasking operating system *allows multiple processes or tasks to be executed at the same time utilizing multiple CPUs.*

Time Sharing Operating System

The CPU performs many tasks by switches that are so frequent that the user can interact with each program while it is running. A time-shared operating system allows multiple users to share computers simultaneously.

A time-shared operating system uses CPU scheduling and multi-programming to provide each user with a small portion of a shared computer at once. Each user has at least one separate program in memory. A program is loaded into memory and executes, it performs a short period of time either before completion or to complete I/O. This short period of time during which the user gets the attention of the CPU is known as **time slice, time slot, or quantum**. It is typically of the order of 10 to 100 milliseconds.



1. **Active State** – The user's program is under the control of the CPU. Only one program is available in this state.
2. **Ready State** – The user program is ready to execute but it is waiting for its turn to get the CPU. More than one user can be in a ready state at a time.
3. **Waiting State** – The user's program is waiting for some input/output operation. More than one user can be in a waiting state at a time.

Network Operating System

a network operating system(NOS) is software that connects multiple devices and computers on the network and allows them to share resources on the network.

- Creating and managing user accounts on the network.
- Controlling access to resources on the network.
- Provide communication services between the devices on the network.
- Monitor and troubleshoot the network.
- Configuring and Managing the resources on the network

Types of Network Operating Systems

- **Peer to Peer:** Peer-to-peer network operating systems allow the sharing of resources and files with small-sized networks and having fewer resources. In general, peer-to-peer network operating systems are used on LAN.
- **Client/server:** Client-server network operating systems provide users access to resources through the central server. This NOS is too expensive to implement and maintain. This operating system is good for the big networks which provide many services.

Examples

- Microsoft Windows Server
- UNIX/Linux
- Artisoft’s LANtastic
- Banyan’s VINES

Real Time Operating System (RTOS)

Real-time **operating systems (RTOS)** are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines. Examples of real-time operating systems are airline traffic control systems, Command Control Systems, airline reservation systems, Heart pacemakers, Network Multimedia Systems, robots, etc.

Types of Real-Time Operating System

Hard Real-Time Operating System

These operating systems guarantee that critical tasks are completed within a range of time. For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by the robot hardly on time., scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Soft real-time operating system

is a type of operating system where the timeliness of task execution is important but not absolutely critical. In a soft real-time system, tasks are given priority and are expected to meet deadlines, but occasional violations of these deadlines are acceptable without catastrophic consequences. These systems emphasize performance and efficiency, but they can tolerate some delays or deadline misses

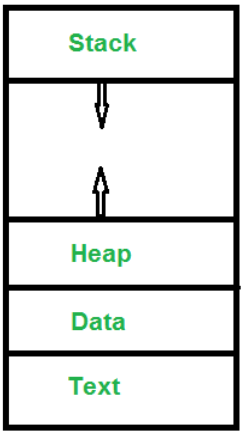
Firm Real-time Operating System

RTOS of this type have to follow deadlines as well. In spite of its small impact, missing a deadline can have unintended consequences, including a reduction in the quality of the product. Example: Multimedia applications

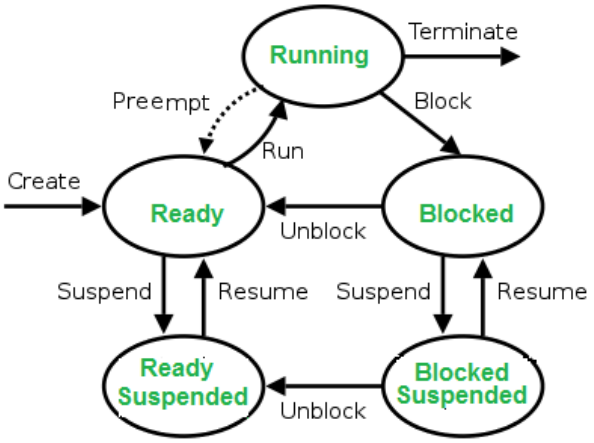
Process Management

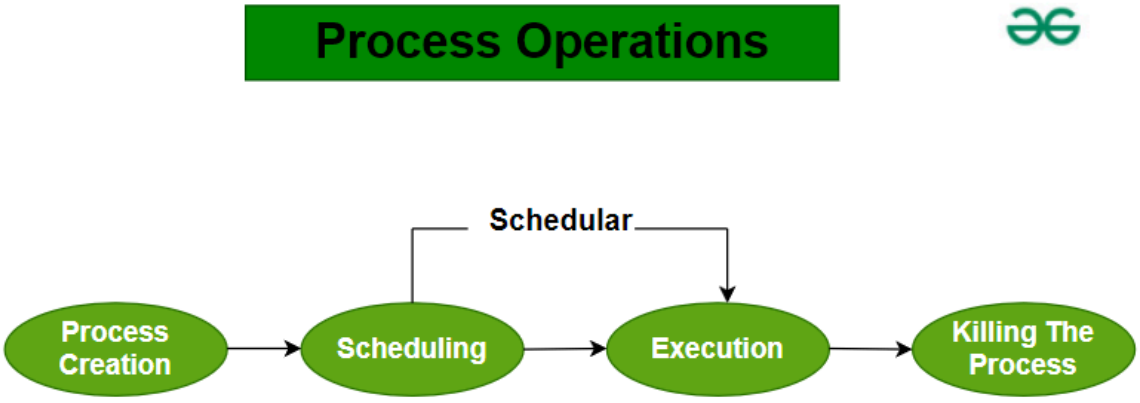
A process is a program in execution. The OS is responsible for managing the start, stop, and scheduling of processes, which are programs running on the system. The operating system uses a number of methods to prevent deadlocks, facilitate inter-process communication, and synchronize processes. Efficient resource allocation, conflict-free process execution, and optimal system performance are all guaranteed by competent process management.

A process in memory is divided into several distinct sections, each serving a different purpose.



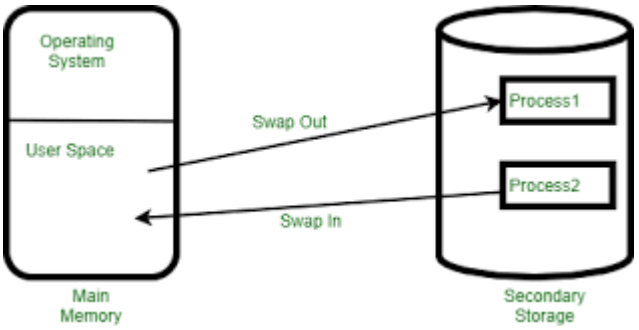
- **Text Section**: Text Section includes the current activity represented by the value of the [Program Counter](#).
- **Stack**: The stack contains temporary data, such as function parameters, returns addresses, and local variables.
- **Data Section**: Contains the global variable.
- **Heap Section**: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





- Process creation in an operating system (OS) is the act of generating a new process. This new process is an instance of a program that can execute independently.
- Once a process is ready to run, it enters the “ready queue.” The scheduler’s job is to pick a process from this queue and start its execution.
- Execution means the CPU starts working on the process. During this time, the process might:
  - Move to a waiting queue if it needs to perform an I/O operation.
  - Get blocked if a higher-priority process needs the CPU.
- After the process finishes its tasks, the operating system ends it and removes its Process Control Block (PCB).

The process of saving the context of one process and loading the context of another process is known as Context Switching. In simple terms, it is like loading and unloading the process from the running state to the ready state



A mode switch occurs when the CPU privilege level is changed, for example when a system call is made or a fault occurs. The kernel works in more a privileged mode than a standard user task. If a user process wants to access things that are only accessible to the kernel, a mode switch must occur.

CPU-Bound vs I/O-Bound Processes

A CPU-bound process requires more CPU time or spends more time in the running state. An I/O-bound process requires more I/O time and less CPU time. An I/O-bound process spends more time in the waiting state.

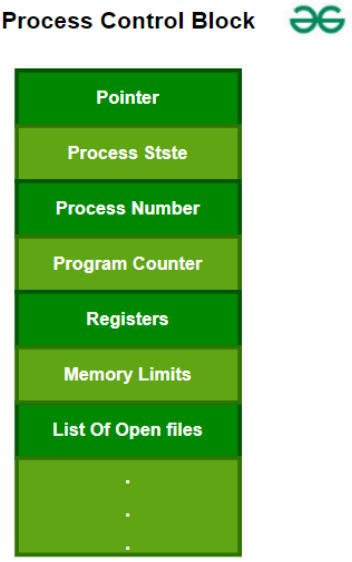
Process Scheduling Algorithms

The operating system can use different scheduling algorithms to schedule processes. Here are some commonly used timing algorithms:

- **First-Come, First-Served (FCFS):** process is executed on a first-come, first-served basis. FCFS is non-preemptive, which means that once a process starts executing, it continues until it is finished or waiting for I/O.
- **Shortest Job First (SJF):** SJF is a proactive scheduling algorithm that selects the process with the shortest burst time. The burst time is the time a process takes to complete its execution. SJF minimizes the average waiting time of processes.
- **Round Robin (RR):** Round Robin is a proactive scheduling algorithm that reserves a fixed amount of time in a round for each process. If a process does not complete its execution within the specified time, it is blocked and added to the end of the queue. RR ensures fair distribution of CPU time to all processes and avoids starvation.
- **Priority Scheduling:** This scheduling algorithm assigns priority to each process and the process with the highest priority is executed first. Priority can be set based on process type, importance, or resource requirements.
- **Multilevel Queue:** This scheduling algorithm divides the ready queue into several separate queues, each queue having a different priority. Processes are queued based on their priority, and each queue uses its own scheduling algorithm. This scheduling algorithm is useful in scenarios where different types of processes have different priorities.

Process Table and Process Control Block (PCB)

As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status.



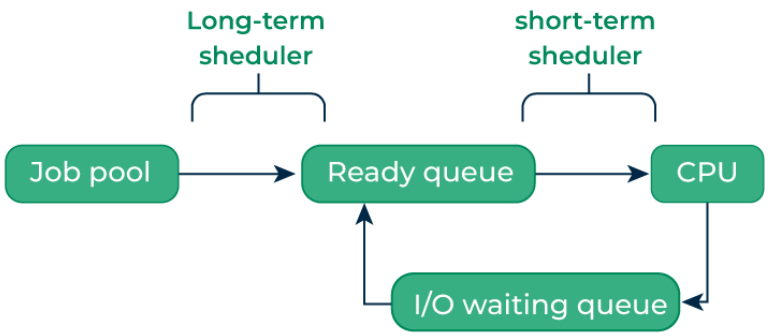
- **Pointer:** It is stack pointer that is required to be saved when the process is switched from one state to another to retain current position of process.
- **Process state:** It stores the respective state of the process.
- **Process number:** Every process is assigned a unique id known as process ID or PID which stores the process identifier.
- **Program counter:** Program Counter stores the counter, which contains the address of the next instruction that is to be executed for the process.
- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.

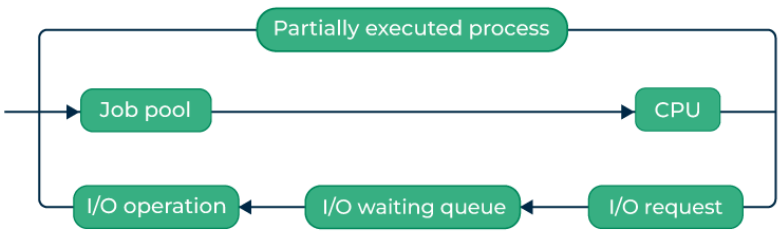
Categories of Scheduling

Scheduling falls into one of two categories:

- **Non-Preemptive:** In this case, a process's resource cannot be taken before the process has finished running. When a running process finishes and transitions to a waiting state, resources are switched.
- **Preemptive:** In this case, the OS assigns resources to a process for a predetermined period. The process switches from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may give other processes priority and substitute the currently active process for the higher priority process.

Types of Process Schedulers





medium -term scheduler

Inter Process Communication (IPC)

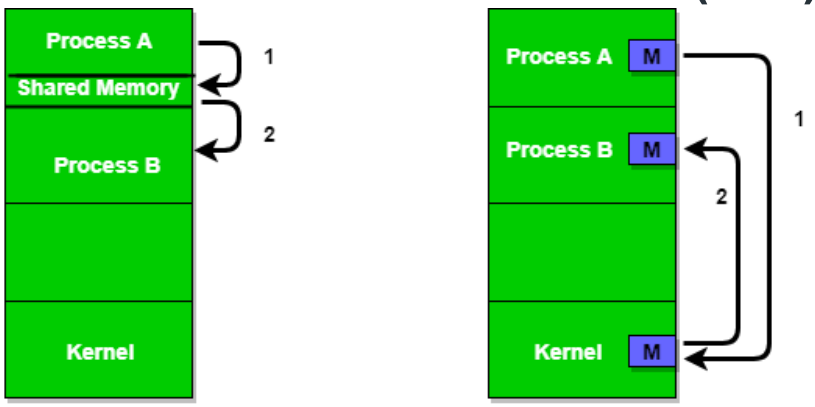


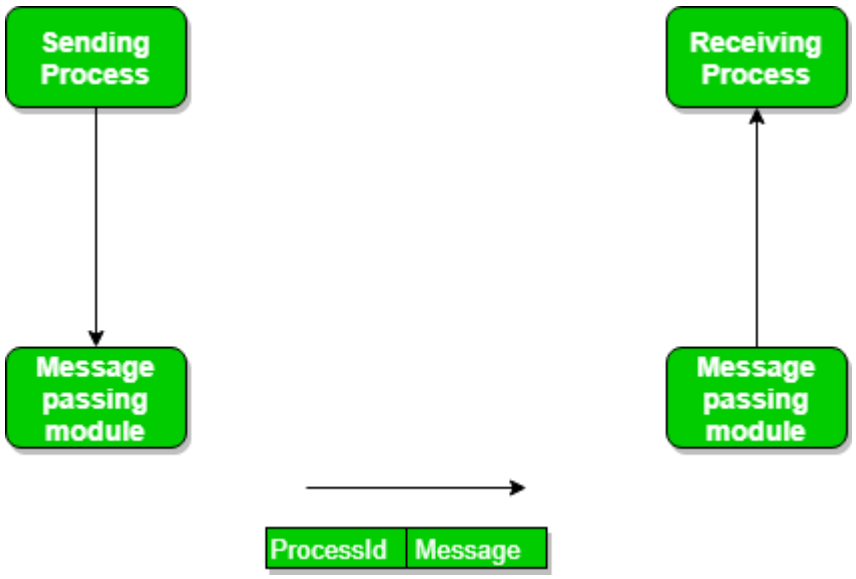
Figure 1 - Shared Memory and Message Passing

i) Shared Memory Method

The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed. There are two versions of this problem: the first one is known as the unbounded buffer problem in which the Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which the Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it.

ii) Messaging Passing Method

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destination) or **send**(message)
  – **receive**(message, host) or **receive**(message)



Message Passing Through Communication Link

Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time.
send(p1, message) means send the message to p1.
Similarly, **receive(p2, message)** means to receive the message from p2.

In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up

Examples of IPC systems

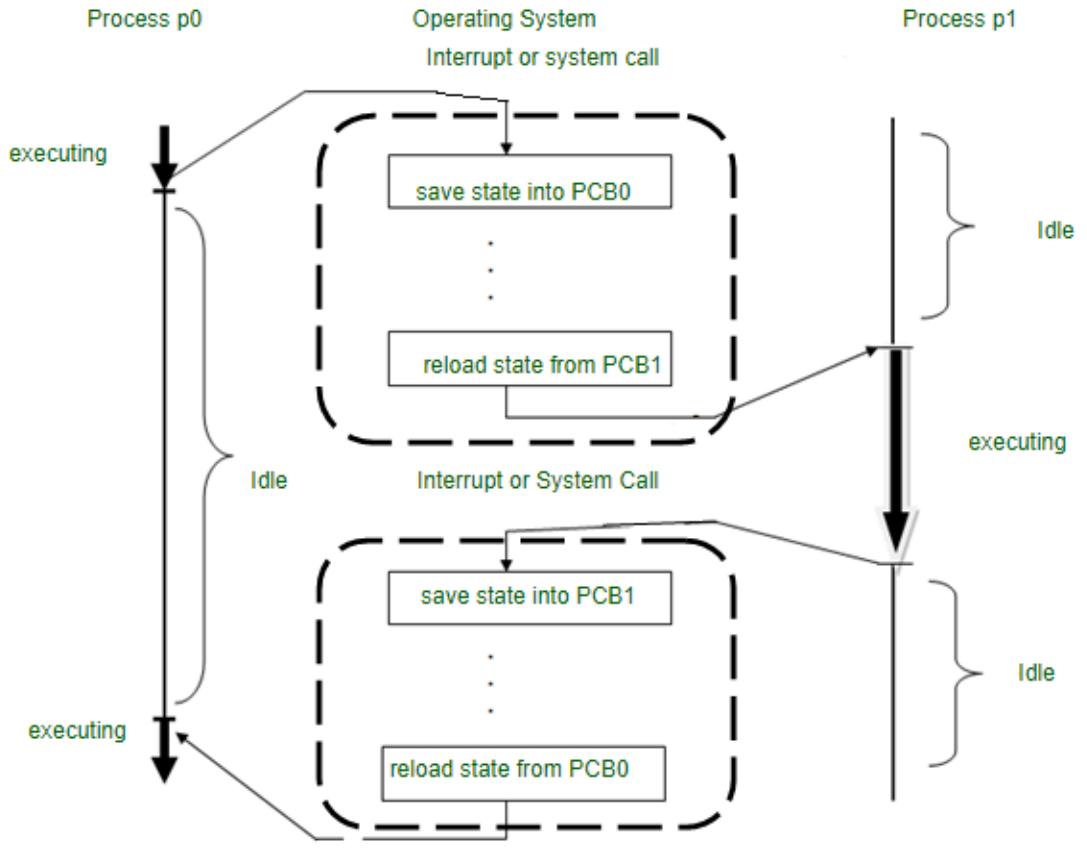
- **Posix** : uses shared memory method.
- **Mach** : uses message passing
- **Windows XP** : uses message passing using local procedural calls

.

Context Switching

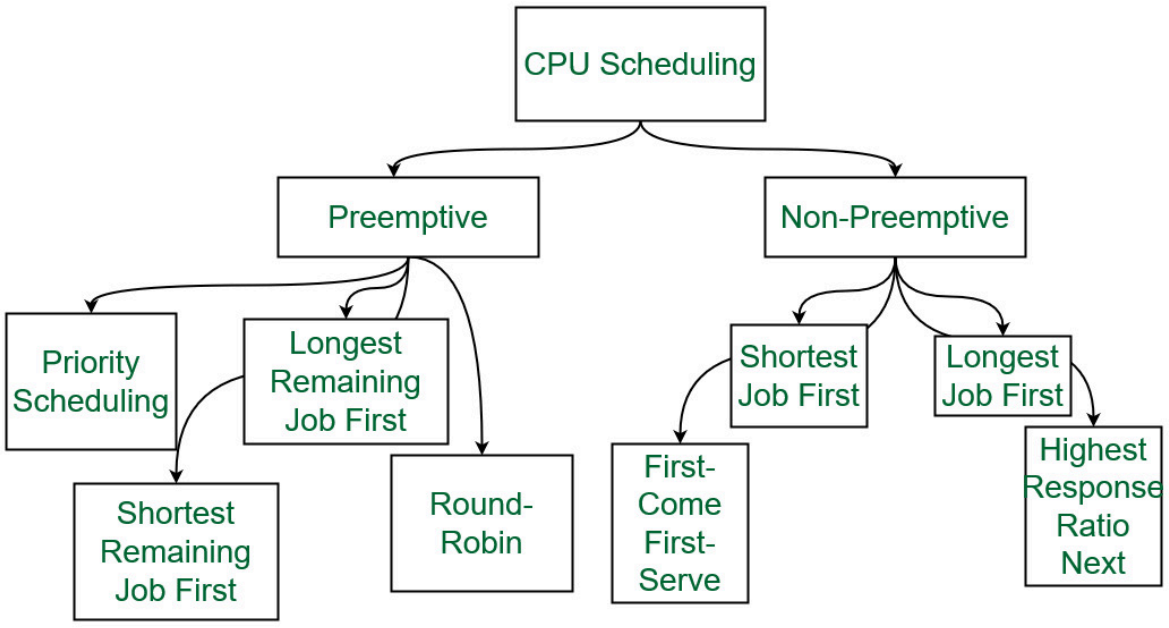
Context switching in an operating system involves saving the context or state of a running process in PCB so that it can be restored later, and then loading the context or state of another. process and run it. This is done when a high priority process enters the ready queue.

The context of a process consists its stack space, address space, virtual address space, register set image (e.g. Program Counter (PC), Instruction Register (IR), Program Status Word (PSW) and other general processor registers), Stack Pointer (SP).



CPU Scheduling in Operating Systems

- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turn Around Time:** Time Difference between completion time and arrival time.
    - Turn Around Time = Completion Time – Arrival Time
- **Waiting Time(W.T):** Time Difference between turn around time and burst time.
    - Waiting Time = Turn Around Time – Burst Time



1. First Come First Serve

FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

2. Shortest Job First(SJF)

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

3. Longest Job First(LJF)

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

4. Priority Scheduling

Preemptive Priority CPU Scheduling Algorithm is a pre-emptive method of CPU scheduling algorithm that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there is more than one process with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

5. Round Robin

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

6. Shortest Remaining Time First(SRTF)

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

7. Longest Remaining Time First(LRTF)

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

8. Highest Response Ratio Next(HRRN)

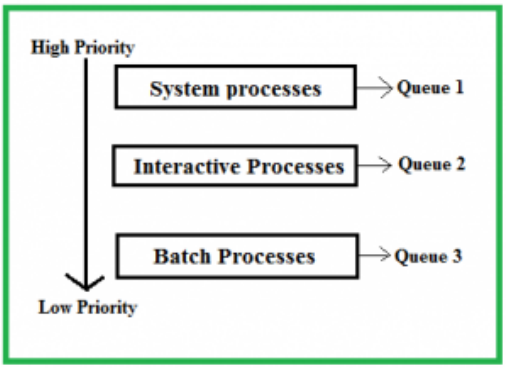
Highest Response Ratio Next is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms. The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once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

Response Ratio = (W + S)/S

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

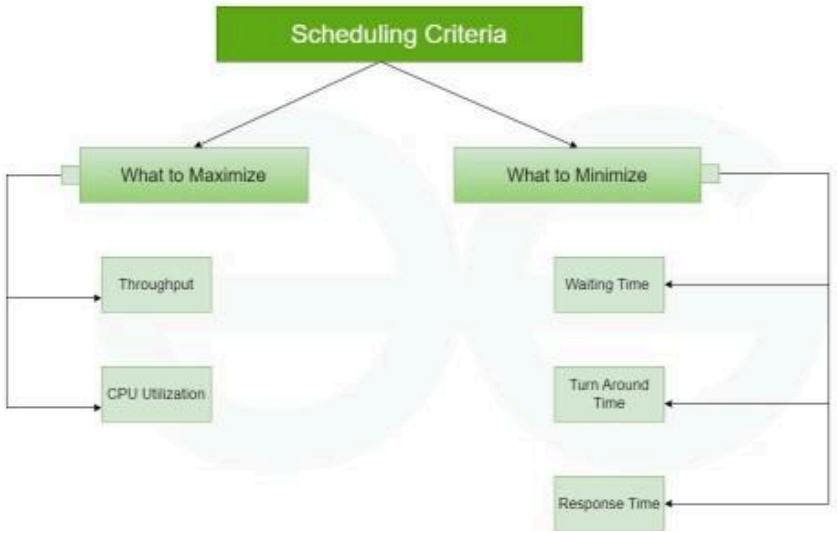
9. Multiple Queue Scheduling

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



10. Multilevel Feedback Queue Scheduling

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



Throughput

A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.

Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time

Multiple-Processor Scheduling in Operating System

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.

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**. A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Thread Scheduling

There is a component in Java that basically decides which thread should execute or get a resource in the operating system.
Scheduling of threads involves two boundary scheduling.

1. Scheduling of user-level threads (ULT) to kernel-level threads (KLT) via lightweight process (LWP) by the application developer.
2. Scheduling of kernel-level threads by the system scheduler to perform different unique OS functions.

Thread in Operating System

A thread is a single sequence stream within a process. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process.

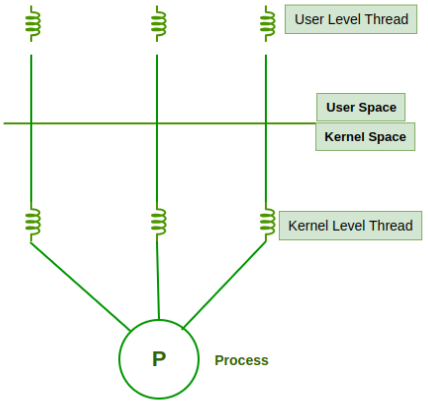
Threads can share common data so they do not need to use inter-process communication. Like the processes, threads also have states like ready, executing, blocked, etc.

These are the basic components of the Operating System.

- Stack Space
- Register Set
- Program Counter

Threads are of two types. These are described below.

- User Level Thread
- Kernel Level Thread



User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user.

A kernel Level Thread is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

Multithreading

Multithreading works by allowing a computer's processor to handle multiple tasks at the same time. Even though the processor can only do one thing at a time, it switches between different threads from various programs so quickly that it looks like everything is happening all at once.

- **Processor Handling**: The processor can execute only one instruction at a time, but it switches between different threads so fast that it gives the illusion of simultaneous execution.
- **Thread Synchronization**: Each thread is like a separate task within a program. They share resources and work together smoothly, ensuring programs run efficiently.
- **Efficient Execution**: Threads in a program can run independently or wait for their turn to process, making programs faster and responsive.
- **Programming Considerations**: Programmers need to be careful about managing threads to avoid problems like conflicts or situations where threads get stuck waiting for each other.

Multitasking is of two types: Processor-based and thread-based.

Lifecycle of a Thread

There are various stages in the lifecycle of a thread. Following are the stages a thread goes through in its whole life.

- **New:** The lifecycle of a born thread (new thread) starts in this state. It remains in this state till a program starts.
- **Runnable**: A thread becomes runnable after it starts. It is considered to be executing the task given to it.
- **Waiting**: While waiting for another thread to perform a task, the currently running thread goes into the waiting state and then transitions back again after receiving a signal from the other thread.
- **Timed Waiting:** A runnable thread enters into this state for a specific time interval and then transitions back when the time interval expires or the event the thread was waiting for occurs.
- **Terminated (Dead)**: A thread enters into this state after completing its task.

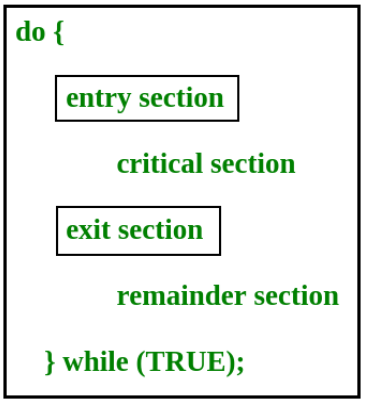
Process Synchronization

The main objective of process synchronization is to ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access.

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

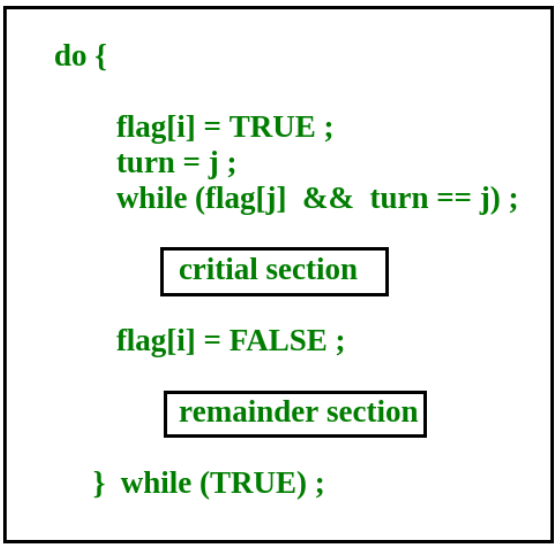
Critical Section Problem

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



- **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter the critical section next, and the selection can not be postponed indefinitely.
- **Bounded Waiting**: A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Peterson’s Solution



Peterson’s Algorithm is used to synchronize two processes. It uses two variables, a bool array **flag** of size 2 and an int variable **turn** to accomplish it. In the solution, i represents the Consumer and j represents the Producer. Initially, the flags are false. When a process wants to execute it’s critical section, it sets its flag to true and turn into the index of the other process. This means that the process wants to execute but it will allow the other process to run first. The process performs busy waiting until the other process has finished it’s own critical section. After this, the current process enters its critical section and adds or removes a random number from the shared buffer. After completing the critical section, it sets it’s own flag to false, indicating it does not wish to execute anymore.

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

Semaphores

A semaphore is a signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that is called the wait function.
A semaphore uses two atomic operations, wait P and signal for process synchronization.

- **Binary Semaphores:** They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of the mutex semaphore to 0 and some other process can enter its critical section.

- **Counting Semaphores:** They can have any value and are not restricted to a certain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

The use of critical sections in a program can cause a number of issues, including:

Deadlock: When two or more threads or processes wait for each other to release a critical section, it can result in a deadlock situation in which none of the threads or processes can move. Deadlocks can be difficult to detect and resolve, and they can have a significant impact on a program's performance and reliability.

Starvation: When a thread or process is repeatedly prevented from entering a critical section, it can result in starvation, in which the thread or process is unable to progress. This can happen if the critical section is held for an unusually long period of time, or if a high-priority thread or process is always given priority when entering the critical section.

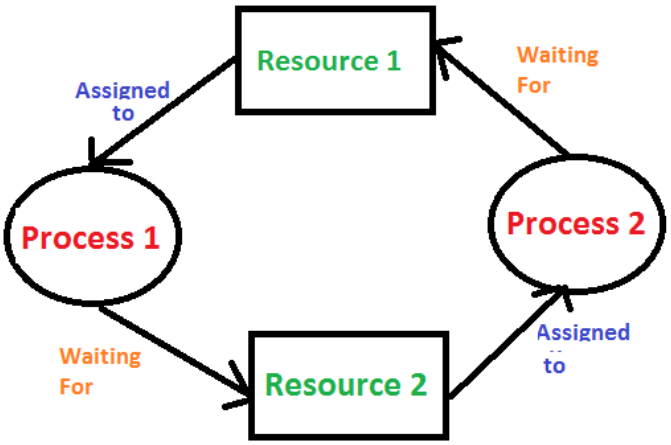
Overhead: When using critical sections, threads or processes must acquire and release locks or semaphores, which can take time and resources. This may reduce the program's overall performance.

Mutual Exclusion in Synchronization

Mutual Exclusion is a property of process synchronization that states that "**no two processes can exist in the critical section at any given point of time**".

Deadlock

A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process.



Necessary Conditions for Deadlock in OS

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

- **Mutual Exclusion:** Two or more resources are non-shareable (Only one process can use at a time).
- **Hold and Wait:** A process is holding at least one resource and waiting for resources.
- **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait:** A set of processes waiting for each other in circular form.

Deadlock Detection

- Resource Aloocation Graph
- Banker's Algorithm

There are three ways to handle deadlock

- Deadlock Prevention or Avoidance
- Deadlock Recovery
- Deadlock Ignorance

Deadlock Prevention

(i) Mutual Exclusion

We only use the Lock for the non-share-able resources and if the resource is share- able (like read only file) then we not use the locks here.

(ii) Hold and Wait

To ensure that Hold and wait never occurs in the system, we must guarantee that whenever process request for resource , it does not hold any other resources.

(iii) No Preemption

If a process is holding some resource and requestion other resources that are acquired and these resource are not available immediately then the resources that current process is holding are preempted. After some time process again request for the old resources and other required resources to re-start.

(iv) Circular Wait:

To remove the circular wait in system we can give the ordering of resources in which a process needs to acquire.

Deadlock Avoidance

We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker's algorithm (Which is in turn a gift from Dijkstra) to avoid deadlock.

Deadlock Recovery

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

There are several Deadlock Recovery Techniques:

- Manual Intervention
- Automatic Recovery
- Process Termination
- Resource Preemption

1. Manual Intervention

When a deadlock is detected, one option is to inform the operator and let them handle the situation manually.

2. Automatic Recovery

An alternative approach is to enable the system to recover from deadlock automatically. This method involves breaking the deadlock cycle by either aborting processes or preempting resources.

3. Process Termination

- **Abort all Deadlocked Processes**
- **Abort one process at a time**

4. Resource Preemption

- **Selecting a Victim**
  Resource preemption involves choosing which resources and processes should be preempted to break the deadlock.The selection order aims to minimize the overall cost of recovery. Factors considered for victim selection may include the number of resources held by a deadlocked process and the amount of time the process has consumed.

- **Rollback**
  Rolling back the process to a safe state and restarting it is a common approach

- **Starvation Prevention**
  To prevent resource starvation, it is essential to ensure that the same process is not always chosen as a victim.

Banker's Algorithm in Operating System

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

The following **Data structures** are used to implement the Banker's Algorithm:
Let **'n'** be the number of processes in the system and **'m'** be the number of resource types.

Available

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type R_j

Max

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process P_i may request at most **'k'** instances of resource type R_j.

Allocation

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process P_i is currently allocated **'k'** instances of resource type R_j

Need

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process P_i currently needs **'k'** instances of resource type R_j
- Need [i, j] = Max [i, j] – Allocation [i, j]

The request will only be granted under the below condition

- If the request made by the process is less than equal to the max needed for that process.
- If the request made by the process is less than equal to the freely available resource in the system.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4....n
2) Find an i such that both
a) Finish[i] = false
b) Need$_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)
4) if Finish [i] = true for all i
then the system is in a safe state
**-+*

Resource-Request Algorithm

Let Request$_i$ be the request array for process P$_i$. Request$_i$ [j] = k means process P$_i$ wants k instances of resource type R$_j$. When a request for resources is made by process P$_i$, the following actions are taken:

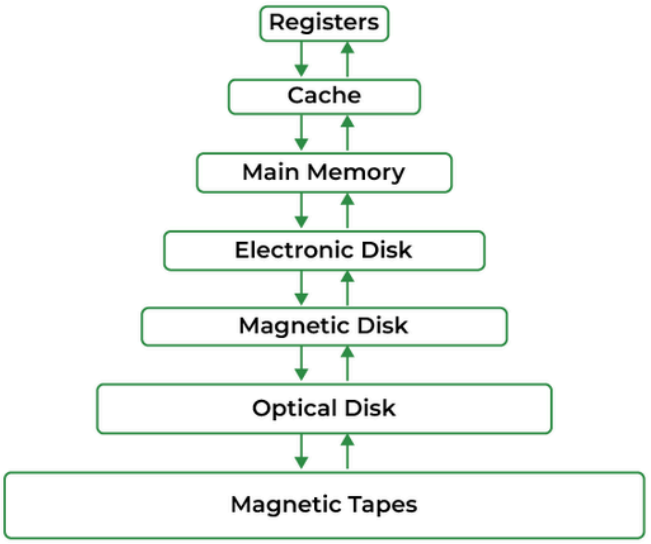
1) If Request$_i$ <= Need$_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If Request$_i$ <= Available
Goto step (3); otherwise, P$_i$ must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:
Available = Available – Requesti
Allocation$_i$ = Allocation$_i$ + Request$_i$
Need$_i$ = Need$_i$– Request$_i$

Wait-for-Graph Algorithm: It is a variant of the Resource Allocation graph. In this algorithm, we only have processes as vertices in the graph. If the Wait-for-Graph contains a cycle then we can say the system is in a Deadlock state. Now we will discuss how the Resource Allocation graph will be converted into Wait-for-Graph in an Algorithmic Approach. We need to remove resources while converting from Resource Allocation Graph to Wait-for-Graph.

- **Resource Allocation Graph:** Contains Processes and Resources.
- **Wait-for-Graph:** Contains only Processes after removing the Resources while conversion from Resource Allocation Graph.

Memory Management in Operating System

Main memory is also known as RAM (Random Access Memory).Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices.



In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management.

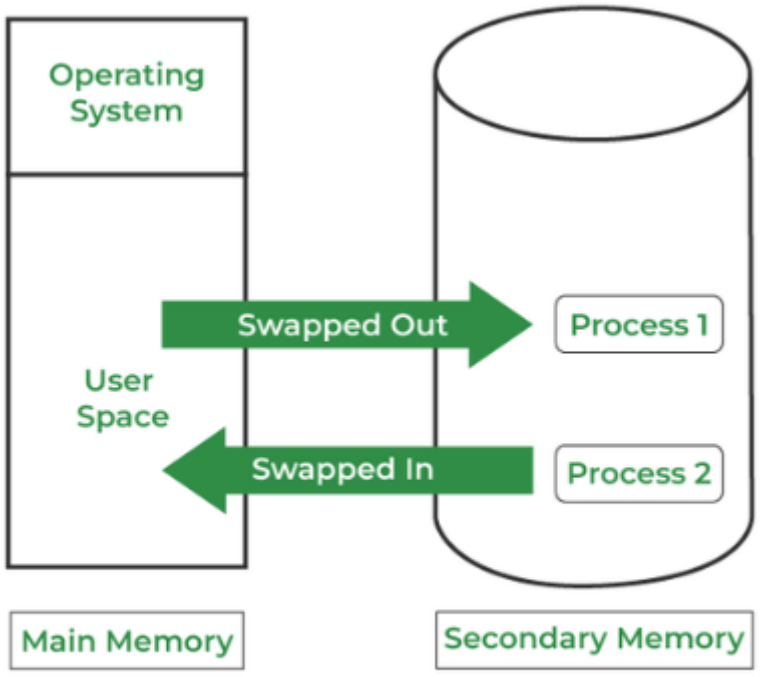
Logical and Physical Address Space

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

Static and Dynamic Loading

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

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



swapping in memory management

Memory Management with Monoprogramming (Without Swapping)

- In this approach, the operating system keeps track of the first and last location available for the allocation of the user program
- The operating system is loaded either at the bottom or at top
- Interrupt vectors are often loaded in low memory therefore, it makes sense to load the operating system in low memory
- Sharing of data and code does not make much sense in a single process environment
- The Operating system can be protected from user programs with the help of a fence register.
-

Multiprogramming with Fixed Partitions (Without Swapping)

- A memory partition scheme with a fixed number of partitions was introduced to support multiprogramming. this scheme is based on contiguous allocation
- Each partition is a block of contiguous memory
- Memory is partitioned into a fixed number of partitions.
- Each partition is of fixed size

First Fit

In the First Fit, the first available free hole fulfil the requirement of the process allocated.

Best Fit

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

Worst Fit

In the Worst Fit, allocate the largest available hole to process. This method produces the largest leftover hole.

Fragmentation

Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process.

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

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

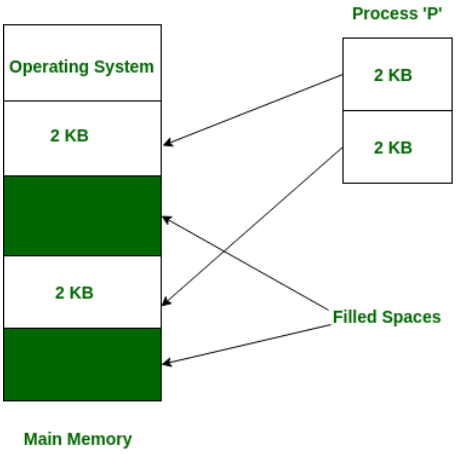
The address generated by the CPU is divided into:

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

Physical Address is divided into:

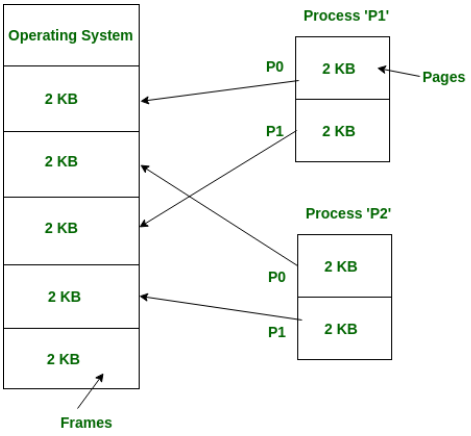
- **Frame Number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number frame
- **Frame Offset(d):** Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Non-Contiguous Allocation in Operating System



Non-contiguous allocation, also known as dynamic or linked allocation, is a memory allocation technique used in operating systems to allocate memory to processes that do not require a contiguous block of memory. In this technique, each process is allocated a series of non-contiguous blocks of memory that can be located anywhere in the physical memory.There are two fundamental approaches to implementing non-contiguous memory allocation. Paging and Segmentation

Paging



In paging, each process consists of fixed-size components called pages. The size of a page is defined by the hardware of a computer, and the demarcation of pages is implicit in it. The memory can accommodate an integral number of pages. It is partitioned into memory areas that have the same size as a page, and each of these memory areas is considered separately for allocation to a page.

```
Size of page in process = Size of frame in memory
```

Segmentation

In segmentation, a programmer identifies components called segments in a process. A segment is a logical entity in a program, e.g., a set of functions, data structures, or objects. Segmentation facilitates the sharing of code, data, and program modules processes. However, segments have different sizes, so the kernel has to use memory reuse techniques such as first-fit or best-fit allocation. Consequently, external fragmentation can arise.

Compaction

Compaction is a technique to collect all the free memory present in the form of fragments into one large chunk of free memory, which can be used to run other processes.
It does that by moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous.

Page Replacement Algorithm

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Optimal Page Replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Least Recently Used

In this algorithm, page will be replaced which is least recently used.

Most Recently Used (MRU)

In this algorithm, page will be replaced which has been used recently.
Belady's anomaly can occur in this algorithm.

Page Fault

A page fault is a type of interrupt or exception that occurs in a computer's operating system when a program attempts to access a page of memory that is not currently loaded into physical RAM (Random Access Memory). Instead, the page is stored on disk in a storage space called the page file or swap space.
Page fault = not hit in frames

Belady's Anomaly

Bélády's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.
This phenomenon is commonly experienced in the following page replacement algorithms:

- First in first out (FIFO)
- Second chance algorithm
- Random page replacement algorithm

A stack-based approach can be used to get rid of Belady's Algorithm.
Optimal Page Replacement Algorithm
Least Recently Used Algorithm (LRU)

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

Causes of thrashing:

1. High degree of multiprogramming.
2. Lack of frames.
3. Page replacement policy.

Techniques to handle:
1. Working Set Model
2. Page Fault Frequency

File System

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

- **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
- **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
- **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
- **HFS (Hierarchical File System):** A file system used by macOS.
- **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

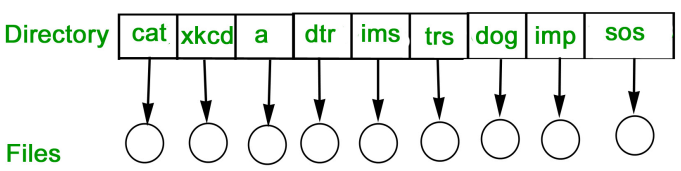
File Directories

The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Single-Level Directory

In this, a single directory is maintained for all the users.

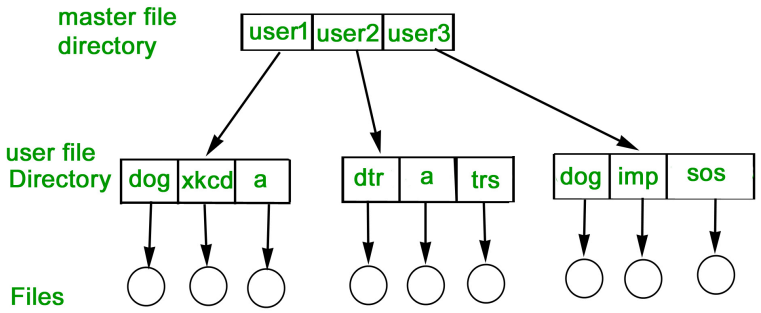
- **Naming Problem:** Users cannot have the same name for two files.
- **Grouping Problem:** Users cannot group files according to their needs



Two-Level Directory

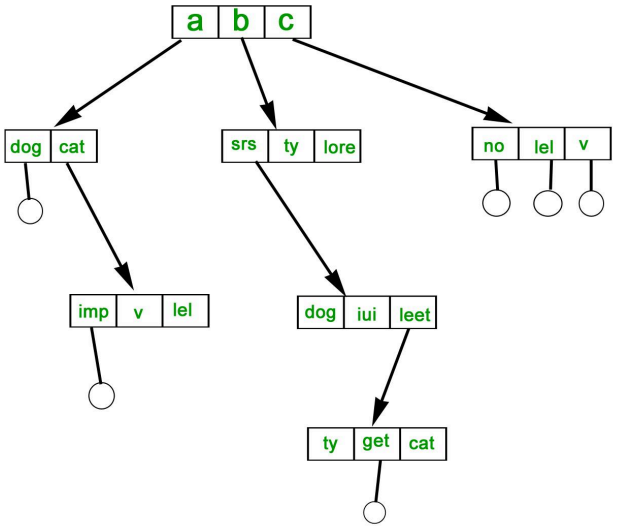
In this separate directories for each user is maintained.

- **Path Name**: Due to two levels there is a path name for every file to locate that file.
- Now, we can have the same file name for different users.
- Searching is efficient in this method.



Tree-Structured Directory

The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



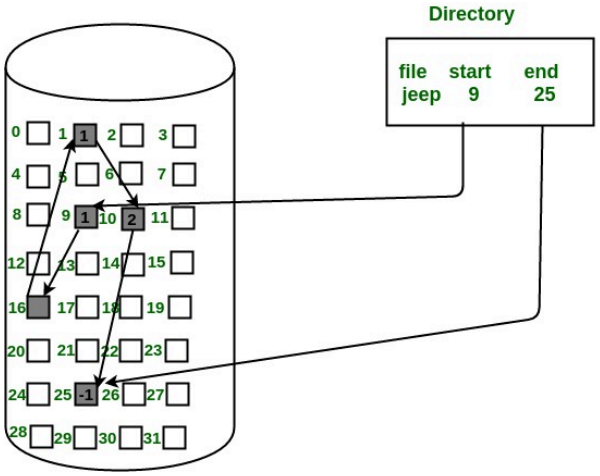
File Allocation Methods

1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,.....b+n-1*.

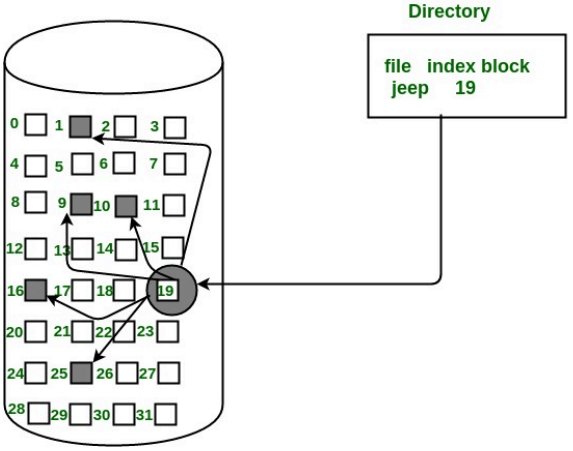
2. Linked List Allocation

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



3. Indexed Allocation

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



Disk Free Space Management

Bit Tables: This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use
Free Block List: In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.

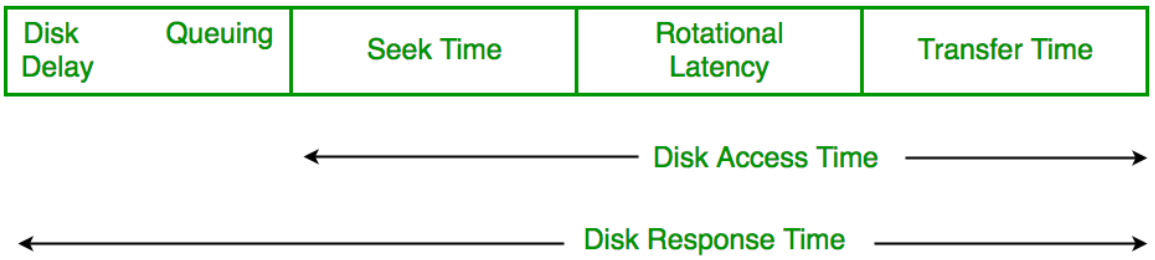
Disk Scheduling Algorithms

Disk scheduling is a technique operating systems use to manage the order in which disk I/O (input/output) requests are processed. Disk scheduling is also known as I/O Scheduling.Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written. So the disk scheduling algorithm that gives a minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of the disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and the number of bytes to be transferred.
- **Disk Access Time:**

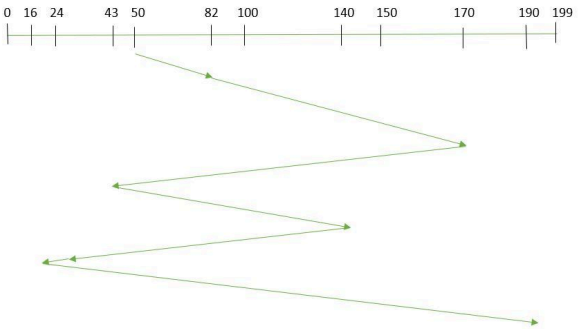
Disk Access Time = Seek Time + Rotational Latency + Transfer Time

*Total Seek Time = Total head Movement * Seek Time*



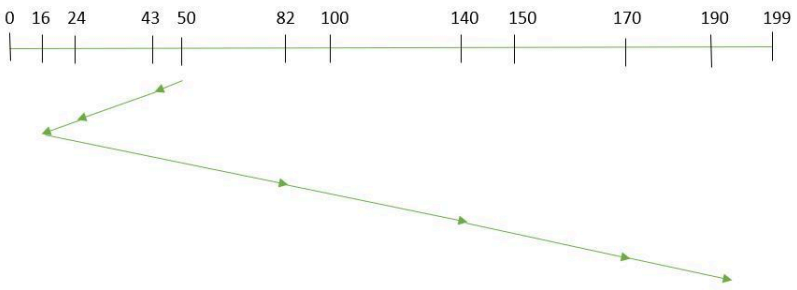
1. FCFS (First Come First Serve)

FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.



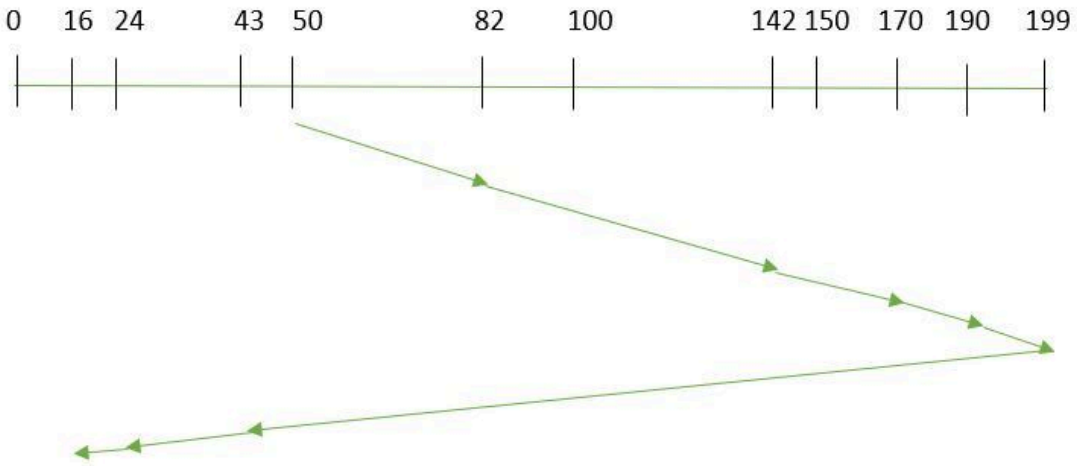
2. SSTF (Shortest Seek Time First)

In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.

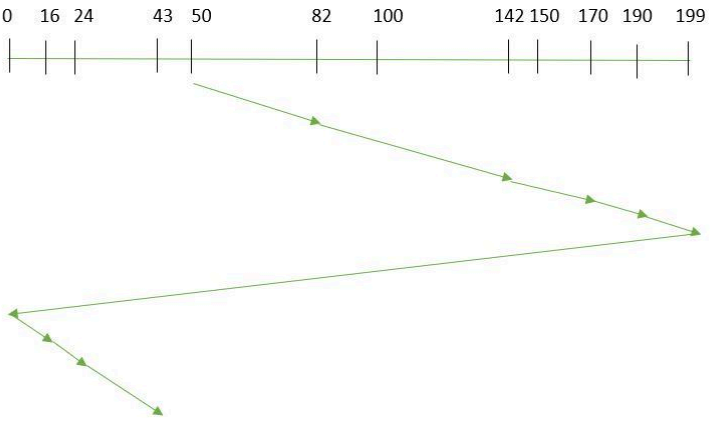


3. SCAN

In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path.

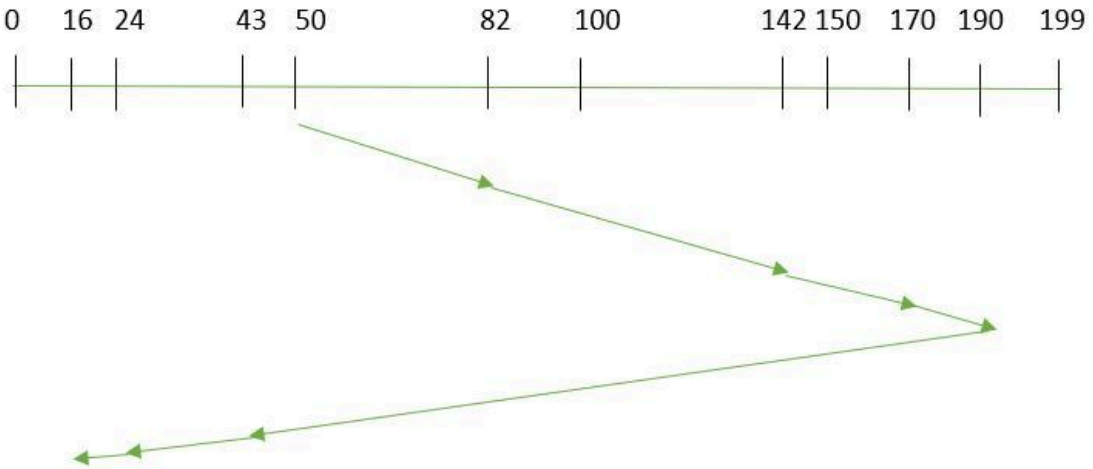


4. C-SCAN

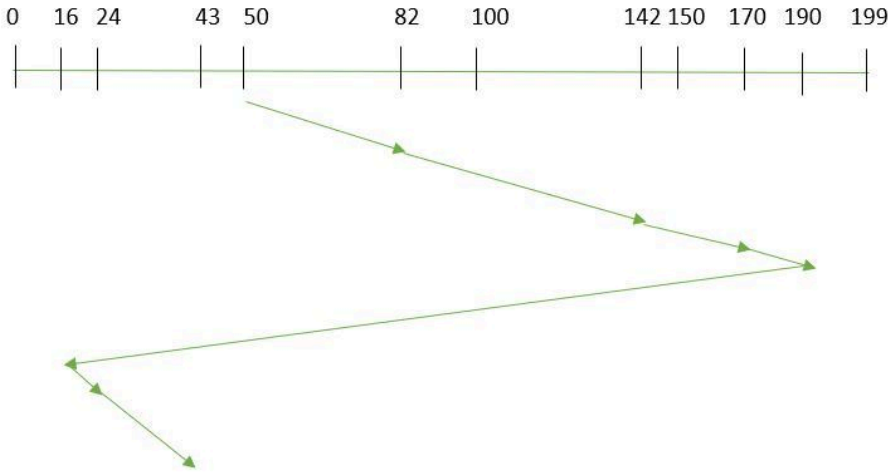


5. LOOK

LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.



6. C-LOOK



7. RSS (Random Scheduling)

It stands for Random Scheduling and just like its name it is natural. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm sits perfectly. Which is why it is usually used for analysis and simulation.

8. LIFO (Last-In First-Out)

In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is newest or last entered is serviced first, and then the rest in the same order.

9. N-STEP SCAN

It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N request and this way all get requests to get a guaranteed service

10. F-SCAN

This algorithm uses two sub-queues. During the scan, all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

RAID (Redundant Arrays of Independent Disks)

RAID (Redundant Arrays of Independent Disks) is a technique that makes use of a combination of multiple disks for storing the data instead of using a single disk for increased performance, data redundancy, or to protect data in the case of a drive failure. RAID (Redundant Array of Independent Disks) is like having backup copies of your important files stored in different places on several hard drives or solid-state drives (SSDs). If one drive stops working, your data is still safe because you have other copies stored on the other drives.

Types of RAID Controller

Hardware Based: In hardware-based RAID, there's a physical controller that manages the whole array. This controller can handle the whole group of hard drives together. It's designed to work with different types of hard drives, like SATA (Serial Advanced Technology Attachment) or SCSI (Small Computer System Interface). Sometimes, this controller is built right into the computer's main board, making it easier to set up and manage your RAID system. It's like having a captain for your team of hard drives, making sure they work together smoothly**.**

Software Based: In software-based RAID, the controller doesn't have its own special hardware. So it use computer's main processor and memory to do its job. It perform the same function as a hardware-based RAID controller, like managing the hard drives and keeping your data safe. But because it's sharing resources with other programs on your computer, it might not make things run as fast. So, while it's still helpful, it might not give you as big of a speed boost as a hardware-based RAID system

Firmware Based: Firmware-based RAID controllers are like helpers built into the computer's main board. They work with the main processor, just like software-based RAID. But they only implement when the computer starts up. Once the operating system is running, a special driver takes over the RAID job. These controllers aren't as expensive as hardware ones, but they make the computer's main processor work harder. People also call them hardware-assisted software RAID, hybrid model RAID, or fake RAID.

Different RAID Levels

1. RAID-0 (Stripping)

Disk 0	Disk 1	Disk 2	Disk 3
0	3	4	6
1	3	5	7
8	10	12	14
9	11	13	15

2. RAID-1 (Mirroring)

RAID 1

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

3. RAID-2 (Bit-Level Stripping with Dedicated Parity)
   In Raid-2, the error of the data is checked at every bit level. Here, we use Hamming Code Parity Method to find the error in the data.

4. RAID-3 (Byte-Level Stripping with Dedicated Parity)

RAID 3

Disk 0	Disk 1	Disk 2	Disk 3
15	16	17	P(15, 16, 17)
18	19	20	P(18, 19, 20)
21	22	23	P(21, 22, 23)
24	25	26	P(24, 25, 26)

5. RAID-4 (Block-Level Stripping with Dedicated Parity)

RAID 4

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

6. RAID-5 (Block-Level Stripping with Distributed Parity)

RAID 5

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

7. RAID-6 (Block-Level Stripping with two Parity Bits)

RAID 6

Disk 1	Disk 2	Disk 3	Disk 4
A1	B1	P(B1)	P(B1)
A2	P(B2)	P(B2)	B2
P(B3)	P(B3)	A3	B3
P(B4)	A4	A4	P(B4)