KDKCE,Nagpur

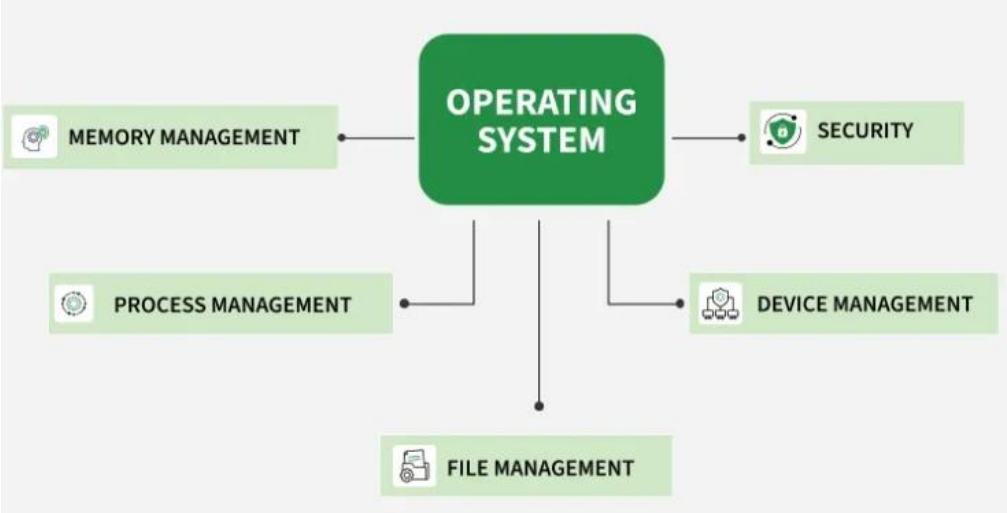
Department of Information Technology

Operating System

Unit – 1

An Operating System (OS) is software that acts as a bridge between the user and the computer hardware. It provides an environment where users can execute programs efficiently and conveniently.

- Manages and controls all computer resources.
- Provides services and resources to user programs.
- Coordinates and monitors program execution to prevent errors.
- Offers a user interface (virtual machine-Install Hyperviosr S/W divide main resources like RAM, HDD and shares the resources) and hides software complexity.
- Supports multiple execution modes for programs.

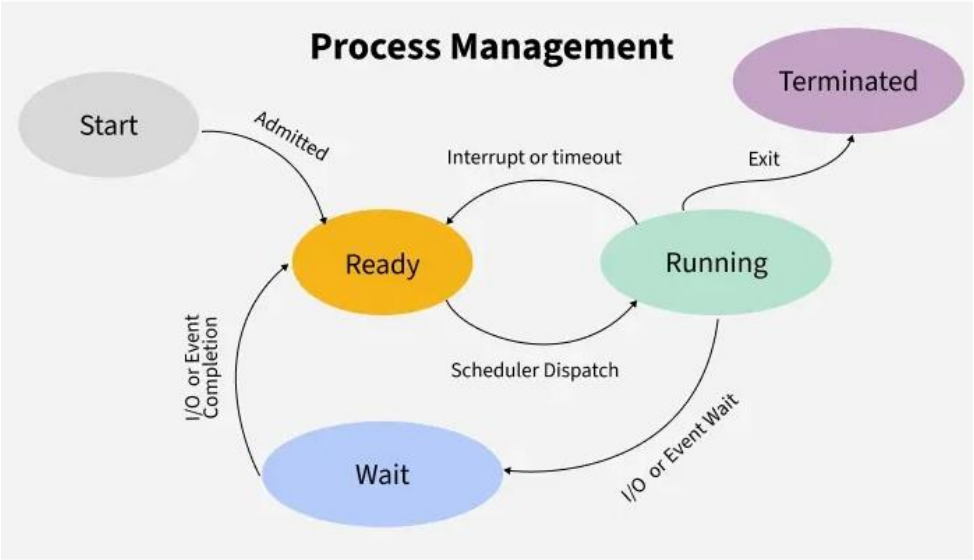


The OS controls all computer resources, coordinates program execution, hides system complexity, ensures security, and improves resource utilization.

Functions of an Operating System

1. Process Management

Process management in operating system is about managing processes.
A Process is a running program. The life cycle of process is from the moment program start until it finishes.



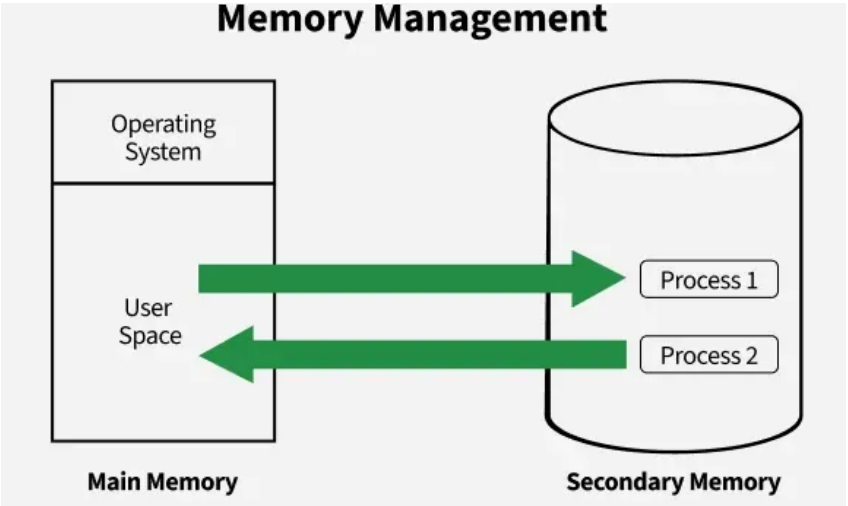
Process State Diagram

Core Functions in Process Management:

- **Scheduling:** Decides which process uses the CPU next (using algorithms like Round Robin or Priority Scheduling).
- **Synchronization:** Ensures orderly execution using locks, semaphores, or monitors to prevent race conditions.
- **Deadlock Handling:** Detects and prevents situations where processes wait forever for resources.
- **Inter-Process Communication (IPC):** Allows processes to exchange data via shared memory or message passing.

2. Memory Management

Memory management in an operating system controls how data is stored and organized in main (primary) memory and secondary storage, ensuring programs get the memory they need and freeing it when no longer used.



Key Activities in Memory Management:

- **Allocation & Deallocation:** Assigns and frees memory as needed.
- **Protection:** Prevents processes from interfering with each other's memory.
- **Virtual Memory:** Uses disk space as extra memory to run large programs.
- **Fragmentation Handling:** Reduces wasted space through compaction.

- **Disk Management:** Handles file storage, free space, disk scheduling, and backups.

3. File System Management

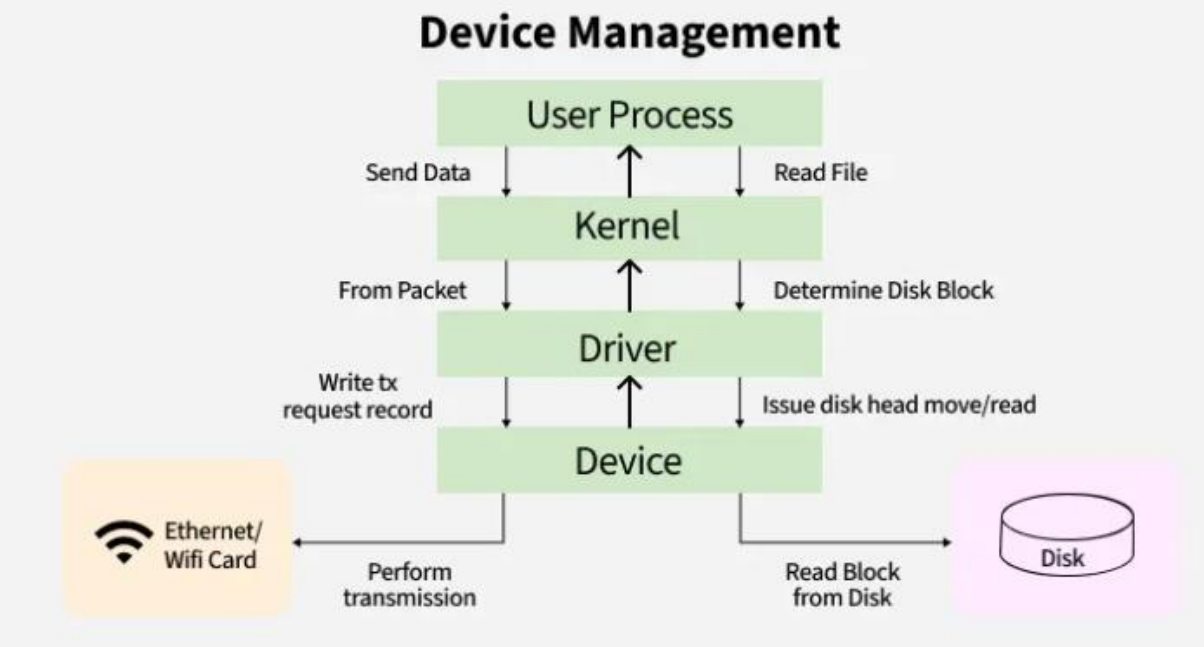
File management in an operating system organizes and controls how data is stored, named, and accessed on storage devices, making it easy for users to find and use their files.

File System Management includes managing of:

- **File Attributes:** Name, type, size, and permissions.
- **File Types:** Text, binary, and executable files.
- **Operations:** Create, read, write, and delete files.
- **Access Methods:** Sequential, direct, and indexed access for faster retrieval.

4. Device Management (I/O System)

Device management in an operating system controls how the computer communicates with hardware devices like printers or disks, using drivers and techniques to ensure smooth and efficient operation.



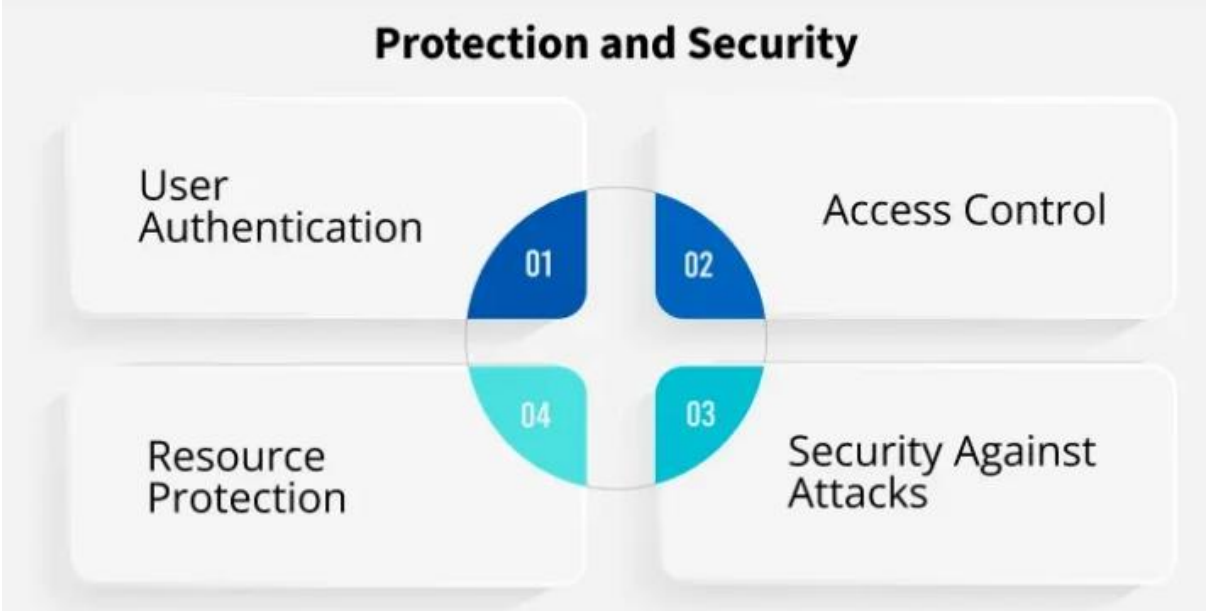
Major components in Device Management:

- **Device Drivers:** Interface between hardware and OS.

- **Buffering & Caching:** Temporarily store data to match device speeds and improve performance.
- **Spooling:** Spooling manages data waiting to be processed, particularly in devices like printers. The OS places print jobs in a spool (a temporary storage area), allowing the CPU to continue other tasks while the printer works through the queue.

5. Protection and Security

Protection and security mechanisms in an OS are designed to safeguard system resources from unauthorized access or misuse. These mechanisms control which processes or users can access specific resources (such as memory, files, and CPU time). It also ensures that only authorized users can perform specific actions.



The OS protects system resources and user data from unauthorized access and attacks.

- **Access Control:** Limits user and process permissions.
- **Authentication:** Verifies users through credentials (UIDs/SIDs).
- **Resource Protection:** Prevents misuse of files, memory, or devices.
- **Security:** Guards against malware, denial-of-service attacks, and data theft.

Additional Functions

Beyond core tasks like process and memory management, OS also focus on:

- **Performance Monitoring:** Tracks system efficiency and identifies bottlenecks.
- **Job Accounting:** Records resource usage for auditing and optimization.
- **Error Detection:** Generates error logs, traces, and dumps to detect and fix issues.

An operating system is a type of software that acts as an interface between the user and the hardware. It is responsible for handling various critical functions of the computer and utilizing resources very efficiently so the operating system is also known as a resource manager. The operating system also acts like a government because just as the government has authority over

everything, similarly the operating system has authority over all resources. Various tasks that are handled by OS are file management, task management, garbage management, memory management, process management, disk management, I/O management, peripherals management, etc.

Evolution of Operating System

Generations of Operating Systems

- **1940s-1950s: Early Beginnings**
    - Computers operated without operating systems (OS).
    - Programs were manually loaded and run, one at a time.
    - The first operating system was introduced in 1956. It was a batch processing system GM-NAA I/O (1956) that automated job handling.
- **1960s: Multiprogramming and Timesharing**
    - Introduction of multiprogramming to utilize CPU efficiently.
    - Timesharing systems, like CTSS (1961) and Multics (1969), allowed multiple users to interact with a single system.
- **1970s: Unix and Personal Computers**
    - Unix (1971) revolutionized OS design with simplicity, portability, and multitasking.
    - Personal computers emerged, leading to simpler OSs like CP/M (1974) and PC-DOS (1981).
- **1980s: GUI and Networking**
    - Graphical User Interfaces (GUIs) gained popularity with systems like Apple Macintosh (1984) and Microsoft Windows (1985).
    - Networking features, like TCP/IP in Unix, became essential.
- **1990s: Linux and Advanced GUIs**
    - Linux (1991) introduced open-source development.
    - Windows and Mac OS refined GUIs and gained widespread adoption.
- **2000s-Present: Mobility and Cloud**
    - Mobile OSs like iOS (2007) and Android (2008) dominate.
    - Cloud-based and virtualization technologies reshape computing, with OSs like Windows Server and Linux driving innovation.
- **AI Integration - (Ongoing)**
  With the growth of time, Artificial intelligence came into picture. Operating system integrates features of AI technology like Siri, Google Assistant, and Alexa and became more powerful and efficient in many way. These AI features with operating system create a entire new feature like voice commands, predictive text, and personalized recommendations.

Evolution of Operating Systems

Mainframe Systems:

- Early systems used resident monitors for job sequencing, reducing setup time.

Batch Processing:

- Grouped similar jobs to minimize setup time, improving efficiency.

Multiprocessor Systems:

- **Symmetric Multiprocessing:** Identical OS copies run on multiple processors, communicating as needed.
- **Asymmetric Multiprocessing:** A master processor controls slave processors.

Real-Time Systems:

- **Hard Real-Time:** Strict timing constraints for critical applications.
- **Soft Real-Time:** Prioritizes critical tasks with less stringent timing.

Characteristics of Operating System

Below are the detailed characteristics of Operating System:

1. **Concurrency:**
   An Operating System should be able to handle multiple tasks or processes at the same time, allowing users to multitask and improve system efficiency.

2. **Hardware Abstraction:**
   An operating system should provide hardware abstraction, which means it should provide a uniform interface to hardware devices, making it easier for software applications to interact with hardware devices without having to know the underlying hardware details.

3. **Resource Allocation:**
   An operating system should allocate system resources fairly so that each process gets access to the resources it needs to function correctly. It should also allocate resources dynamically, based on the current system load and user demand.

4. **Virtualization:**
   An operating system create virtual resources i.e virtual memory and virtual CPUs, in order to enhance the efficiency of system and also to enable multiple processes to run at the same time.

5. **Security:**
   An operating system provides security features like protection against unauthorized access, file encryption of sensitive data, firewalls, and antivirus softwares. It also provides features like backup and recovery to ensure data availability in case of data loss or system failure.

6. **Fault Tolerance:**
   An operating system is designed in such a way that it is able to handle hardware and software failures and performs error detection and recovery mechanisms in order to minimize system downtime and data loss.

7. **Scalability:**
   An operating system is capable to scale from small embedded devices to large

servers and clusters, providing efficient resource management and performance regardless of the system size.

8. **Compatibility:**
   An operating system is compatible with a wide range of hardware and software devices, allowing software applications to run on different hardware platforms.

9. **Customizability:**
   An operating system should be customizable, enabling users to tailor the system to their needs and preferences, such as changing the UI, adding or removing system features, or modifying system settings.

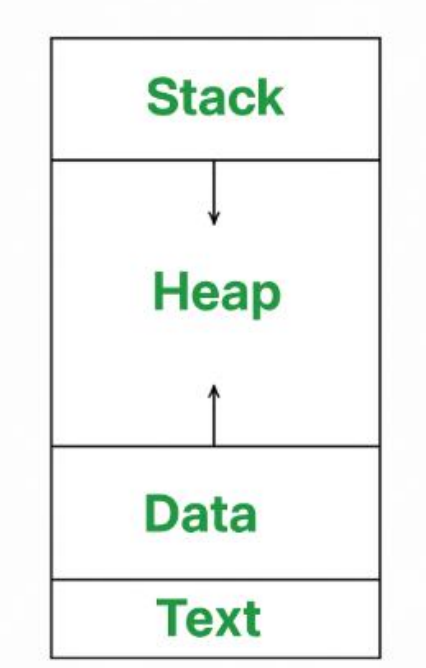
10. **Ease of Use:**
    An operating system is user-friendly and very easy to use, providing a simple and intuitive interface for users with no or less computer knowledge to interact with the system and run applications.

A process is a program in execution. For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process.

- A process is an 'active' entity instead of a program, which is considered a 'passive' entity.
- A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created). .

How Does a Process Look Like in Memory?

A process in memory is divided into several distinct sections, each serving a different purpose. Here's how a process typically looks in memory.



- **Text Section**: A text or code segment contains executable instructions. It is typically a read only section

- **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.

Attributes of a Process

A **process** has several important attributes that help the operating system manage and control it. These attributes are stored in a structure called the **Process Control Block (PCB)** (sometimes called a task control block). The PCB keeps all the key information about the process, including:

1. **Process ID (PID)**: A unique number assigned to each process so the operating system can identify it.
2. **Process State**: This shows the current status of the process, like whether it is running, waiting, or ready to execute.
3. **Priority and other CPU Scheduling Information**: Data that helps the operating system decide which process should run next, like priority levels and pointers to scheduling queues.
4. **I/O Information**: Information about input/output devices the process is using.
5. **File Descriptors:** Information about open files and network connections.
6. **Accounting Information**: Tracks how long the process has run, the amount of CPU time used, and other resource usage data.
7. **Memory Management Information**: Details about the memory space allocated to the process, including where it is loaded in memory and the structure of its memory layout (stack, heap, etc.).

Files System Calls

A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed. System Calls are,

- A way for programs to interact with the operating system.
- Provide the services of the operating system to the user programs.
- Only entry points into the kernel and are executed in kernel mode.

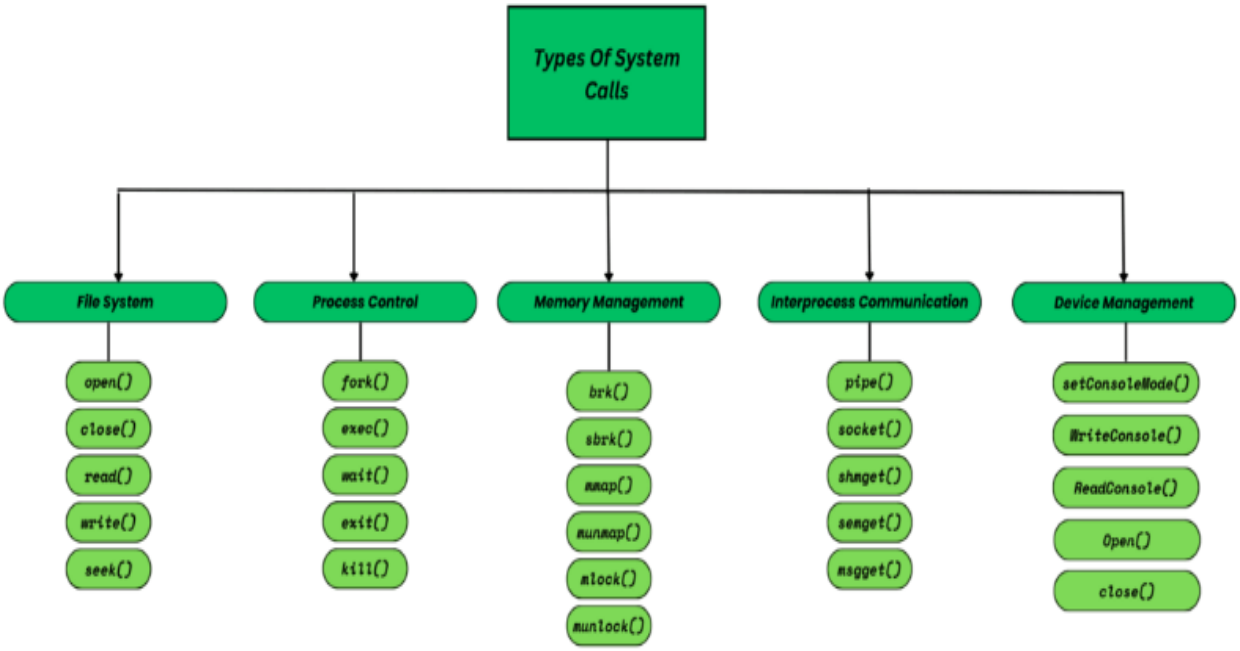
System Call

How do System Calls Work?

- A system call allows a program to request services from the operating system.
- It can be written in high-level languages (C, C++, Pascal) or in assembly.
- When a program makes a system call, it switches to kernel mode.
- The OS handles the request, performs the task (e.g., file access, process control), and returns the result.
- Without system calls, each program would need its own way to access hardware, making systems inconsistent and error-prone.

Types of System Calls

Services provided by an OS are typically related to any kind of operation that a user program can perform like creation, termination, forking, moving, communication, etc. Similar types of operations are grouped into one single system call category. System calls are classified into the following categories:



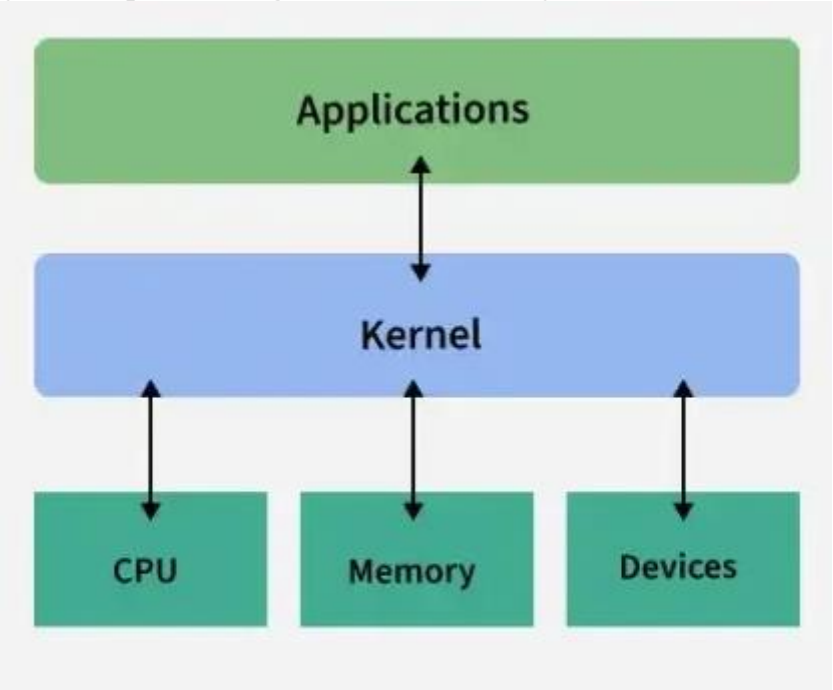
- **File System:** Used to create, open, read, write, and manage files and directories.
- **Process Control:** Used to create, execute, synchronize, and terminate processes.
- **Memory Management:** Used to allocate, deallocate, and manage memory for processes.
- **Interprocess Communication (IPC):** Used for data exchange and communication between different processes.
- Device Management: Used to request and release devices, and to perform read/write operations on them.

What is OS Architecture?

The **architecture of an operating system** refers to the internal structure and design that defines how the OS manages resources and provides services to users and applications. It organizes the system into different layers, each with a specific role:

- **Hardware layer —** physical devices like CPU, memory, disk, and I/O devices.
- **Kernel layer** — the core of the OS that directly interacts with hardware.
- **Shell layer** — the interface (command line or GUI) through which users interact with the system.
- **System calls** — special functions that connect applications with kernel services.

Kernel, shell, and system calls work together as the very heart of an operating system, powering its functionality and user interaction.



Kernel (Operating System)

Kernel in Operating System

*"A **kernel** is the core part of an operating system. It acts as a bridge between software applications and the hardware of a computer."*

The **kernel** is the **most crucial part of an OS** as it directly interacts with hardware and manages system resources. It provides essential services for processes and applications.

Key Functions of the Kernel:

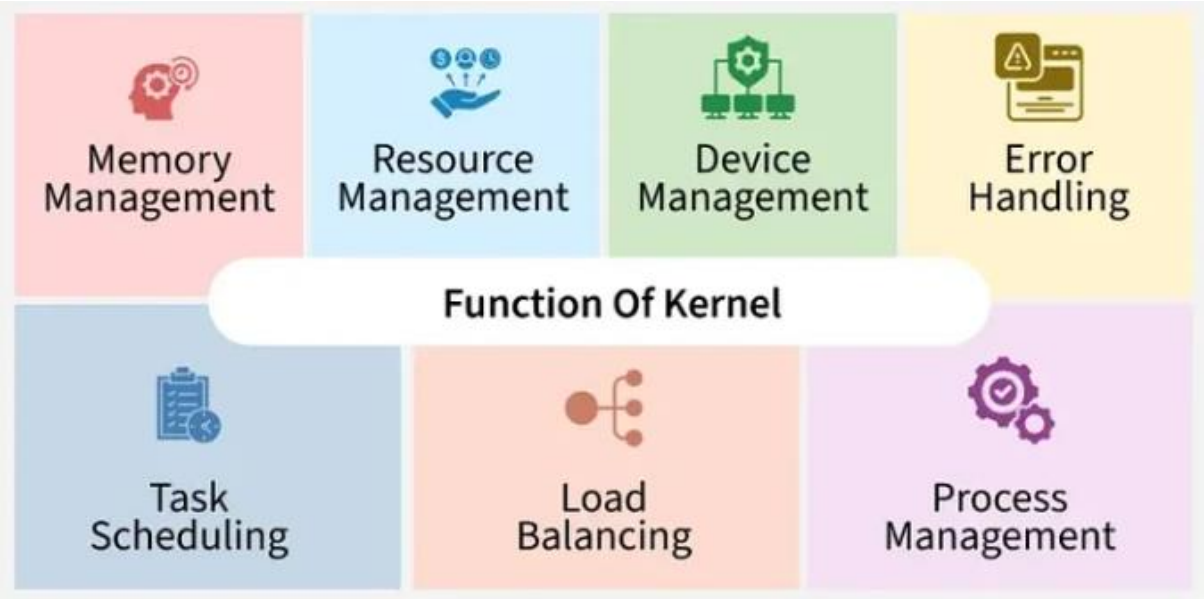
✅ Process Management — Schedules and executes processes using CPU scheduling algorithms (FCFS, Round Robin, Priority Scheduling).

✅ Memory Management — Allocates and deallocates memory, implements virtual memory, paging, segmentation.

✅ Device Management — Manages I/O devices through device drivers.

✅ File System Management — Organizes files and directories, ensures security and permissions.

✅ Interrupt Handling — Handles hardware and software interrupts to ensure smooth execution.



Types of Kernels

Press enter or click to view image in full size

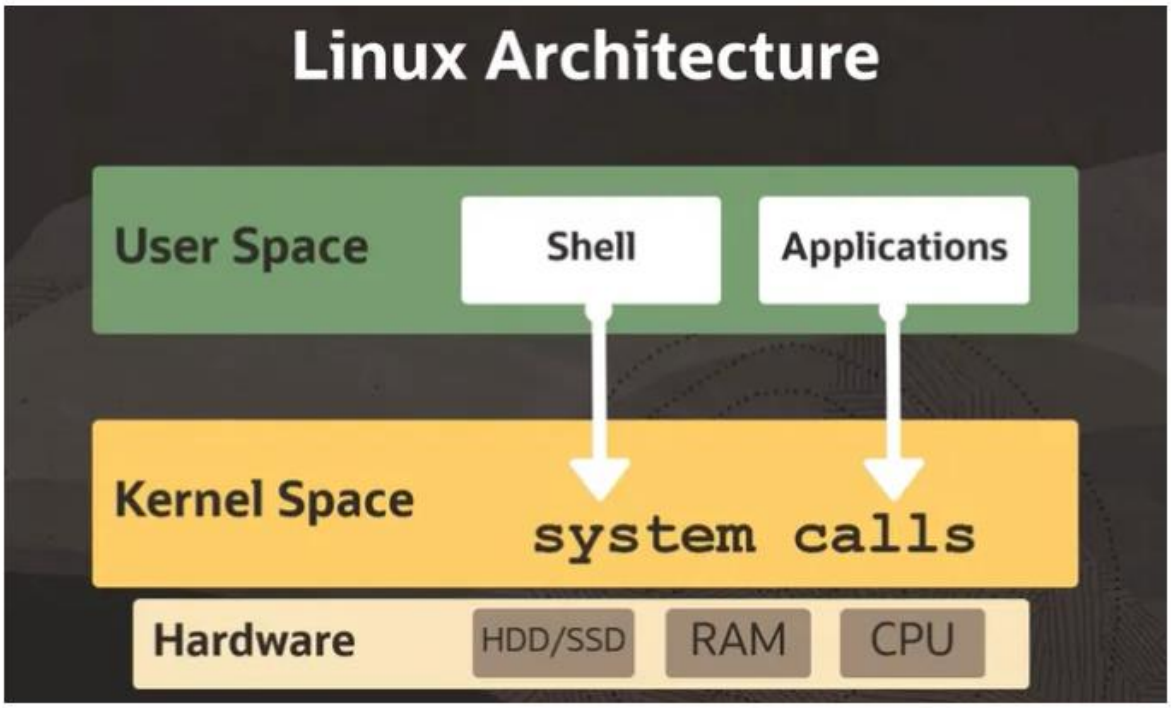
Kernel Type	Description	Example OS
Monolithic Kernel	Single large program, faster but difficult to modify.	Linux, UNIX
Microkernel	Minimalist design, only essential functions run in kernel mode.	QNX, Minix
Hybrid Kernel	Combination of monolithic and microkernel.	Windows, macOS
Exokernel	Gives direct hardware access to applications.	Experimental OS designs

Working of Kernel

- The kernel is the first part of the OS loaded into memory during boot, and it stays resident while the system is running.
- It operates in a privileged mode (kernel mode), separate from user mode for applications; user apps can't directly access hardware or critical resources.
- Applications make requests to the kernel via *system calls* (or software interrupts). The kernel handles these by switching from user mode to kernel mode.

- Kernel executes the requested operation (e.g. file I/O, process creation, memory allocation).
- On completion, kernel returns result (or error) to user space.

Example: Linux kernel architecture



Shell in Operating System

The ***Shell*** is an interface that allows users to interact with the OS. It acts as a bridge between the user and the kernel by processing commands.

Types of Shells:

Shell Type	Description	Example OS
Command Line Interface (CLI)	Text-based interface, efficient but requires knowledge of commands.	Linux (Bash), Windows (PowerShell)
Graphical User Interface (GUI)	Visual interface with icons, windows, and buttons.	Windows, macOS
Voice-Based Interface	Uses voice commands for interaction.	Siri, Alexa

Example:

- The Bash shell in Linux allows users to type commands to interact with the system.

- The Windows GUI provides a graphical way to manage files and programs.

Interaction Between Kernel and Shell

1. User enters a command in the Shell (e.g., ls to list files).
2. Shell interprets the command and sends a request to the Kernel.
3. Kernel processes the request (accessing files, executing processes, etc.).
4. Kernel sends output back to the Shell.
5. Shell displays the result to the User.

Example:

When a user opens a file in Windows, the GUI Shell sends a request to the Kernel, which retrieves the file and displays it.

*In simple terms, **Kernel** is the heart of the OS, managing resources and hardware whereas the **Shell** is the user interface, allowing interaction with the system.*

System Calls in Operating System

*A **system call** is an interface between a program running in user space and the operating system (OS).* Application programs use system calls to request services and functionalities from the OS's kernel. This mechanism allows the program to call for a service, like reading from a file, without accessing system resources directly.

When a program invokes a system call, the execution context switches from user to kernel mode, allowing the system to access hardware and perform the required operations safely. After the operation is completed, the control returns to user mode, and the program continues its execution.

This layered approach facilitated by system calls:

- Ensures that hardware resources are isolated from user space processes.
- Prevents direct access to the kernel or hardware memory.
- Allows application code to run across different hardware architectures.

How Do System Calls Work?

1. **System Call Request**. The application requests a system call by invoking its corresponding function. For instance, the program might use the `read()` function to read data from a file.
2. **Context Switch to Kernel Space**. A software interrupt or special instruction is used to trigger a context switch and transition from the user mode to the kernel mode.
3. **System Call Identified**. The system uses an index to identify the system call and address the corresponding kernel function.

Disk Scheduling

Disk scheduling algorithms manage how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed.

- Disk scheduling is also known as I/O Scheduling.
- The main goals of disk scheduling are to optimize the performance of disk operations, reduce the time it takes to access data and improve overall system efficiency.
- Common disk scheduling methods include First-Come, First-Served
  1. (FCFS)
  2. Shortest Seek Time First (SSTF)
  3. SCAN
  4. C-SCAN
  5. LOOK
  6. C-LOOK.

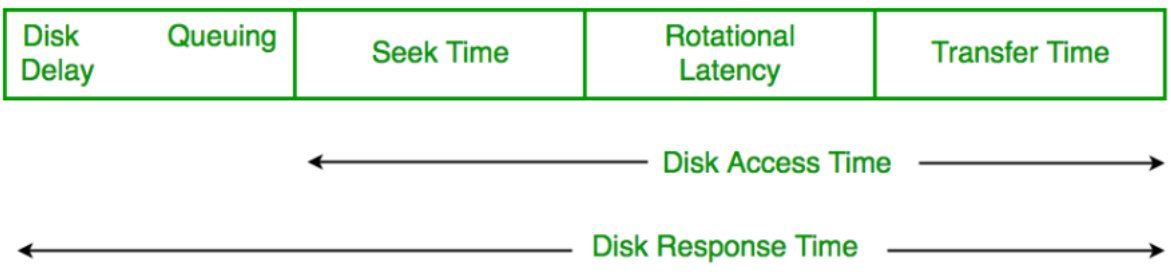
Importance of Disk Scheduling in Operating System

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more requests may be far from each other so this can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Key Terms Associated with Disk Scheduling

- **Seek Time:** Time taken to move the disk arm to the track where data is located.
- **Rotational Latency:** Time taken for the desired sector to rotate under the read/write head.
- **Transfer Time:** Time taken to actually read or write the data, depending on disk speed and data size.

Disk Access Time = Seek Time + Rotational Latency + Transfer Time
Total Seek Time = Total head Movement * Seek Time



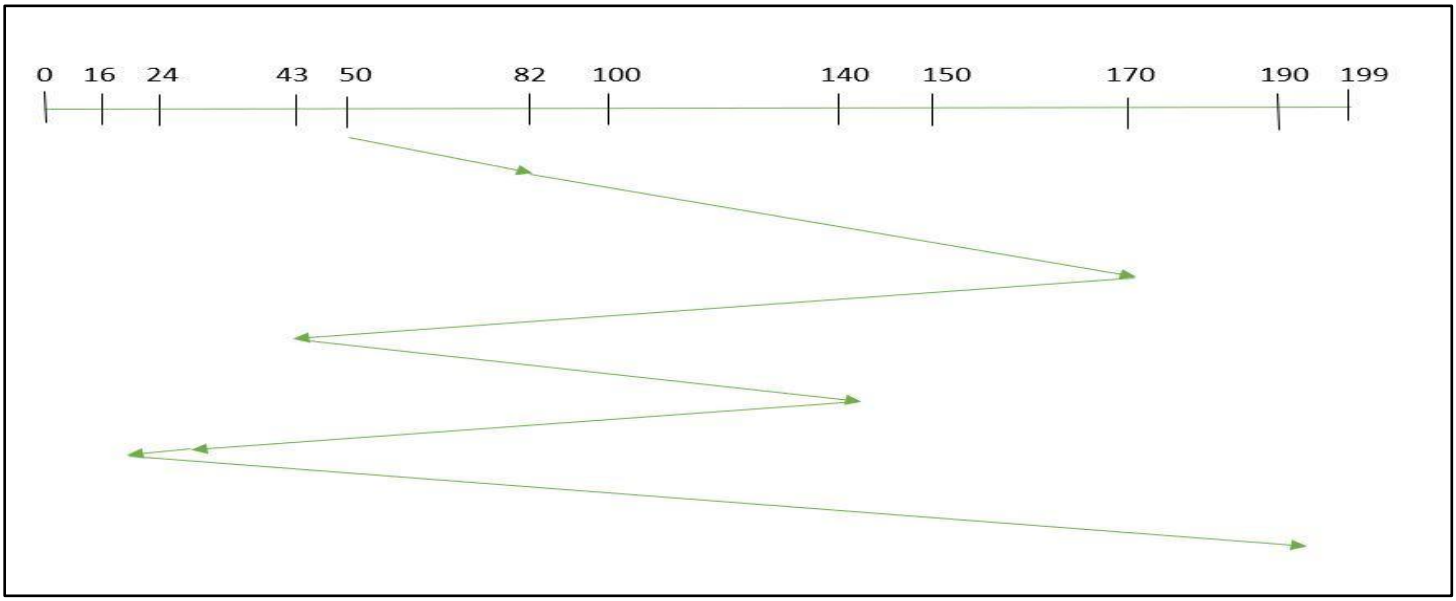
Disk Access Time and Disk Response Time

Goals of Disk Scheduling Algorithms

- Minimize Seek Time
- Maximize Throughput
- Minimize Latency
- Ensuring Fairness
- Efficiency in Resource Utilization

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. Let us understand this with the help of an example.



First Come First Serve

Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50

So, total overhead movement (total distance covered by the disk arm) = (82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

Advantages of FCFS

Here are some of the advantages of First Come First Serve.

- Every request gets a fair chance
- No indefinite postponement

Disadvantages of FCFS

Here are some of the disadvantages of First Come First Serve.

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

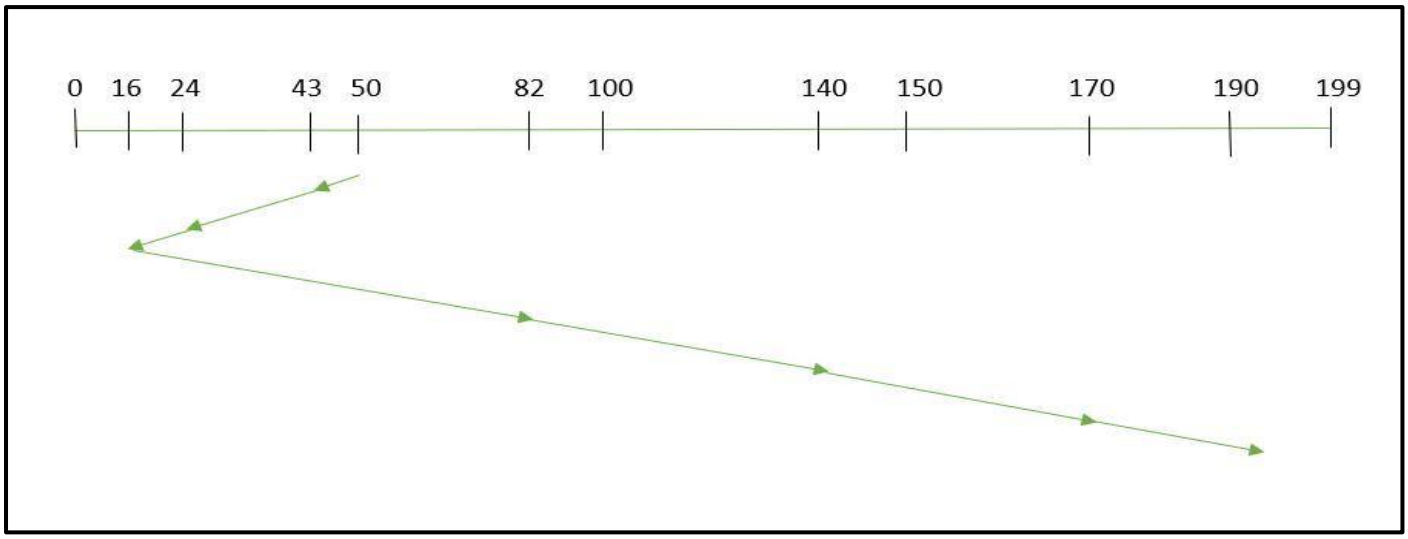
Learn more in detail: FCFS

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. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time

and increases the throughput of the system. Let us understand this with the help of an example.

Example:



Shortest Seek Time First

Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50

total overhead movement (total distance covered by the disk arm) = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

Advantages of Shortest Seek Time First

Here are some of the advantages of Shortest Seek Time First.

- The average Response Time decreases
- Throughput increases

Disadvantages of Shortest Seek Time First

Here are some of the disadvantages of Shortest Seek Time First.

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

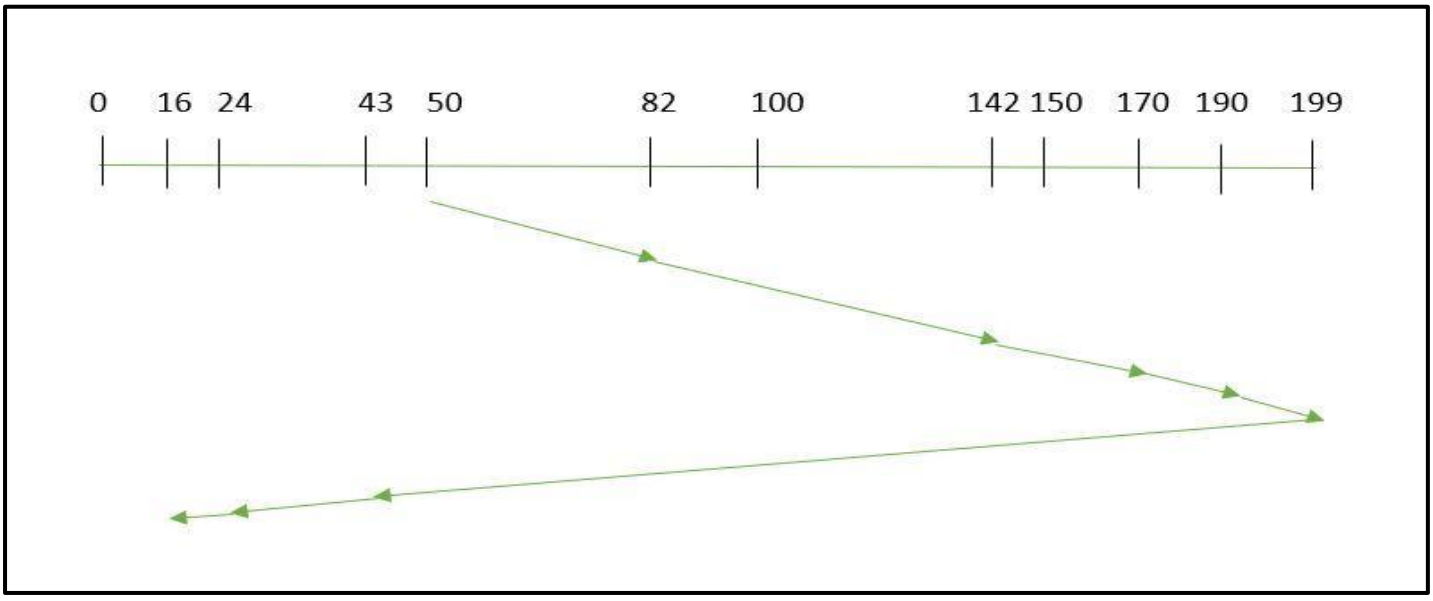
Learn More in detail: shortest seek time first

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. So, this algorithm works as an elevator and is hence also known as an **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

SCAN Algorithm



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

Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as

= (199-50) + (199-16) = 332

Advantages of SCAN Algorithm

Here are some of the advantages of the SCAN Algorithm.

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

Disadvantages of SCAN Algorithm:

- Head may move to disk end unnecessarily
- High waiting time for some requests
- New requests may wait longer

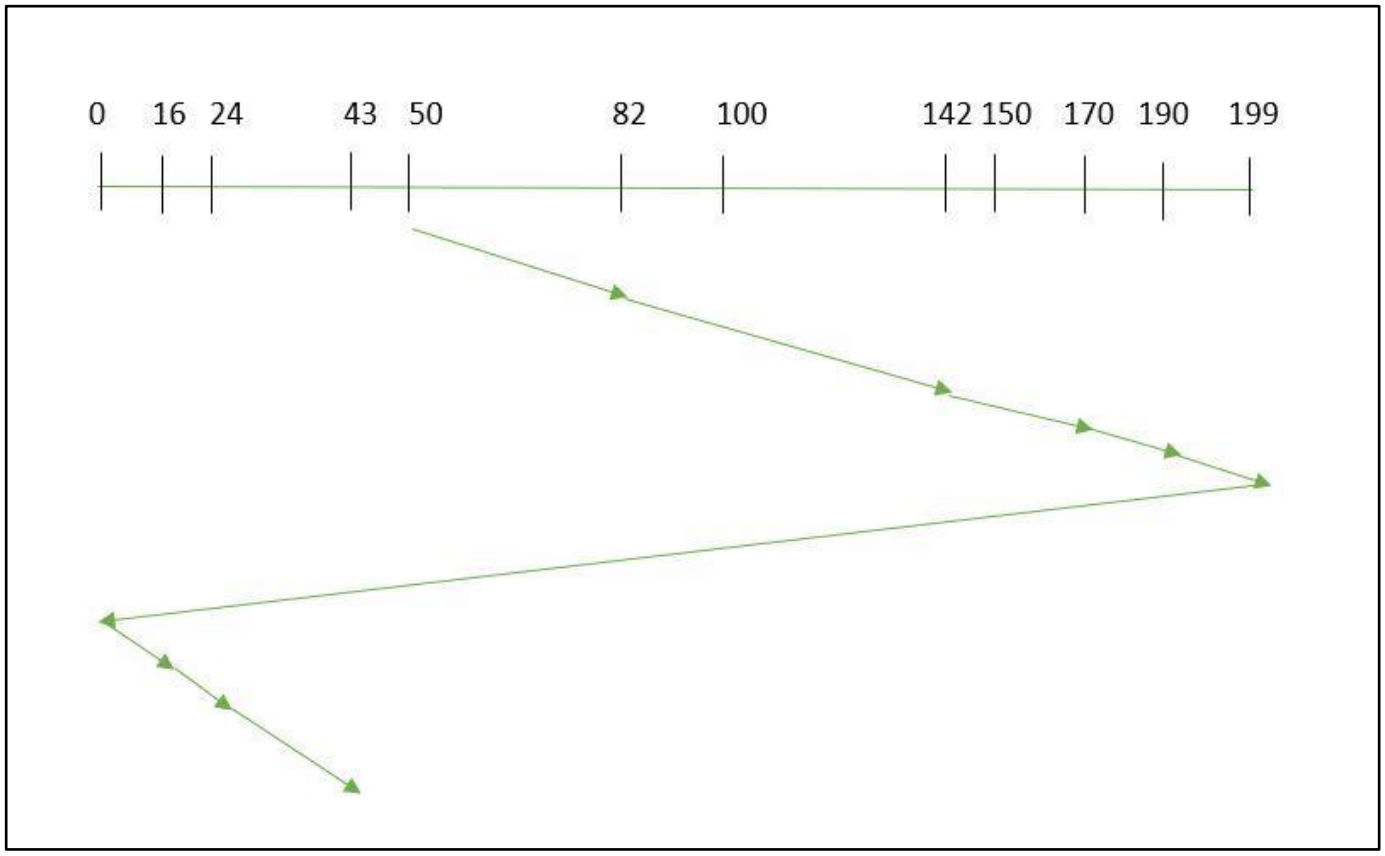
Learn More in detail: SCAN

4. C-SCAN

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

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

Example:



Circular SCAN

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

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50) + (199-0) + (43-0) = 391

Advantages of C-SCAN Algorithm:

- Eliminates starvation of requests
- Better performance for heavy disk load
- More fair than SCAN algorithm

Learn more in detail: ***C-SCAN***

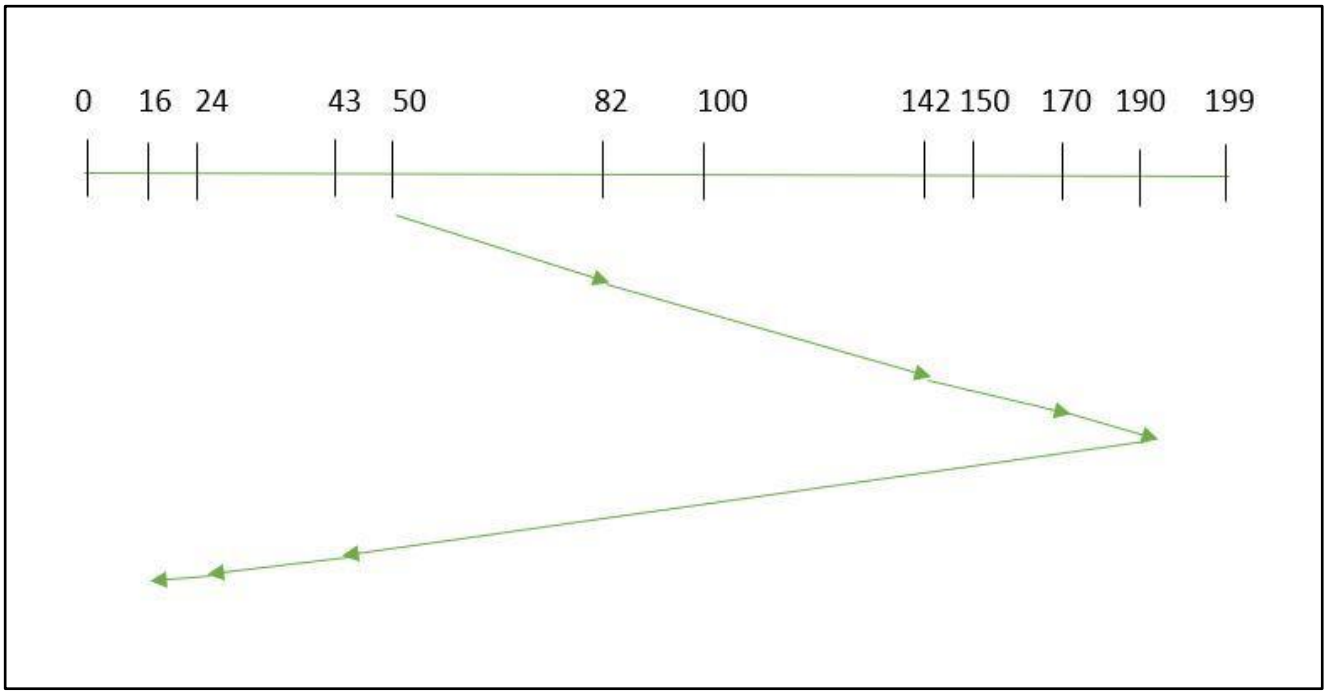
Disadvantages of C-SCAN Algorithm:

- Extra head movement due to return to start
- Increased seek time compared to SCAN in light load
- More overhead because of circular movement

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. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:



LOOK Algorithm

Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".
So, the total overhead movement (total distance covered by the disk arm) is calculated as:
= (190-50) + (190-16) = 314

Advantages

- **Reduced Unnecessary Movement:** The disk arm only goes as far as the last request in each direction, avoiding travel to the disk's physical end (unlike SCAN).
- **Faster Response:** Less head movement leads to quicker service for requests.

Disadvantages of LOOK Algorithm:

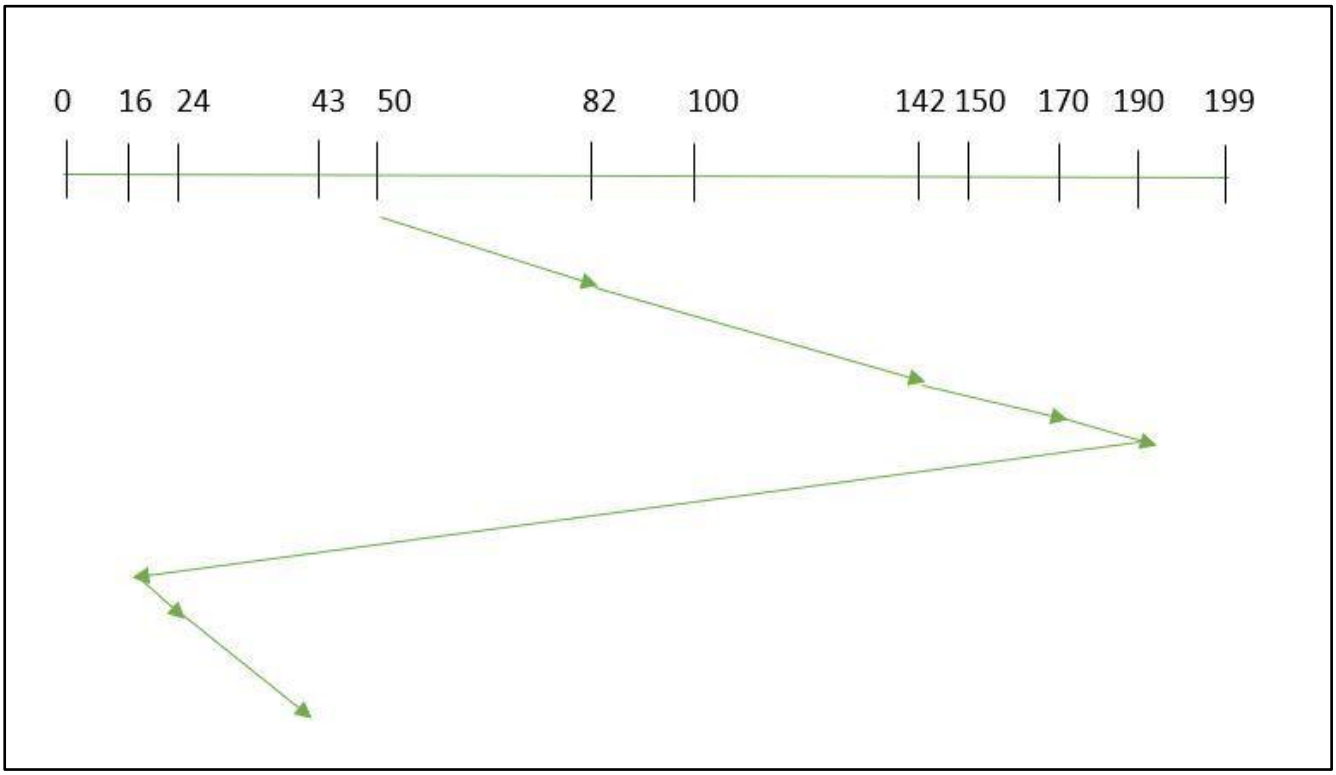
- Waiting time can still be high for some requests
- Performance depends on request distribution
- Not completely fair to newly arrived requests

Learn more in detail: LOOK

6. C-LOOK

As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

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



C-LOOK

So, the total overhead movement (total distance covered by the disk arm) is calculated as

= (190-50) + (190-16) + (43-16) = 341

Advantages:

- **Uniform Wait Time:** Requests are serviced in a circular manner, so waiting times are more predictable and fair.
- **Reduced Head Movement**: The arm only goes as far as the last request in one direction, then jumps back, saving time compared to C-SCAN.

Disadvantages of C-LOOK Algorithm:

- Extra head movement due to circular scanning
- Increased seek time compared to LOOK under light load
- Newly arrived requests may have to wait for a full cycle

Learn more in detail: 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.

- Maximizes locality and resource utilization
- Can seem a little unfair to other requests and if new requests keep coming in, it cause starvation to the old and existing ones.