

**DEPARTMENT OF INFORMATION TECHNOLOGY & COMPUTER
ENGINEERING**



LECTURE NOTES

SUBJECT: OPERATING SYSTEM

BRANCH: 4TH SEM, IT

FACULTY NAME: DR. ANITARANI BRAHMA

Jharsuguda Engineering School, Jharsuguda

Department of Information Technology

Vision:

To focus on development of skilled and confident personalities of today and tomorrow by using cutting edge technology in the department of Information Technology to accept need based challenges with a sense of social responsibilities.

Mission:

- **M1:** To impart quality education by implementing state-of- the- art teaching-learning methods to enrich the academic competency, credibility and integrity of the students.
- **M2:** To implement the educational program in our department from fundamental engineering to recent technology as per emerging trends in the field of Information Technology.
- **M3:** To facilitate a conducive ambience and infrastructure to develop professional skills and nurture innovation in students.
- **M4:** To inculcate sensitivity towards society, respect for environment and promote high standards of ethics.

LEARNING OUTCOMES:

After completing this course, student will be able to:

- Explain the basic concept about the OS,
- Illustrate UNIX/LINUX architecture,
- Explain Process management, memory management and file management,
- Demonstrate I/O system including RAID,
- Explain the concept of deadlocks and measures to prevent them,
- Illustrate OS Security.

Course Outcome

CO1:

Explain the fundamental concepts, structure, and architecture of Operating Systems including UNIX/Linux.

(Aligned with Mission: M1, M2)

CO2:

Analyze process management techniques including scheduling algorithms, inter-process communication, and multithreading.

(Aligned with Mission: M1, M2, M3)

CO3:

Apply memory management techniques such as paging, segmentation, swapping, and virtual memory for efficient resource utilization.

(Aligned with Mission: M1, M2, M3)

CO4:

Describe and evaluate file management systems, disk scheduling algorithms, and I/O system operations.

(Aligned with Mission: M1, M2, M3)

CO5:

Analyze deadlock handling strategies and apply operating system security principles with ethical and social responsibility.

(Aligned with Mission: M1, M3, M4)

CHAPTER-1

Overview of Operating System

Basic Concepts of Operating System

An Operating System (OS) is system software that acts as an interface between the user and computer hardware. It manages resources efficiently and provides common services for computer programs.

Objectives of Operating System:

- Convenience – makes the computer easy to use
- Efficiency – utilizes hardware resources effectively
- Ability to evolve – allows introduction of new functions

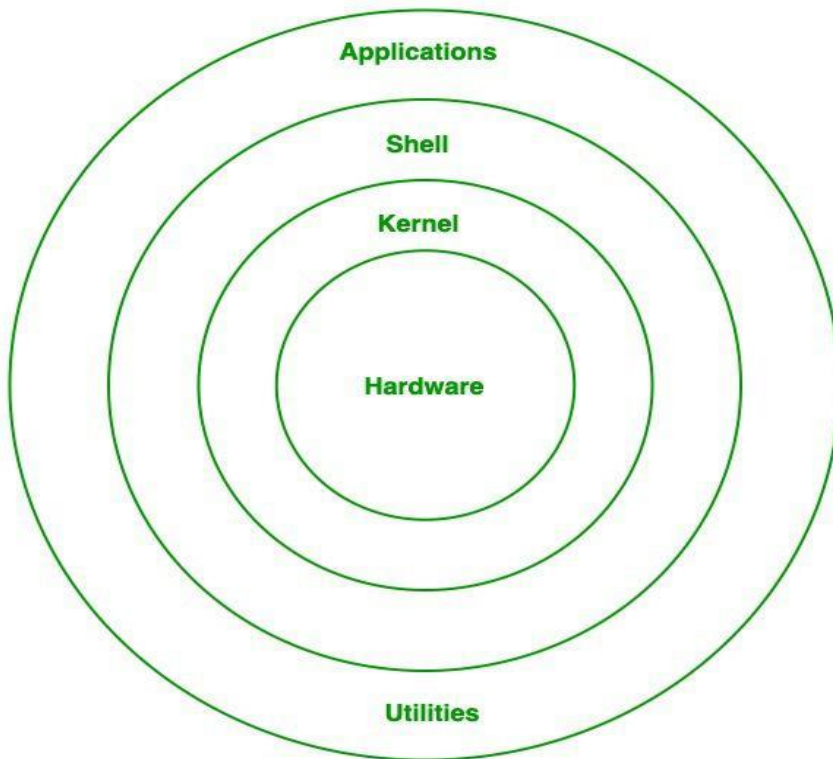
Functions of Operating System:

- Process Management
- Memory Management
- File Management
- Device Management
- Security and Protection
- User Interface (CLI / GUI)

UNIX / LINUX Architecture:

UNIX/Linux operating systems follow a layered architecture that ensures modularity and reliability.

The main components of Linux operating system are: Application, Shell, Kernel, Hardware, Utilities



Architecture Layers:

- **Hardware**: Physical components like CPU, memory, disk
- **Kernel**: Core of OS managing resources
- **Shell**: Command interpreter between user and kernel
- **Utilities & Applications**: User-level programs

1. Kernel & Its Services:

The Kernel is the heart of an operating system.

It remains in memory and controls system operations.

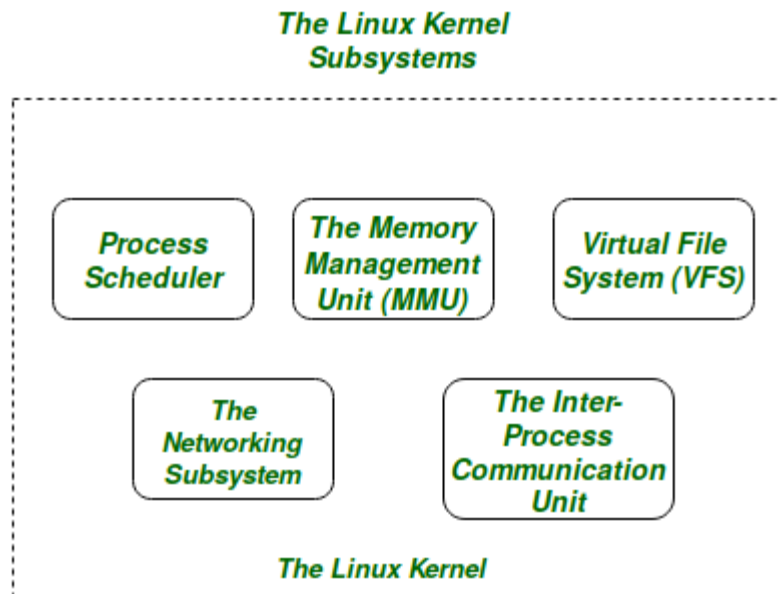
Kernel is the main core component it lies **between the shell and the hardware**.

It controls the activity of other hardware components.

The kernel is software that manages communication between the hardware and the system.

Kernel Responsibilities:

- Memory management: Manages and allocates memory efficiently.
- Resource allocation: Distributes system resources to different processes.
- Device management: Controls input/output devices like printers and scanners.
- Process management: Manages process execution and scheduling.
- Application interaction: Bridges applications with system-level functions.
- Security: Provides essential system-level security.



2. Shell

The Shell is also software or It can be determined as the interface to the kernel.

It takes **commands** from the user and **interprets** them.

The shell **transmits** these commands to the **kernel**, which then performs the requested operations.

Users can just enter the command and using the kernel's function that specific task is performed accordingly.

3. Hardware Layer

Hardware layer of Linux is the lowest level of operating system track. It plays a vital role in managing all the hardware components. It includes device drivers, kernel functions, memory management, CPU control, and I/O operations.

4. System Utilities

System utilities are the command line tools that perform various tasks provided by user to make system management and administration better. These utilities enable user to perform different tasks, such as file management, system monitoring, network configuration, user management etc.

System Calls

System calls provide an interface between user programs and kernel services.

Examples of system calls:

- Process control: fork(), exit()
- File management: open(), read(), write(), close()
- Device management
- Communication
- Information maintenance

System Programs

System programs provide a convenient environment for the development and execution of programs. They make system calls to request services from the kernel.

Categories of System Programs:

- File management (ls, cp, rm)
- Status information (ps, top, df)
- Programming support (gcc, javac)
- Communication (mail, ftp)
- Command interpreters (bash)

Review Questions

1. Define Operating System and explain its functions.
2. Draw and explain UNIX/Linux architecture.
3. What is Kernel? Explain its services.

4. What are system calls? Give examples.
5. Explain different types of system programs.

Short Answer Questions (2/5 Marks)

- What is an Operating System?
- Define Kernel.
- What is a system call?
- Name any four system programs.

CHAPTER 4

(FILE MANAGEMENT)

File Concept

A **file** is the basic unit of storage in an operating system.

A file is a collection of related information.

The file system resides on secondary storage and provides efficient and convenient access to the disk.

File system implementation in an operating system refers to how the file system manages the storage and retrieval of data on a physical storage device such as a hard drive, solid-state drive, or flash drive.

Examples of Files

- Text files (.txt)
- Document files (.docx)
- Image files (.jpg, .png)
- Program files (.exe)
- Audio/Video files (.mp3, .mp4)

File Attributes

File attributes provide information about the file.

Common file attributes include:

- **File Name** – Name of the file
- **File Type** – Type of file (text, image, etc.)
- **File Size** – Size of file in bytes
- **Location** – Path where file is stored
- **Creation Date** – Date file was created
- **Modification Date** – Last modified date
- **Protection** – Access permissions

3. File Operations

Operating systems perform several operations on files.

Common File Operations

1. **Create**
 - Creates a new file.
2. **Open**
 - Opens an existing file.
3. **Read**
 - Reads data from file.
4. **Write**
 - Writes data into file.
5. **Append**
 - Adds data at the end of file.
6. **Close**
 - Closes the file after use.
7. **Delete**
 - Removes the file.
8. **Rename**
 - Changes the file name.

File Access Methods:

File access methods are techniques used by an OS to read and write data in files.

They define how information is organized, retrieved, and modified.

There are three ways to access a file in a computer system:

- Sequential-Access
- Direct Access
- Index sequential Method

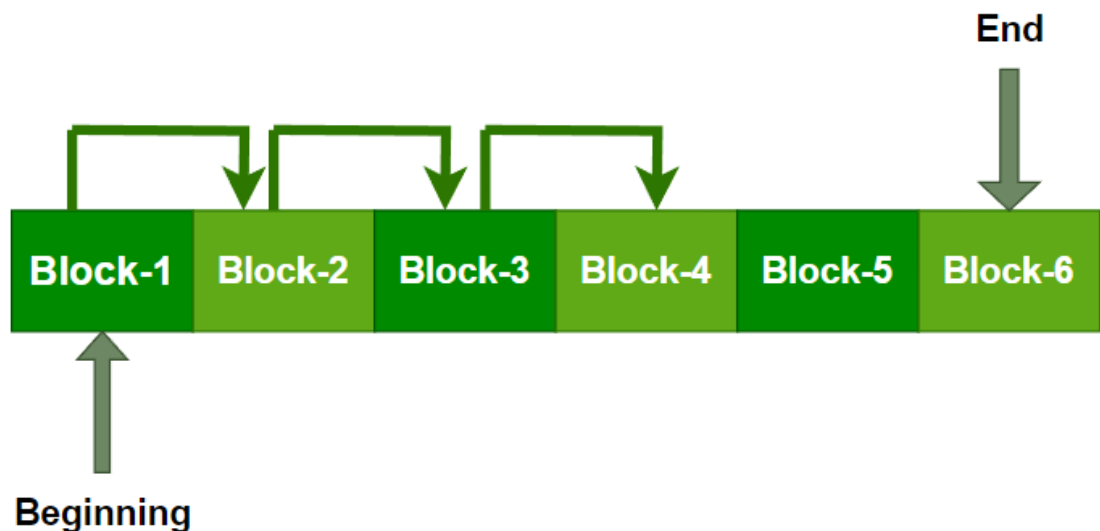
a) Sequential Access

A file access method where data is read or written **in order, one record after another**, starting from the beginning.

The file pointer moves forward automatically after each operation.



Sequential Access Method



Sequential Access Method

Key Points related to Sequential Access

- Data is accessed from one record right after another record in an order.
- When we use the read command, it moves ahead pointer by one.
- When we use the write command, it will allocate memory and move the pointer to the end of the file.
- Such a method is reasonable for tape.

Advantages of Sequential Access Method

- It is simple to implement this file access mechanism.
- It is less prone to data corruption as the data is written sequentially and not randomly.

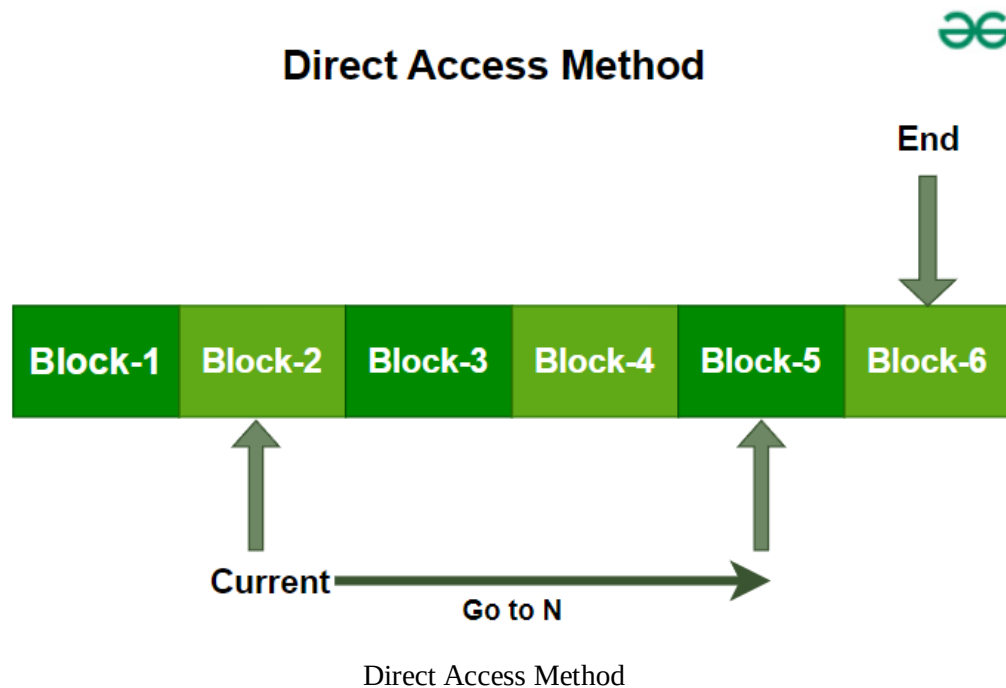
Disadvantages of Sequential Access Method

- Slow for searching specific records.
- Inserting/updating in the middle is difficult.
- Can waste storage if records have varying lengths.
-

b) Direct Access Method

A file access method that allows data to be **read or written directly at any block or record**, using its address (block number).

It supports **random access** without scanning previous records.



Advantages

- The files can be immediately accessed decreasing the average access time.
- In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

Disadvantages

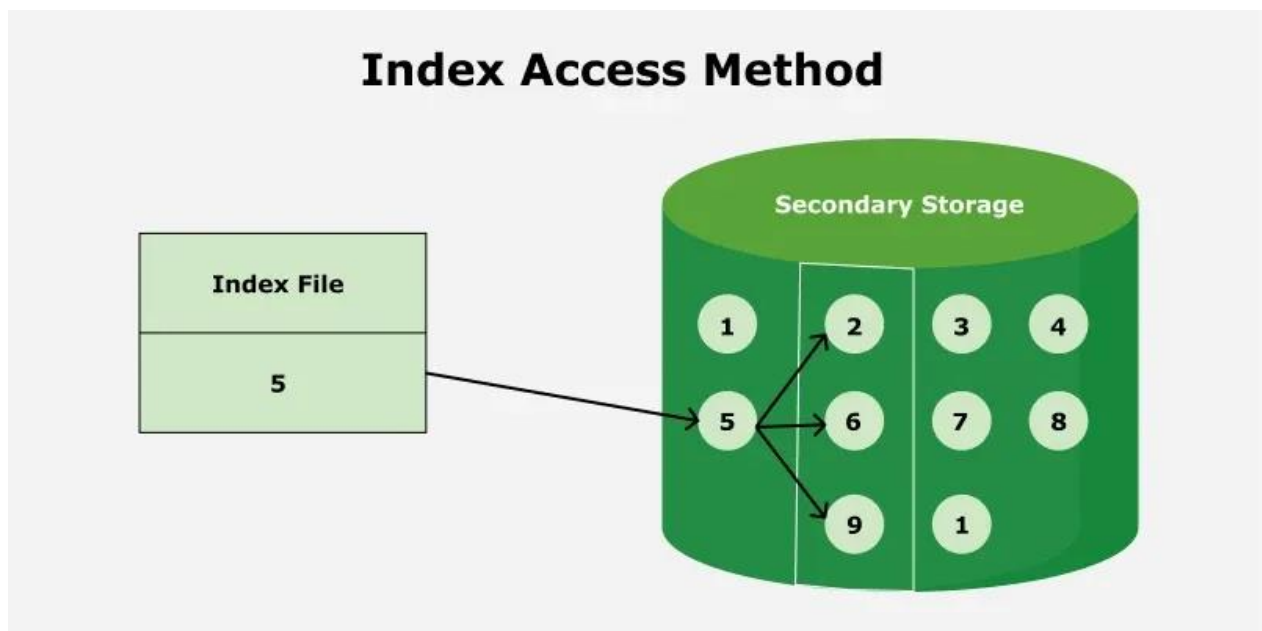
- **Complex Implementation** : Implementing direct access can be complex, requiring sophisticated algorithms and data structures to manage and locate records efficiently.
- **Higher Storage Overhead** : Direct access methods often require additional storage for maintaining data location information (such as pointers or address tables), which can increase the overall storage requirements.

c) Index Sequential method

It is the other method of accessing a file that is built on the top of the sequential access method.

These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks.

To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.



Index Access Method

- It is built on top of Sequential access.
- It control the pointer by using index.

Advantages

- **Fast Searching:** Index enables quick lookups.
- **Flexible:** Supports both sequential and random access.
- **Reduced Access Time:** Quickly locates data in large files.

Disadvantages

- **Complexity:** Harder to implement and maintain than sequential access.
- **Extra Storage:** Requires additional space for indexes.

- **Slower Updates:** Both data and index must be updated during insertions, deletions, or modifications.
- **Maintenance Overhead:** Indexes need frequent updates in dynamic environments.

Directory Structure

- The operating system uses directories to track where files are stored.
- Different directory structures can be used to suit various organisational needs.
- Directory structures help in that help organise and manage files efficiently.

Types of Directory Structures:

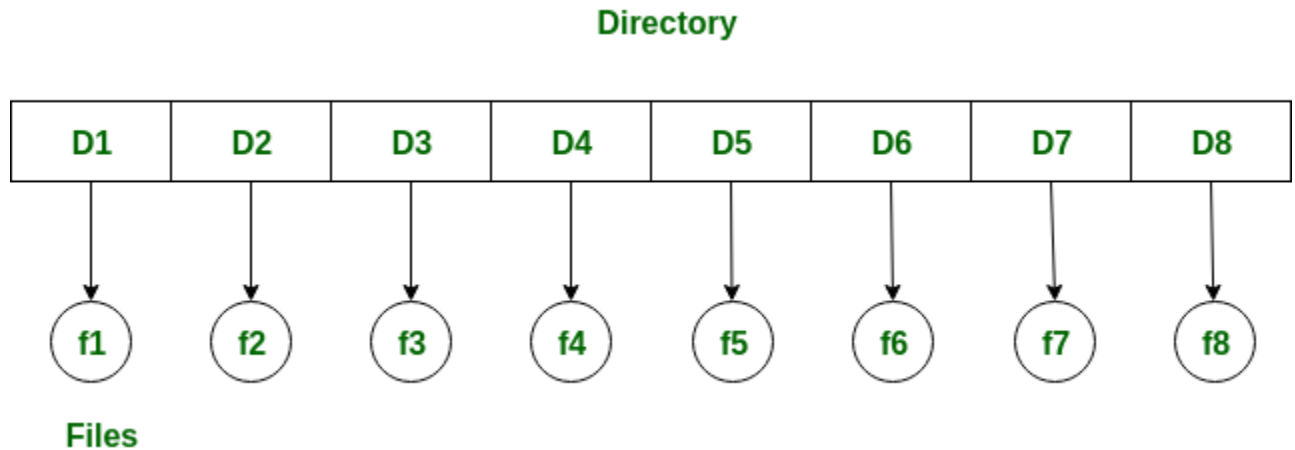
- **Single-Level Directory:** All files stored in one directory.
- **Two-Level Directory:** Each user has a separate directory.
- **Tree Structure Directory:** Directories contain subdirectories (most commonly used).
- **Graph Directory Structure:** Allows shared files/directories.

Single-Level Directory

The single-level directory is the **simplest directory structure**.

In it, all files are contained in the same directory which makes it easy to support and understand.

Since all the files are in the same directory, they must have a **unique name**.



Advantages

- **Logical Organization:** Arrange files hierarchically for easy navigation.
- **Efficiency:** Faster file searching and access.
- **Security:** Restrict access at directory level to protect data.
- **Backup & Recovery:** Simplifies locating and restoring files.
- **Scalability:** Easily supports growth with new files and directories

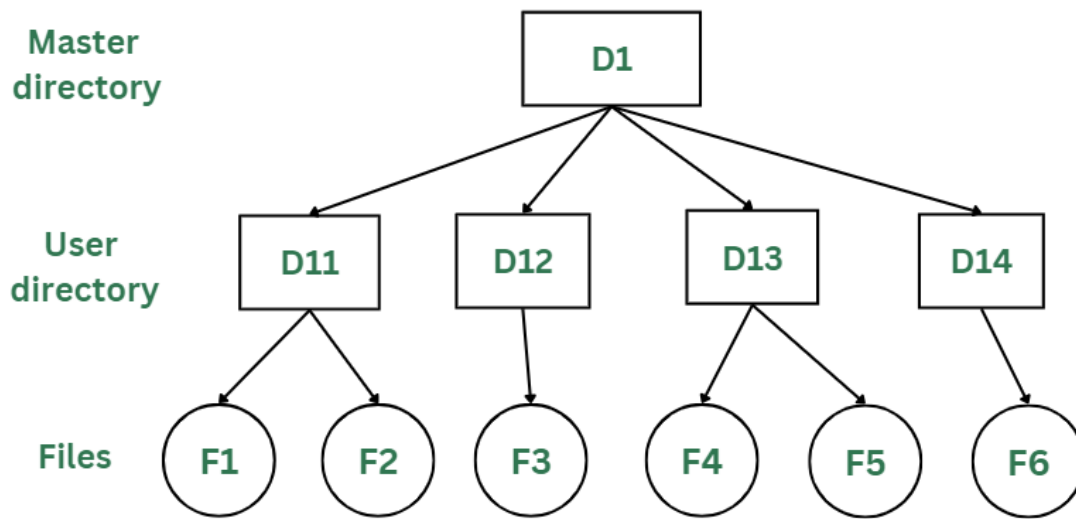
Disadvantages

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

2) Two-Level Directory

In a two-level directory structure, each user has a separate **User File Directory (UFD)** containing only their files.

A **Master File Directory (MFD)** stores entries for all users and points to their respective UFDs, preventing filename conflicts between users.



Two-Levels Directory Structure

Advantages

- The main advantage is there can be more than two files with same name, and would be very helpful if there are multiple users.
- A security would be there which would prevent user to access other user's files.
- Searching of the files becomes very easy in this directory structure.

Disadvantages

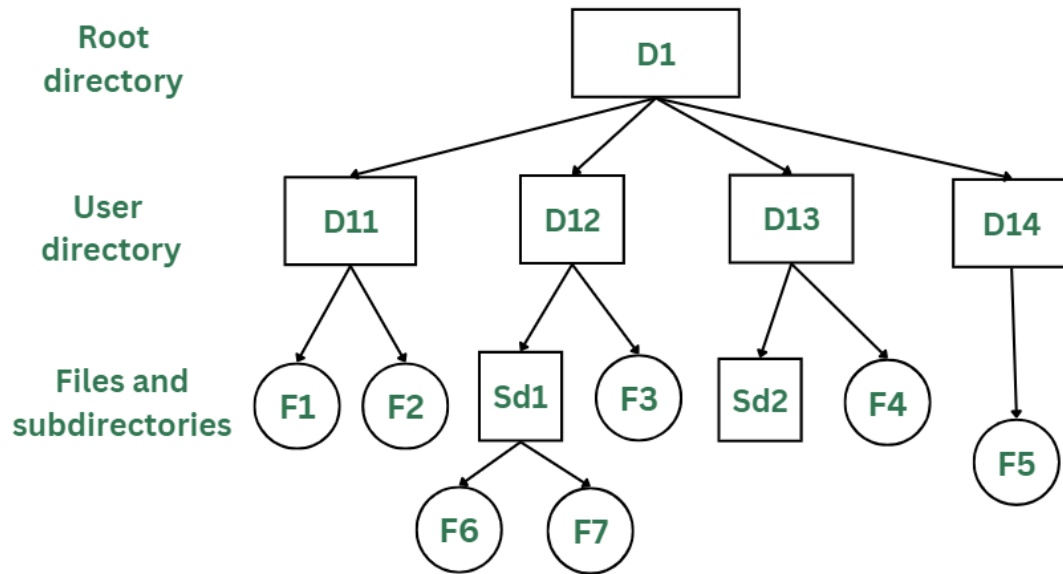
- User cannot share the file with the other users.
- Unlike the advantage users can create their own files, users don't have the ability to create subdirectories.
- Scalability is not possible because one user can't group the same types of files together.

3) Tree Structure/ Hierarchical Structure

The tree directory structure is the most common in personal computers.

It resembles an upside-down tree, with the root directory at the top containing all user directories.

Each user can create files and subdirectories within their own directory but cannot access or modify the root or other users' directories.



Tree/Hierarchical Directory Structure

Advantages

- This directory structure allows subdirectories inside a directory.
- The searching is easier.
- It helps in organizing files by importance, making important files easier to locate.
- This directory is more scalable than the other two directory structures explained.

Disadvantages

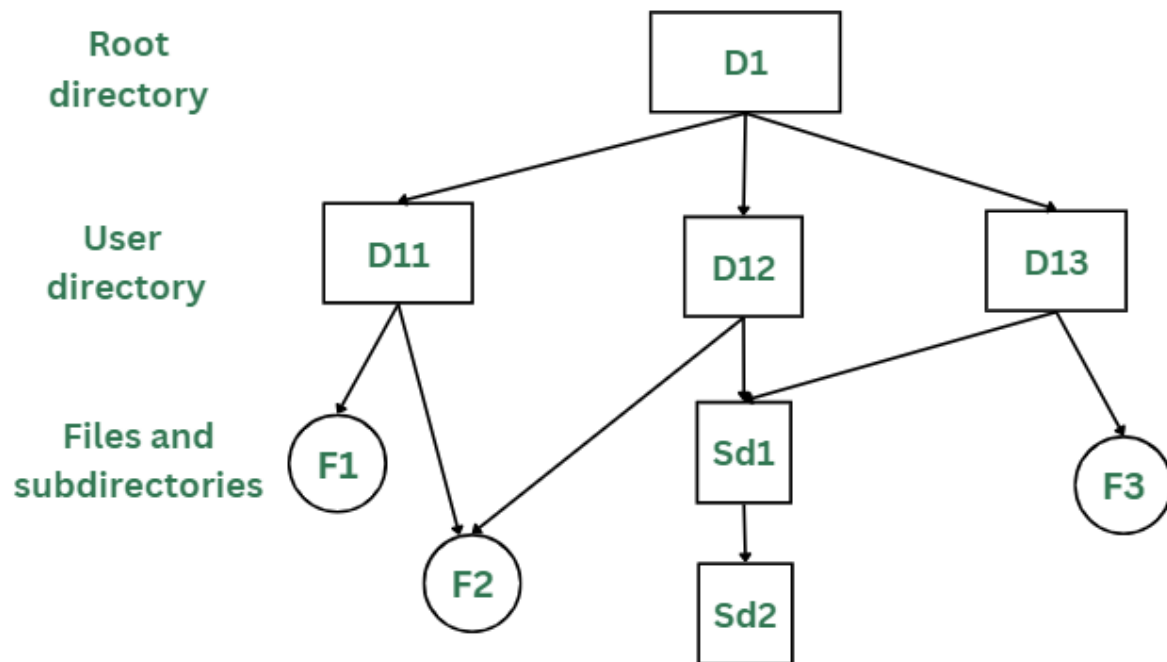
- As the user isn't allowed to access other user's directory, this prevents the file sharing among users.
- As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
- Users cannot modify the root directory data.

4) Acyclic Graph Structure

In the earlier directory structures, a file could only be accessed from the directory it was stored in.

The acyclic graph directory structure solves this by allowing a file or subdirectory to be shared across multiple directories using links.

Changes made by one user are visible to all users sharing that file.



Acyclic Graph Structure

Advantages

- Sharing of files and directories is allowed between multiple users.
- Searching becomes too easy.

- Flexibility is increased as file sharing and editing access is there for multiple users.

Disadvantages

- Because of the complex structure it has, it is difficult to implement this directory structure.
- The user must be very cautious to edit or even deletion of file as the file is accessed by multiple users.
- If we need to delete the file, then we need to delete all the references of the file in order to delete it permanently.

File Sharing & Protection

File Sharing in an Operating System (OS) denotes how information and files are shared between different users, computers, or devices on a network

Access Rights based on operation:

- **Read (R):** View file contents
- **Write (W):** Modify file
- **Execute (X):** Run program
- **Append** - Writing the new information to the already existing file, editing must be end at the end of the existing file.
- **Delete** - Deleting the file which is of no use and using its space for the another data.
- **List** - List the name and attributes of the file.

Protection Methods:

1. Access Control List (ACL):

It specifies the names of the users and the types of access associated with each of the user.

The main problem with the access list is their length.

If we want to allow everyone to read a file, we must list all the users with the read access

Three classification of users in connection with each file:

- **Owner** - Owner is the user who has created the file.
- **Group** - A group is a set of members who has similar needs and they are sharing the same file.
- **Universe** - In the system, all other users are under the category called universe.

2. Password Protection

3. Encryption

File System Implementation

File system implementation deals with how files are stored.

The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

File system implementation in an operating system refers to how the file system manages the storage and retrieval of data on a physical storage device such as a hard drive, solid-state drive, or flash drive.

Main Components:

- File Control Block (FCB): Stores file name, size, location, permissions.
- Inode: Stores owner details, size, and disk block locations (UNIX).

File Allocation Methods:

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

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

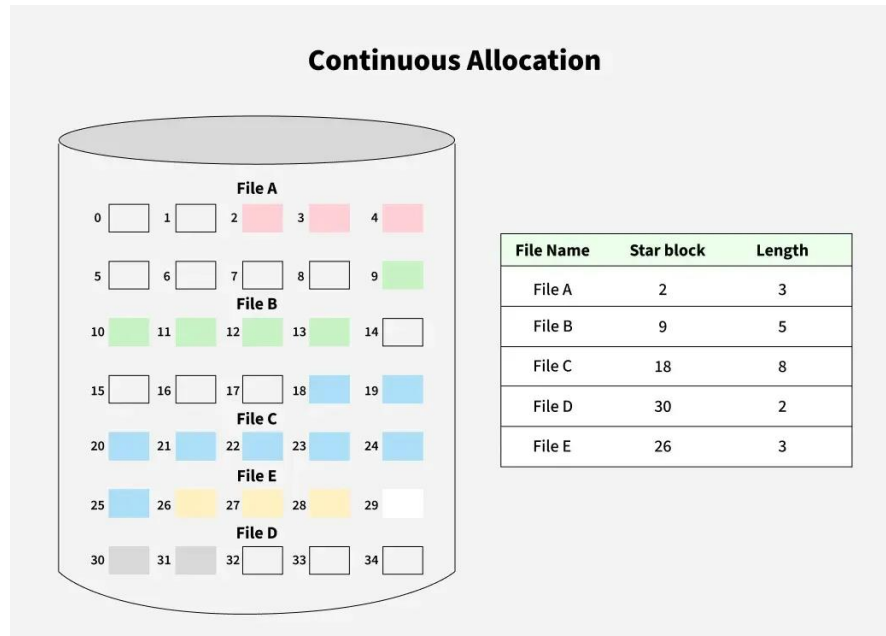
The main idea behind these methods is to provide Efficient disk space utilization & Fast access to the file blocks.

Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains:

- Address of starting block

- Length of the allocated portion.
- **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:
 $b, b+1, b+2, \dots, b+n-1$.



Contiguous Allocation

Advantages

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the k th block of the file which starts at block b can easily be obtained as $(b+k)(b+k)$.
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages

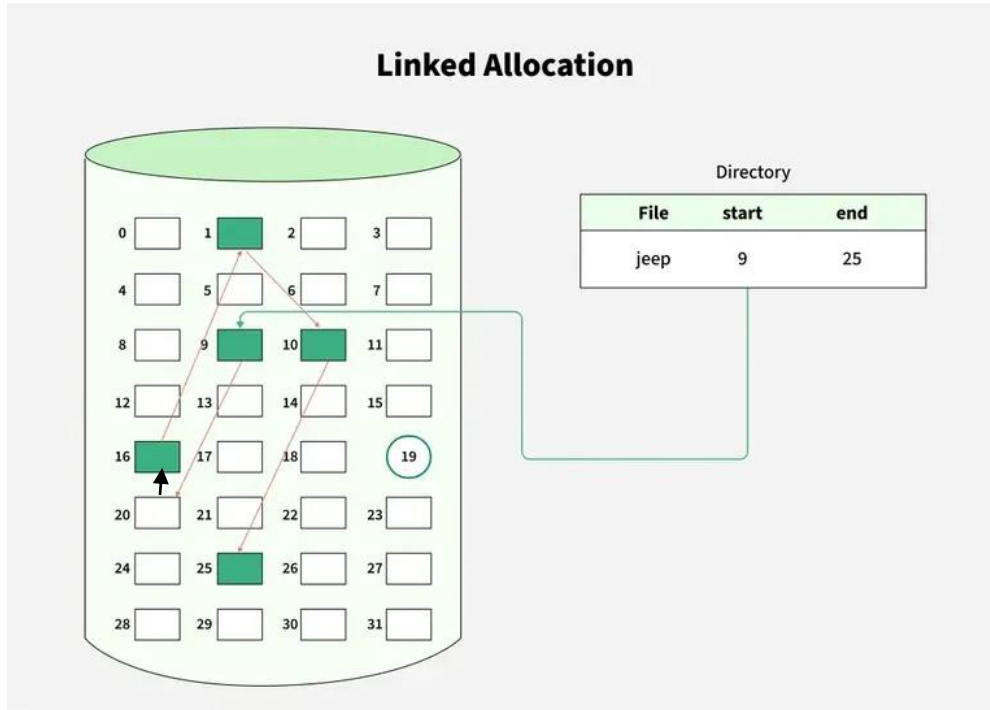
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked Allocation

In this scheme, each file is a linked list of disk blocks which need not be contiguous. Here,

- The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block.

- Each block contains a pointer to the next block occupied by the file.
- **Example:** The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Linked Allocation

Advantages

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages

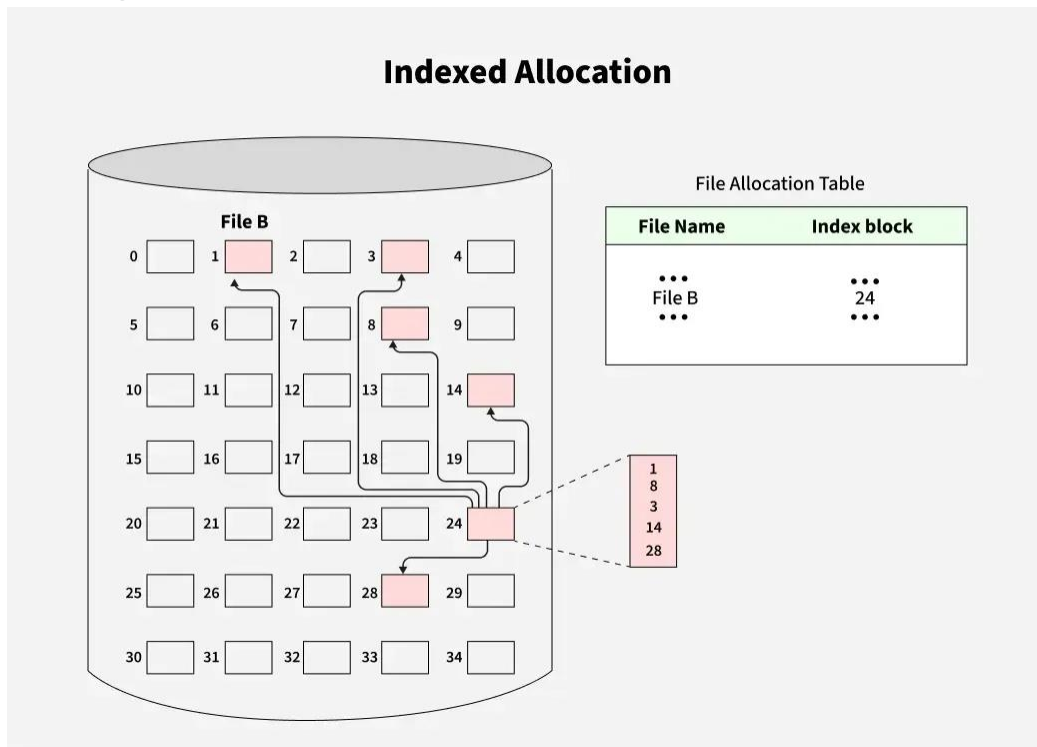
- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.

- Pointers required in the linked allocation incur some extra overhead.

4. Indexed Allocation

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Here,

- Each file has its own index block.
- The i^{th} entry in the index block contains the disk address of the i^{th} file block.
- The directory entry contains the address of the index block as shown in the image.



Indexed Allocation

Advantages

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

Disadvantages

- The pointer overhead for indexed allocation is greater than linked allocation.

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

Free Space Management

Free space management tracks unused disk space.

Methods:

- Bit Map (Bit Vector)
- Linked List Method
- Grouping Method
- Counting Method

5. File System Types

Common File System Types:

- FAT (File Allocation Table): Used in USB drives and older systems.
- NTFS (New Technology File System): Used in Windows.
- EXT (Extended File System): Used in Linux (EXT2, EXT3, EXT4).
- CDFS: Used for CD-ROMs.
- NFS (Network File System): Used for network file sharing.

Types:

- **FAT32 (File Allocation Table 32):** Commonly used in older versions of Windows and compatible with various operating systems.

- **NTFS (New Technology File System):** Used in modern Windows operating systems, offering improved performance, reliability, and security features.
- **ext4 (Fourth Extended File System):** Used in Linux distributions, providing features such as journaling, large file support, and extended file attributes.
- **HFS+ (Hierarchical File System Plus):** Used in macOS systems prior to macOS High Sierra, offering support for journaling and case-insensitive file names.
- **APFS (Apple File System):** Introduced in macOS High Sierra and the default file system for macOS and iOS devices, featuring enhanced performance, security, and snapshot capabilities.
- **ZFS (Zettabyte File System):** A high-performance file system known for its advanced features, including data integrity, volume management, and efficient snapshots.

CHAPTER 5

(I/O SYSTEMS)

MASS STORAGE:

Mass storage provides permanent data storage, typically using **Magnetic Disks** (Hard Disk Drives - HDDs) for fast direct access, or **Magnetic Tapes** for long-term, sequential access.

- **Logical Blocks:** The OS addresses disks as large 1D arrays of logical blocks, which are mapped to the physical sectors of the disk.
- **Performance Metrics:**
 - **Seek Time:** Time for the disk arm to move the heads to the cylinder containing the desired sector (main factor in performance).
 - **Rotational Latency:** Time for the desired sector to rotate under the disk head.

Disk Structure:

A disk structure in an operating system organizes physical storage (HDDs) using platters, tracks, sectors, and read/write heads to store data magnetically. Data is arranged on concentric tracks divided into sectors, with read/write heads on a disk arm accessing data

Key Components of Disk Structure

- **Platters:** Circular magnetic discs that store data.
- **Tracks:** Concentric circles on the surface of a platter.
- **Sectors:** Subdivisions of tracks; the smallest physical storage unit.
- **Cylinders:** A set of tracks at the same radius across multiple platters.

- **Read/Write Head:** Component that reads/writes data on the magnetic surface.
- **Disk Arm:** Moves the read/write heads to the desired track.

Types of Disk Structures

- **Magnetic Disks (HDD):** Use rotating magnetic platters for storage.
- **Optical Disks:** Use laser technology, such as CDs or DVDs.
- **Solid State Drives (SSD):** Flash-based memory without moving parts.

Disk Scheduling Algorithm:

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.

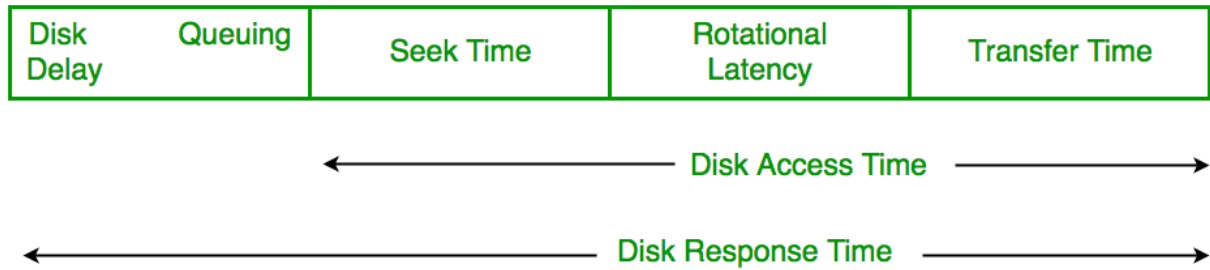
Key Terms Associated with Disk Scheduling

- **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*



Goals of Disk Scheduling Algorithms

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

Disk Scheduling Algorithms

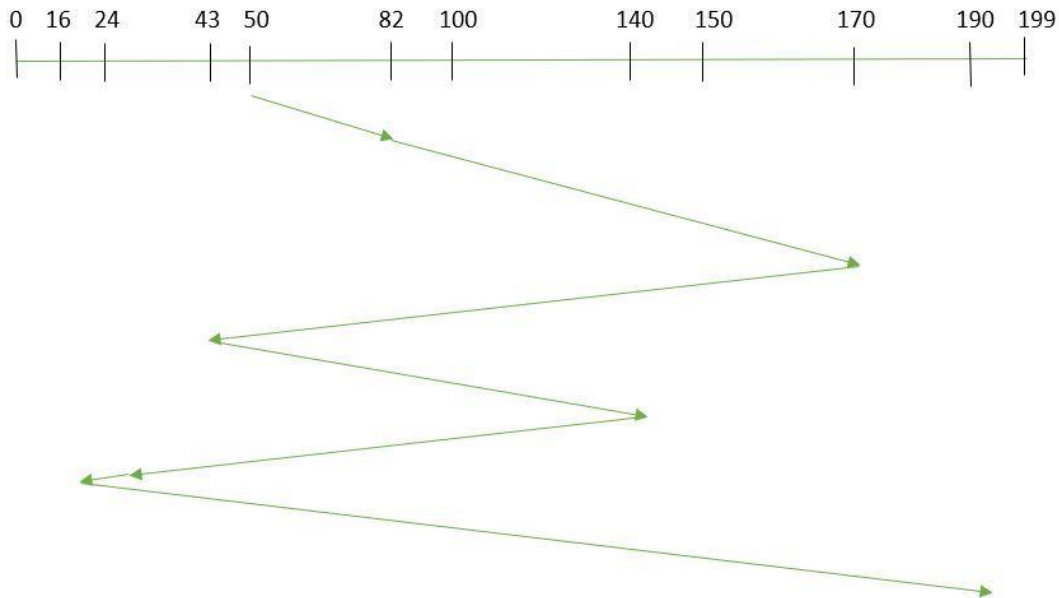
There are several Disk Scheduling Algorithms. We will discuss in detail each one of them.

- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN
- C-SCAN
- LOOK
- C-LOOK

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.



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

- Every request gets a fair chance
- No indefinite postponement

Disadvantages of FCFS

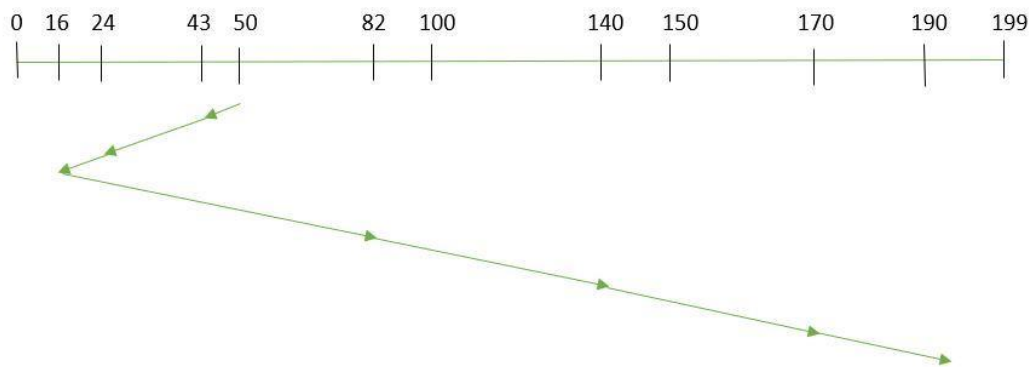
- Does not try to optimize seek time
- May not provide the best possible service

2. SSTF (Shortest Seek Time First)

The request near the disk arm will get executed first.

SSTF is an improvement over FCFS as it decreases the average response time and increases the throughput of the system.

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

- The average Response Time decreases
- Throughput increases

Disadvantages of Shortest Seek Time First

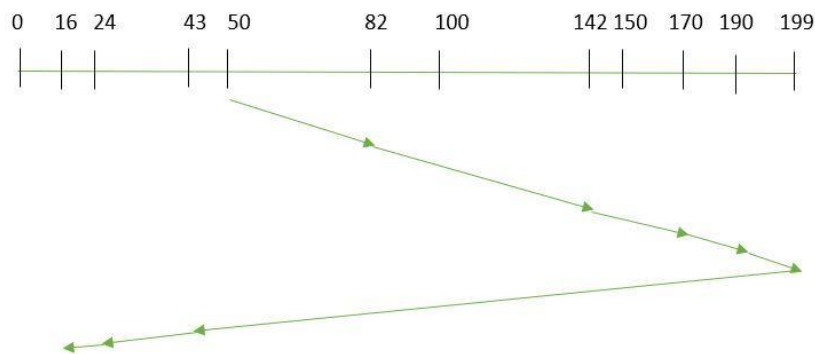
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

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

- 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

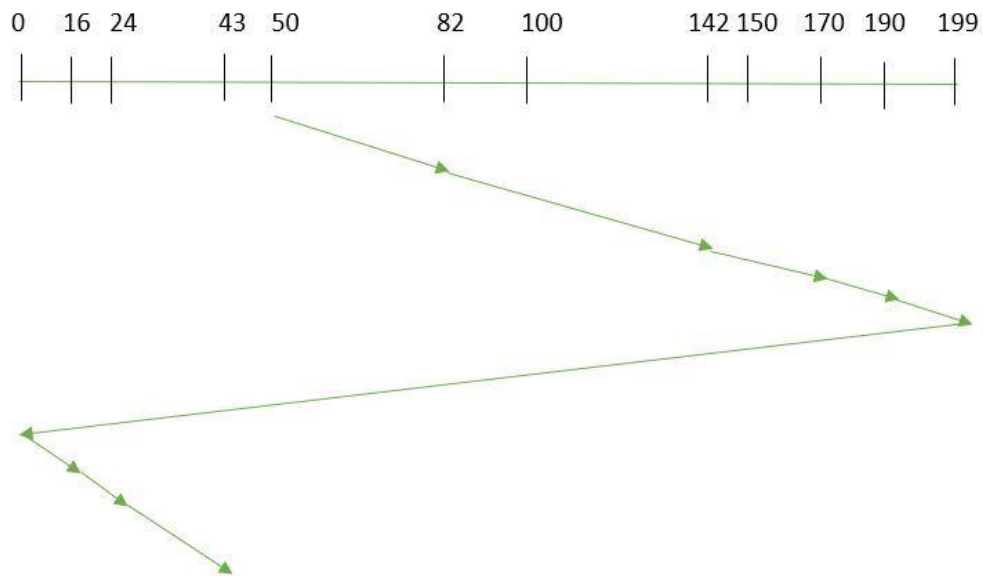
4. C-SCAN

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

Disadvantages of C-SCAN Algorithm:

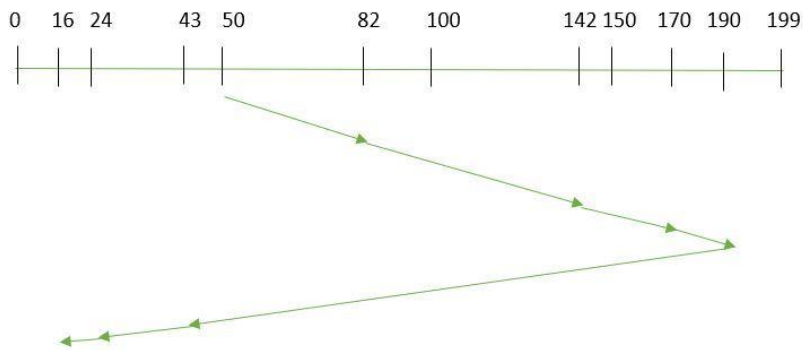
- Extra head movement due to return to start
- Increased seek time compared to SCAN in light load
- More overhead because of circular movement

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

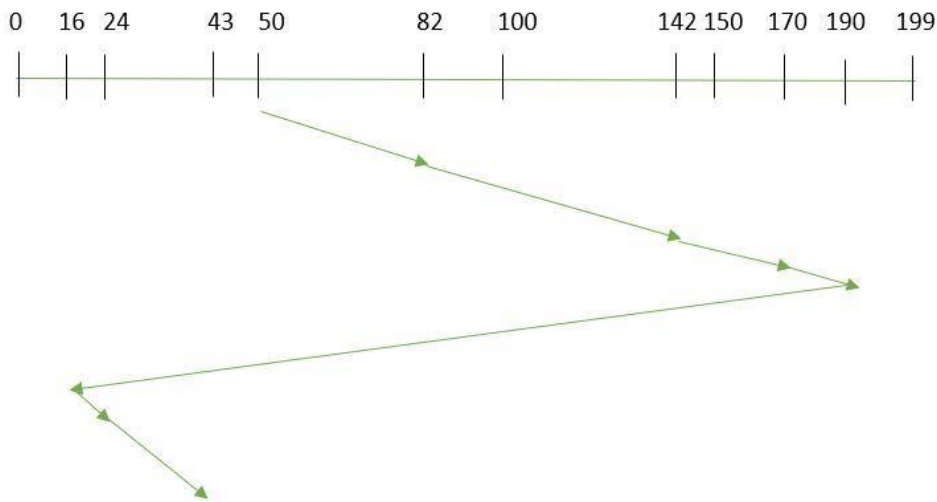
6. C-LOOK

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

RAID (Redundant Arrays of Independent Disks)

RAID is a technique that combines multiple hard drives or SSDs into a single system to improve performance, data safety or both.

If one drive fails, data can still be recovered from the others.

Note: Different RAID levels offer different combinations of speed, storage capacity and fault tolerance.

How RAID Works?

In RAID (Redundant Array of Independent Disks), data is not stored on just one hard drive but is distributed across multiple drives.

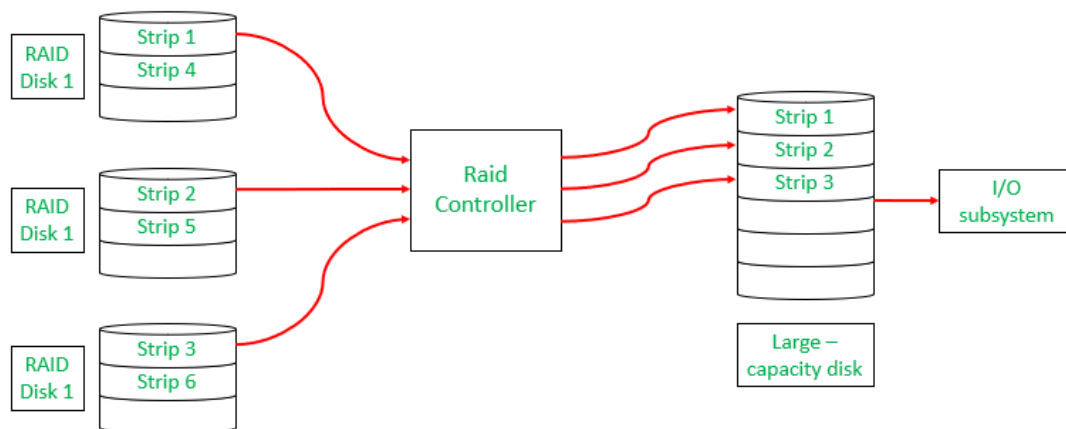
The data is split into small blocks (like dividing a file into chunks). These blocks are written across multiple drives in parallel.

- **Mirroring (RAID 1):** Exact copy of data is kept on another drive.
- **Parity (RAID 5, RAID 6):** A calculated value (parity block) is stored to allow data recovery in case of failure.
- **Fault Tolerance:** If one drive fails, RAID uses the redundant data (mirror or parity) to reconstruct missing data.

Note: This improves performance by allowing multiple drives to read/write data simultaneously.

RAID Controller

A RAID controller manages multiple hard drives, making them work together as one system. It helps improve speed and adds data protection by handling drive failures. Think of it as a smart manager that boosts performance and keeps your data safe.



RAID Controller

Why Data Redundancy?

Data redundancy, although taking up extra space, adds to disk reliability. This means,

- That in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation.
- On the other hand, if the data is spread across multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

It offers:

1. Reliability

Refers to the system's ability to tolerate disk faults and prevent data loss.

Example:

- RAID 0 offers no fault tolerance; if one disk fails all data is lost.
- RAID 5 can tolerate one disk failure due to parity data.
- RAID 6 can handle two simultaneous disk failures.

2. Availability

The fraction of time the RAID system is operational and available for use.

Example:

- RAID 1 (Mirroring) allows immediate data access even during a single disk failure.
- RAID 5 and 6 may degrade performance during a rebuild, but data remains available.

3. Performance

Measures how efficiently the RAID system handles data processing tasks. This includes:

- **Response Time:** How quickly the system responds to data requests.

- **Throughput:** The rate at which the system processes data (e.g., MB/s or IOPS).

Key Factors:

RAID levels affect performance differently:

- RAID 0 offers high throughput but no redundancy.
- RAID 1 improves read performance by serving data from either mirrored disk but may not improve write performance significantly.
- RAID 5/6 introduces overhead for parity calculations, affecting write speeds.
- Workload type (e.g., sequential vs. random read/write operations).

Performance Trade-offs: *Higher redundancy often comes at the cost of slower writes (due to parity calculations).*

4. Capacity

The amount of usable storage available to the user after accounting for redundancy mechanisms. For a set of N disks, each with B blocks, the available capacity depends on the RAID level:

- **RAID 0:** All $N \times B$ blocks are usable (no redundancy).
- **RAID 1:** Usable capacity is B (only one disk's capacity due to mirroring).
- **RAID 5:** Usable capacity is $(N-1) \times B$ (one disk's worth of capacity used for parity).
- **RAID 6:** Usable capacity is $(N-2) \times B$ (two disks' worth used for parity).

Different RAID Levels

- RAID-0 (Stripping)
- RAID-1 (Mirroring)
- RAID-2 (Bit-Level Stripping with Dedicated Parity)
- RAID-3 (Byte-Level Stripping with Dedicated Parity)
- RAID-4 (Block-Level Stripping with Dedicated Parity)
- RAID-5 (Block-Level Stripping with Distributed Parity)
- RAID-6 (Block-Level Stripping with two Parity Bits)

1. RAID-0 (Striping)

RAID-0 improves system performance by splitting data into smaller "blocks" and spreading them across multiple disks. This process is called "striping."

It enhances data access speed by enabling parallel read/write operations but provides no redundancy or fault tolerance.

RAID 0

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

RAID-0

- A set of blocks distributed across the disks forms a "stripe." For instance, "0, 1, 2, 3" is one stripe.
- Instead of placing just one block into a disk at a time, we can work with two (or more) blocks placed into a disk before moving on to the next one.

Evaluation:

- **Reliability:** 0, There is no duplication of data. Hence, a block once lost cannot be recovered.
- **Capacity:** $N \times B$ $N \times B$, The entire space is being used to store data. Since there is no duplication, N disks each having B blocks are fully utilized.

Benefits:

- It is easy to implement.
- It utilizes the storage capacity in a better way.

Limitations:

- A single drive loss can result in the complete failure of the system.
- It's not a good choice for a critical system.

2. RAID-1 (Mirroring)

RAID-1 enhances reliability by creating an identical copy (mirror) of each data block on separate disks.

This ensures that even if one disk fails, the data remains accessible from its duplicate. While this configuration is highly reliable, it requires significant storage overhead.

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

Raid-1

Mirroring: Each block of data is written to two (or more) disks.

For example:

- Block 0 is stored on Disk 0 and its duplicate on Disk 1.
- Block 1 is stored on Disk 2 and its duplicate on Disk 3.

Mirroring Level 2: In, "mirroring level 2" indicates that there are two copies of each block, distributed across different disks. RAID 0 was unable to tolerate any disk failure. But RAID 1 is capable of reliability.

Evaluation: Assume a RAID system with mirroring level 2.

- **Reliability:** 1 to $N/2$, If we are lucky enough and disks 0 and 2 fail, then this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, $N/2$ disk failures can be handled.
- **Capacity:** $(N*B)/2$ $(N*B)/2$, Only half the space is being used to store data. The other half is just a mirror of the already stored data.

Benefits:

- It covers complete redundancy.
- It can increase data security and speed.

Limitations:

- It is highly expensive.
- Storage capacity is less.

3. RAID-2 (Bit-Level Striping with Dedicated Parity)

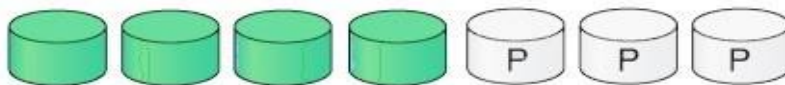
RAID-2 is a specialized RAID level that uses **bit-level striping** combined with error correction using [Hamming Code](#).

Bit-level striping means data is split into **individual bits** and distributed across multiple disks.

- Each disk stores **1 bit** of a byte.
- All disks work **together simultaneously** to read/write a single byte.

In this configuration, data is distributed at the bit level across multiple drives and a dedicated parity drive is used for error detection and correction.

While it offers strong fault tolerance, its complexity and cost make it rarely used in practice.



RAID 2: memory-style error-correcting codes.

RAID 2

Benefits:

- In case of Error Correction, it uses hamming code.
- It Uses one designated drive to store parity.

Limitations:

- It has a complex structure and high cost due to extra drive.
- It requires an extra drive for error detection.

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

RAID-3 enhances fault tolerance by employing **byte-level striping** across multiple drives and storing parity information on a dedicated parity drive.

Byte-level striping means data is split into **bytes** and stored across multiple disks.

- Each disk stores **one full byte**.
- Multiple disks are used together to process data.

The dedicated parity drive allows for the reconstruction of lost data if a single drive fails.

This configuration is suitable for workloads requiring high throughput for sequential data but is less efficient for random I/O operations.

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)

Raid-3

- Here Disk 3 contains the Parity bits for Disk 0, Disk 1 and Disk 2. If data loss occurs, we can construct it with Disk 3.

Evaluation:

- **Reliability:** RAID-3 can tolerate the failure of one disk. The lost data can be reconstructed using the parity drive and the remaining data drives.
- **Capacity:** Usable Capacity = $(N-1) \times B$ where N is the total number of drives and B is the number of blocks per drive. The capacity of one drive is reserved for storing parity information.

Benefits:

- Data can be transferred in bulk.
- Data can be accessed in parallel.

Limitations:

- It requires an additional drive for parity.
- In the case of small-size files, it performs slowly.

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

RAID-5 builds on RAID-4 by distributing parity information across all disks instead of storing it on a dedicated parity drive.

This distributed parity significantly improves write performance, especially for random write operations, while maintaining fault tolerance for single disk failures.

RAID-5 is one of the most commonly used RAID configurations due to its balance between reliability, performance and storage efficiency.

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

Raid-5

Block-Level Striping: Data is divided into blocks and striped across all drives in the array.

Distributed Parity: Parity bits, calculated using the XOR function, are distributed across all drives in a rotating pattern.

Example:

- Disk 0: Data (D1, D2), Parity (P1)
- Disk 1: Data (D3, D4), Parity (P2)
- Disk 2: Data (D5, D6), Parity (P3)

Data Recovery: In case of a single disk failure, the missing data can be reconstructed by XOR-ing the remaining data blocks and parity information.

Evaluation:

- **Reliability:** 1, RAID-5 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data. This is identical to RAID-4.
- **Capacity:** $(N-1)*B$ $(N-1)*B$, Overall, space equivalent to one disk is utilized in storing the parity. Hence, $(N-1)$ disks are made available for data storage, each disk having B blocks.

Benefits:

- Data can be reconstructed using [parity bits](#).

- It makes the performance better.

Limitations:

- Its technology is complex and extra space is required.
- If both discs get damaged, data will be lost forever.

CHAPTER 6

(DEADLOCK)

Operating System Notes

Deadlock

Deadlock is a situation in an Operating System where two or more processes wait indefinitely for resources held by each other, and none of them can proceed.

A process in an operating system typically uses resources in the following sequence:

- Request a resource
- Use the resource
- Release the resource

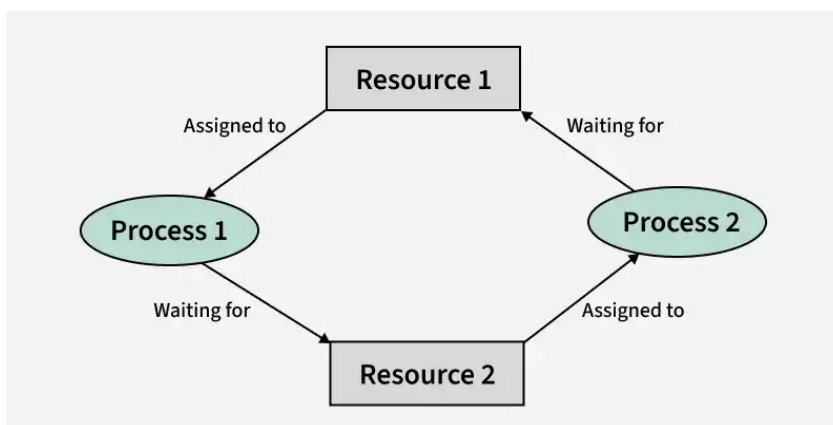
Deadlock arises when processes hold some resources while waiting for others.

Example of Deadlock

Process P1 holds Printer and requests Scanner.

Process P2 holds Scanner and requests Printer.

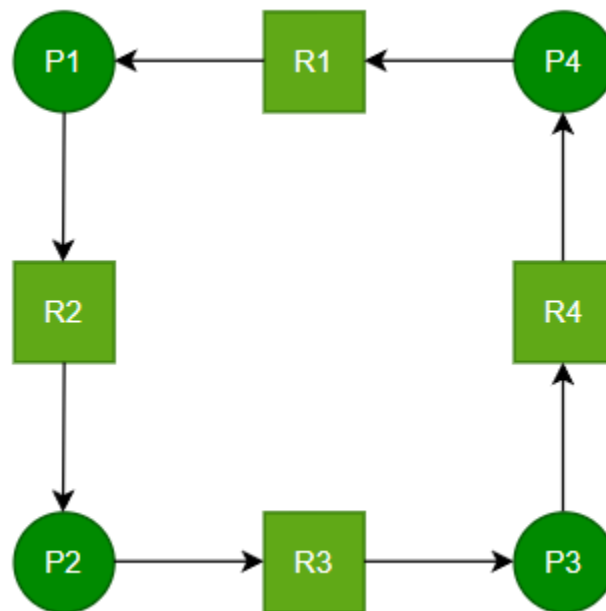
Both wait forever, causing deadlock.



Necessary Conditions for Deadlock in OS

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

1. **Mutual Exclusion:** Only one process can use a resource at any given time i.e. the resources are non-sharable.
2. **Hold and Wait:** A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
3. **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:** set of processes are waiting for each other in a circular fashion. For example, imagine four processes **P1**, **P2**, **P3**, and **P4** and four resources **R1**, **R2**, **R3**, and **R4**.



Deadlock Prevention

Deadlock prevention ensures that at least one of the four necessary conditions does not occur.

Deadlock prevention is a strategy used in computer systems to ensure that different processes can run smoothly without getting stuck waiting for each other forever.

Methods of Deadlock Prevention

1. Eliminate Mutual Exclusion

- Some resources, like a printer, are inherently non-sharable, so this condition is difficult to break.
- However, sharable resources like read-only files can be accessed by multiple processes at the same time.
- For non-sharable resources, prevention through this method is not possible.

2. Eliminate Hold and Wait

There are two ways to eliminate hold and wait:

- **By eliminating wait:** The process specifies the resources it requires in advance so that it does not have to wait for allocation after execution starts. For Example, Process1 declares in advance that it requires both Resource1 and Resource2.
- **By eliminating hold:** The process has to release all resources it is currently holding before making a new request. For Example: Process1 must release Resource2 and Resource3 before requesting Resource1.

3. Eliminate No Preemption

No preemption means resources can't be taken away once allocated. To prevent this:

- **Processes must release resources voluntarily:** A process gives up resources once it finishes using them.
- **Avoid partial allocation:** If a process requests resources that are unavailable, it must release all currently held resources and wait until all required resources are free.

4. Eliminate Circular Wait

Circular wait happens when processes form a cycle, each waiting for a resource held by the next. To prevent this:

- Impose a strict ordering on resources.
- Assign each resource a unique number.
- Processes can only request resources in increasing order of their numbers.
- This prevents cycles, as no process can go backwards in numbering.

Advantages of Deadlock Prevention

- Deadlocks never occur.
- System remains safe.

Disadvantages of Deadlock Prevention

- Low resource utilization.
- Reduced system performance.

Deadlock Detection

Deadlock detection allows deadlock to occur and then detects it.

Deadlock detection is the process of identifying when processes are stuck waiting for resources held by other processes.

Deadlock Detection

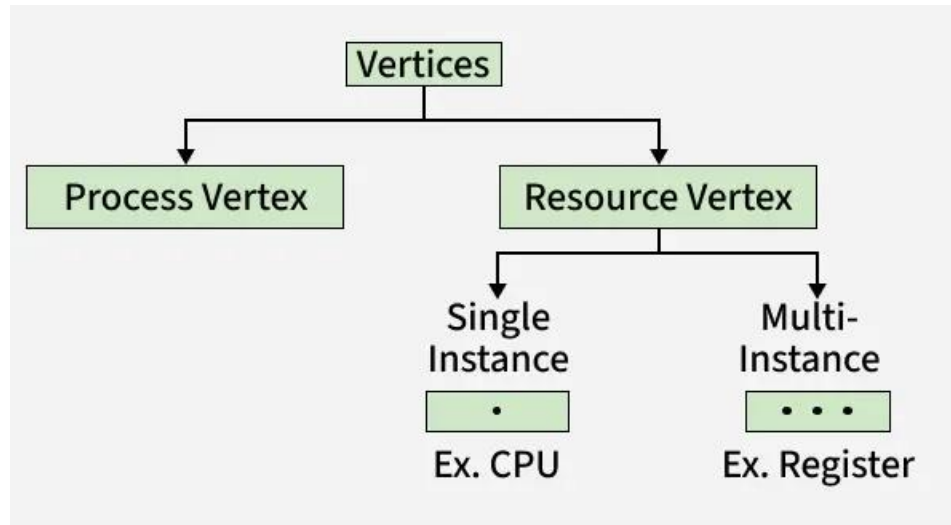
1. If Resources Have a Single Instance

In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a sufficient condition for deadlock.

In the below diagram, resource 1 and resource 2 have single instances. There is a cycle $R1 \rightarrow P1 \rightarrow R2 \rightarrow P2$. So, Deadlock is Confirmed.

Detection Methods

1. Resource Allocation Graph – Used when single instance resources exist.
2. Matrix Method – Used when multiple instances exist using Allocation, Request, and Available matrices.



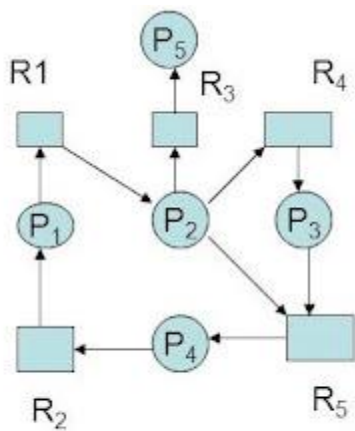
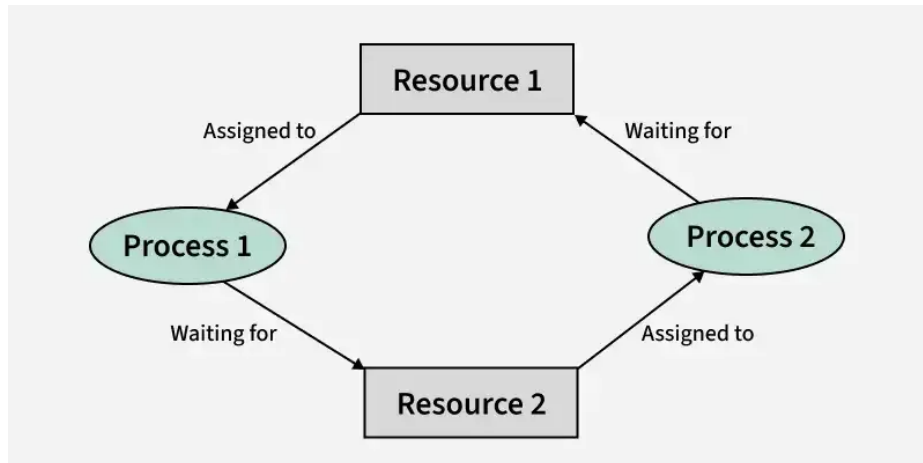
2. If There are Multiple Instances of Resources

Detection of the cycle is necessary but not a sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

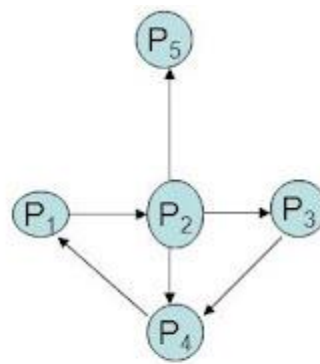
For systems with multiple instances of resources, algorithms like [Banker's Algorithm](#) can be adapted to periodically check for deadlocks.

3. Wait-For Graph Algorithm

The [Wait-For Graph Algorithm](#) is a deadlock detection algorithm used to detect deadlocks in a system where resources can have multiple instances. The algorithm works by constructing a Wait-For Graph, which is a directed graph that represents the dependencies between processes and resources.



Resource allocation graph



Corresponding wait for Graph

Detection Algorithm Steps

1. Check available resources.
2. Find process whose request is less than available resources.
3. Allocate resources and mark process completed.
4. Repeat until no process can proceed.

If unfinished processes remain, deadlock is detected.

Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.

- **Killing The Process:** Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again and keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait conditions.
- **Process Rollback:** Rollback deadlocked processes to a previously saved state where the deadlock condition did not exist. It requires **check pointing** to periodically save the state of processes.
- **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, and preempted resources are allocated to other processes so that there is a possibility of recovering the system from the deadlock. In this case, the system goes into starvation.
- **Concurrency Control:** Concurrency control mechanisms prevent data inconsistencies in systems with multiple concurrent processes. They ensure **that processes do not access the same data simultaneously**, avoiding errors and potential deadlocks. By managing access to shared resources, these mechanisms help maintain system stability and prevent processes from blocking each other.

Banker's Algorithm

Banker's Algorithm is a **deadlock avoidance algorithm** used in **Operating System** to ensure that the system **never enters a deadlock state**.

Definition

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm that **checks whether allocating resources to a process keeps the system in a safe state**.

It is called **Banker's Algorithm** because it works like a **banker giving loans** — the banker gives a loan only if the bank can still satisfy the needs of all customers.

Safe State

A **safe state** is a state where the system can allocate resources to each process **without causing deadlock**.

There exists a **Safe Sequence** in which all processes can finish execution.

Example of Safe Sequence:

P1 → P3 → P4 → P0 → P2

Unsafe State

An **unsafe state** does **not guarantee deadlock**, but **deadlock may occur**.

Deadlock Avoidance

The algorithm **avoids deadlock** by checking system safety **before allocating resources**.

Data Structures Used

The Banker's Algorithm uses the following matrices:

- 1 **Available**: Number of available resources of each type.
- 2 **Max Matrix**: Maximum demand of each process.
- 3 **Allocation Matrix**: Resources currently allocated.
- 4 **Need Matrix**: Remaining resource requirement.

Formula:

$$\text{Need} = \text{Max} - \text{Allocation}$$

Banker's Algorithm Steps

Step 1: Calculate Need Matrix

$$\text{Need} = \text{Max} - \text{Allocation}$$

Step 2: Find a Process

Find a process whose:

$$\text{Need} \leq \text{Available}$$

Step 3: Allocate Resources

If found:

- Add its Allocation to Available
- Mark process as Finished

Step 4: Repeat

Repeat until:

- All processes finish → **Safe State**
- No process found → **Unsafe State**

Advantages

- Avoids deadlock
- Ensures safe resource allocation
- Efficient for systems with known resource limits

Disadvantages

- Requires advance knowledge of **maximum resource needs**
- Not suitable for dynamic systems
- Complex for large systems

CHAPTER-VII

OS SECURITY

Operating System (OS) Security refers to the **mechanisms used to protect system resources, data, and programs from unauthorized access, misuse, or damage**. Modern operating systems include several built-in security features such as **authentication, access control, access rights management, and logging systems** to ensure safe operation.

1. Authentication

Definition

Authentication is the process of **verifying the identity of a user, device, or system** before granting access to system resources.

It ensures that **only legitimate users** are allowed to enter the system.

□ Authentication answers the question:
"Who is the user?"

Purpose of Authentication

Authentication is necessary to:

- Protect confidential data
 - Prevent unauthorized users
 - Maintain privacy
 - Ensure system accountability
 - Reduce risk of cyber attacks
-

Authentication Process

Typical authentication involves:

1. User enters **username**
 2. User enters **password or credentials**
 3. System verifies credentials
 4. Access granted or denied
-

Factors of Authentication

Authentication methods are categorized into **three major factors**:

1. Knowledge Factor (Something You Know)

User provides information known only to them.

Examples:

- Password
- PIN
- Security questions
- Pattern lock

Example in OS:

Logging into a computer using **username and password**.

2. Possession Factor (Something You Have)

User must possess a physical device.

Examples:

- Smart card
- OTP token
- Mobile phone

- ATM card

Example:

Receiving **OTP (One-Time Password)** on mobile.

3. Inherence Factor (Something You Are)

Uses **biometric identification**.

Examples:

- Fingerprint recognition
- Face recognition
- Iris scanning
- Voice recognition

Example:

Fingerprint login in laptops.

Types of Authentication

1. Single-Factor Authentication (SFA)

Uses **only one authentication method**.

Example:

- Password login

Disadvantage:

Less secure.

2. Two-Factor Authentication (2FA)

Uses **two authentication methods**.

Example:

- Password + OTP

Advantage:

More secure than single-factor authentication.

3. Multi-Factor Authentication (MFA)

Uses **two or more authentication factors**.

Example:

- Password + OTP + Fingerprint

Used in:

- Banking systems
 - Government portals
-

Password Security

A **strong password** should:

- Contain uppercase letters
- Contain lowercase letters
- Include numbers
- Include special symbols
- Be at least 8–12 characters long

Example:

Strong Password: C0mputer@123

Weak Password: 12345

Authentication Techniques Used in Operating Systems

- Password-based login
 - Biometric authentication
 - Smart card authentication
 - Network login authentication
-

Advantages of Authentication

- Prevents unauthorized access
 - Protects sensitive information
 - Ensures secure login
 - Maintains user accountability
-

Disadvantages

- Passwords can be forgotten
 - Weak passwords can be hacked
 - Biometric systems are expensive
 - Authentication delays system access
-

2. Access Control

Definition

Access Control is the process of **restricting or allowing access to system resources** based on user permissions.

After authentication, access control determines:

- "What resources can the user access?"
-

Objectives of Access Control

Access control ensures:

- Data confidentiality
 - Data integrity
 - Resource protection
 - Controlled system usage
-

Access Control Process

The general steps include:

1. User authentication
 2. System checks permissions
 3. Access decision made
 4. Resource accessed or denied
-

Access Control Models

1. Discretionary Access Control (DAC)

In DAC:

- Resource owner controls access
- Permissions can be granted to other users
- Flexible but less secure

Example:

File owner shares document with another user.

Advantages of DAC

- Easy to implement
 - Flexible access management
 - Suitable for personal systems
-

Disadvantages of DAC

- Less secure
 - Users may accidentally grant access
 - Vulnerable to malware
-

2. Mandatory Access Control (MAC)

In MAC:

- Access decisions controlled by the system
- Users cannot change permissions
- Used in **high-security environments**

Examples:

- Military systems
 - Government systems
-

Security Levels in MAC

Resources classified into:

- Top Secret
- Secret
- Confidential
- Public

Users can only access resources matching their clearance level.

Advantages of MAC

- Very secure
- Strict access policies
- Prevents unauthorized sharing

Disadvantages of MAC

- Complex implementation
 - Less flexible
 - Expensive to maintain
-

3. Role-Based Access Control (RBAC)

In RBAC:

- Access given based on **user roles**
- Roles define permissions

Example:

Role Permissions

Admin Full control

Faculty Modify files

Student Read-only

Advantages of RBAC

- Easy management

- Scalable
 - Suitable for large organizations
-

Disadvantages of RBAC

- Requires proper role definition
 - Complex in dynamic systems
-

3. Access Rights

Definition

Access Rights are **permissions assigned to users** that define how they can interact with system resources.

These rights control **file and program usage**.

Common Types of Access Rights

1. Read (R)

Allows user to:

- Open file
- View contents

Example:

- Reading text file
-

2. Write (W)

Allows user to:

- Modify file content
- Save changes

Example:

- Editing document
-

3. Execute (X)

Allows user to:

- Run programs

Example:

- Running software application
-

4. Delete (D)

Allows user to:

- Remove file from system
-

5. Append (A)

Allows user to:

- Add new data at end of file
-

File Permission Representation

Permissions are commonly represented using **symbols**:

Symbol	Meaning
--------	---------

r	Read
---	------

w	Write
---	-------

x	Execute
---	---------

-	No permission
---	---------------

Example of File Permissions

`-rwxr-xr--`

Breakdown:

User Category	Permission
---------------	------------

Owner	rwx
-------	-----

Group	r-x
-------	-----

Others	r--
--------	-----

Meaning:

- Owner can read, write, execute
 - Group can read, execute
 - Others can read only
-

Access Control List (ACL)

An **Access Control List (ACL)** is a table that defines:

- Which user can access
- Which operations allowed

Example:

User	File	Permission
------	------	------------

A	file1	Read
---	-------	------

B	file1	Write
---	-------	-------

Admin	file1	Full
-------	-------	------

Importance of Access Rights

- Protects system files
- Prevents accidental deletion
- Ensures data privacy
- Maintains system stability

4. System Logs

Definition

System Logs are **automatically generated records** that store details about system operations, activities, and events.

They help monitor and analyze system behavior.

Purpose of System Logs

System logs help to:

- Track user activities
 - Detect security breaches
 - Identify system errors
 - Monitor performance
 - Troubleshoot problems
-

Information Stored in System Logs

Typical log records include:

- Login and logout times
 - File access events
 - System errors
 - Hardware failures
 - Software crashes
 - Security alerts
 - Network connections
-

Types of System Logs

1. Security Logs

Contain:

- Login attempts
- Failed login attempts
- Password changes
- Unauthorized access attempts

Used for:

- Detecting hacking attempts
 - Security auditing
-

2. System Logs

Contain:

- System startup
- Shutdown records
- Hardware events
- Driver errors

Used for:

- Troubleshooting system failures
-

3. Application Logs

Contain:

- Application errors
- Program activities
- Software crashes

Used for:

- Debugging applications
-

4. Event Logs

Contain:

- Significant system events

- Warning messages
- Failure events

Used for:

- Monitoring system behavior
-

Example of a System Log Entry

Date: 13-04-2026

Time: 09:35 AM

User: student01

Action: Login Failed

Reason: Incorrect Password

IP Address: 192.168.1.10

Log Management

System administrators perform:

- Log monitoring
 - Log backup
 - Log analysis
 - Log deletion (after storage limit)
-

Advantages of System Logs

- Detect intrusions
 - Maintain audit trail
 - Improve security
 - Help forensic investigation
 - Support system maintenance
-

Disadvantages

- Requires storage space
 - Needs regular monitoring
 - Large logs become difficult to analyze
-

Relationship Between Authentication, Access Control, and Access Rights

These components work together:

1. **Authentication** → verifies user identity
 2. **Access Control** → checks permission rules
 3. **Access Rights** → defines allowed operations
 4. **System Logs** → records all activities
-

Real-Life Example

Consider a **college computer lab**:

1. Student logs in using **username and password**
→ Authentication
 2. System checks student permissions
→ Access Control
 3. Student allowed to **read files only**
→ Access Rights
 4. Login time recorded in system
→ System Logs
-

Advantages of OS Security Mechanisms

- Protects confidential data

- Prevents unauthorized access
 - Improves reliability
 - Maintains privacy
 - Supports auditing
-

Disadvantages

- Increases system complexity
- Requires management effort
- May slow system performance
- Needs user training