# unit 3

# Memory Management

## Memory management :

Memory management is a crucial component of an operating system (OS) that oversees and coordinates how memory is allocated, used, and freed up. It ensures that processes have enough memory to run efficiently without interfering with one another, and it manages memory resources to optimize system performance. Here's an introduction to the fundamentals of memory management in an OS:

1. Goals of Memory Management

Efficiency: To use the limited memory available optimally.

Isolation and Protection: To protect each process's memory space from interference by others.

Multi-Tasking: To allow multiple processes to run concurrently by sharing memory.

Data Integrity and Security: To prevent unauthorized access to memory by isolating processes.

## Address Binding:

Address Binding is a process in operating systems that maps logical addresses (generated by a program) to physical addresses (used by the memory). This mapping is crucial for program execution, as it allows programs to use memory efficiently without knowing the exact physical location in memory where they'll be loaded.

Types of Address Binding

Compile-Time Binding:

If it’s known in advance where the program will reside in memory, the compiler generates absolute addresses (direct physical addresses).

Example: Embedded systems often use compile-time binding since programs typically run from the same physical memory address each time.

Load-Time Binding:

If the memory location of a program isn’t known at compile time, binding occurs when the program is loaded into memory. The OS assigns the addresses when loading the program.

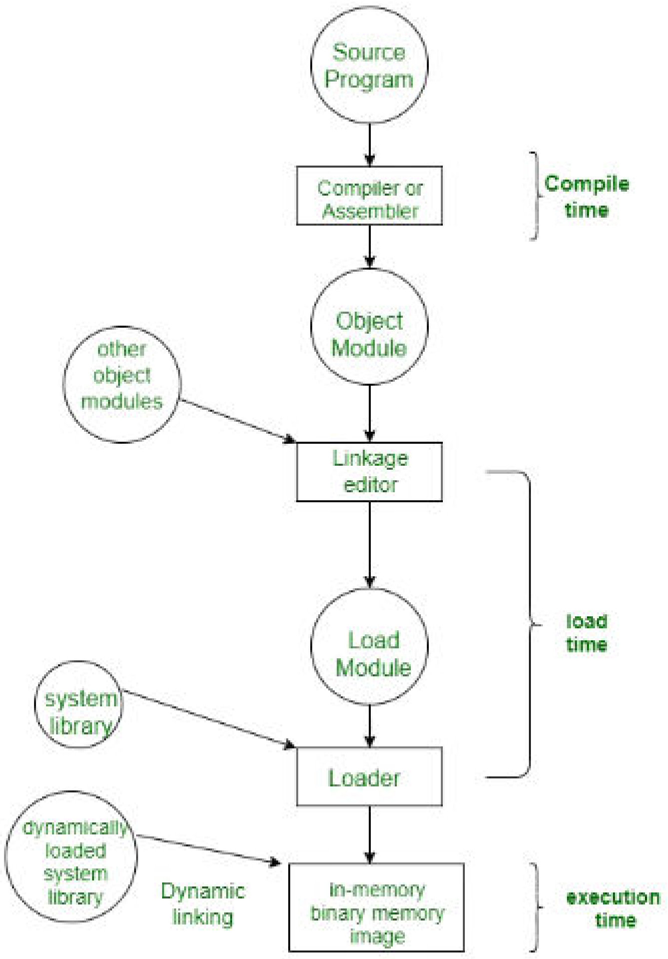
Example: Desktop applications might use load-time binding to adapt to available memory.

Execution-Time Binding:

When a program might be moved in memory during execution (due to swapping or dynamic relocation), address binding occurs at runtime.

This type is flexible, allowing a program to run from different memory addresses at different times.

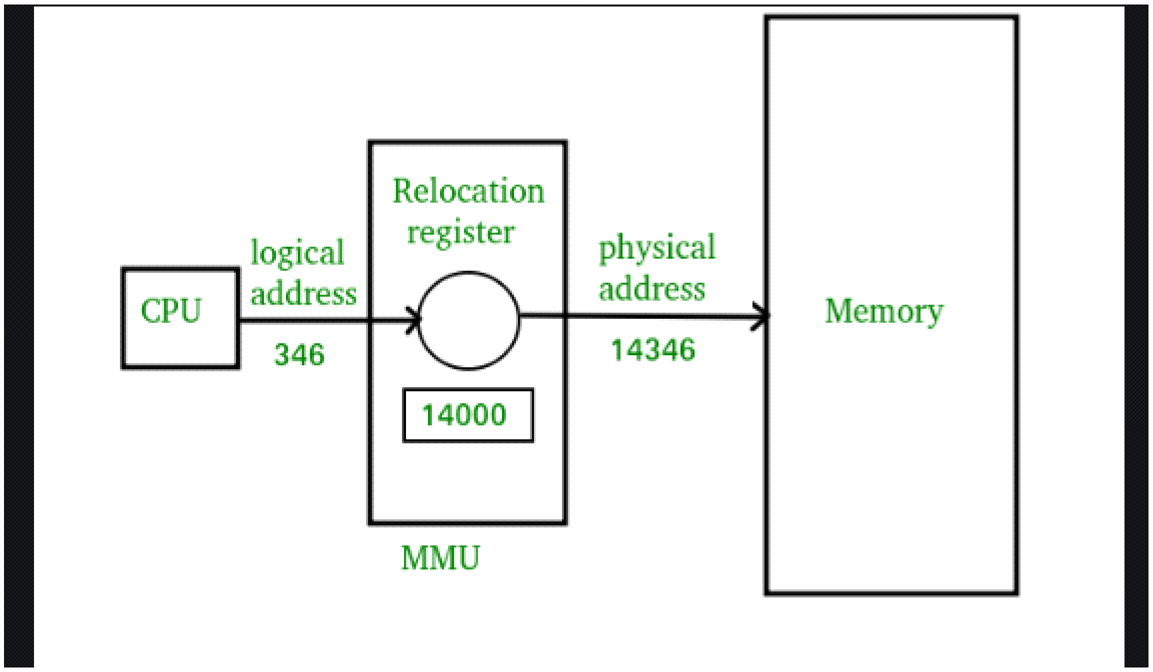
Example: Modern OSs use execution-time binding, allowing programs to run in any available memory location.



Logical vs. Physical Address

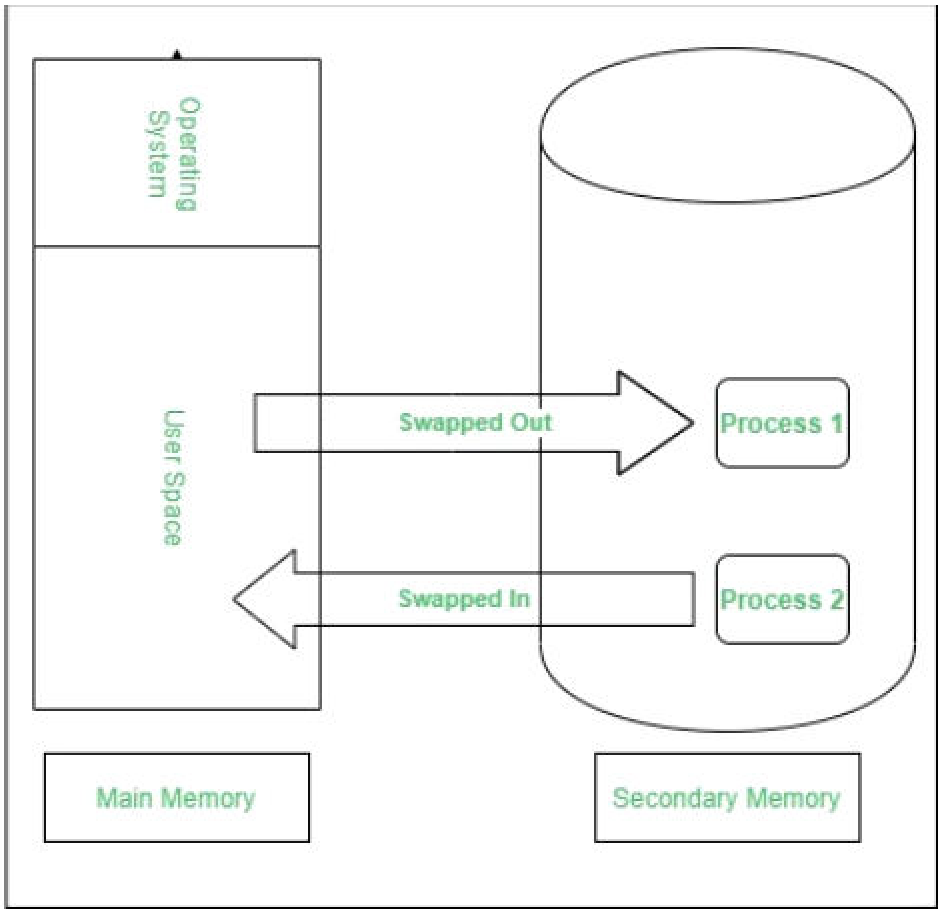
Logical Address: Also known as a virtual address, this is the address generated by the CPU during program execution. It is a reference to a memory location independent of the actual physical location. Logical addresses are used by programs to access memory and are part of the address space that a program uses.

Physical Address: This is the actual address in the RAM, which the memory unit recognizes. It is the real address in hardware memory. While the logical address space is visible to the program, the physical address space is used by the memory management system to execute programs.



## Swapping

Swapping is a memory management technique in which a process is temporarily moved from main memory (RAM) to secondary storage (disk) to free up memory for other processes. When the process is needed again, it is loaded back into main memory. Swapping allows more processes to run concurrently by making better use of limited RAM, especially when there is not enough physical memory to hold all active processes.



Goal: To maximize the use of RAM by moving inactive or lower-priority processes to disk.

Swapping Process:

If memory is full and a new process needs to run, an inactive process may be "swapped out" (moved to disk).

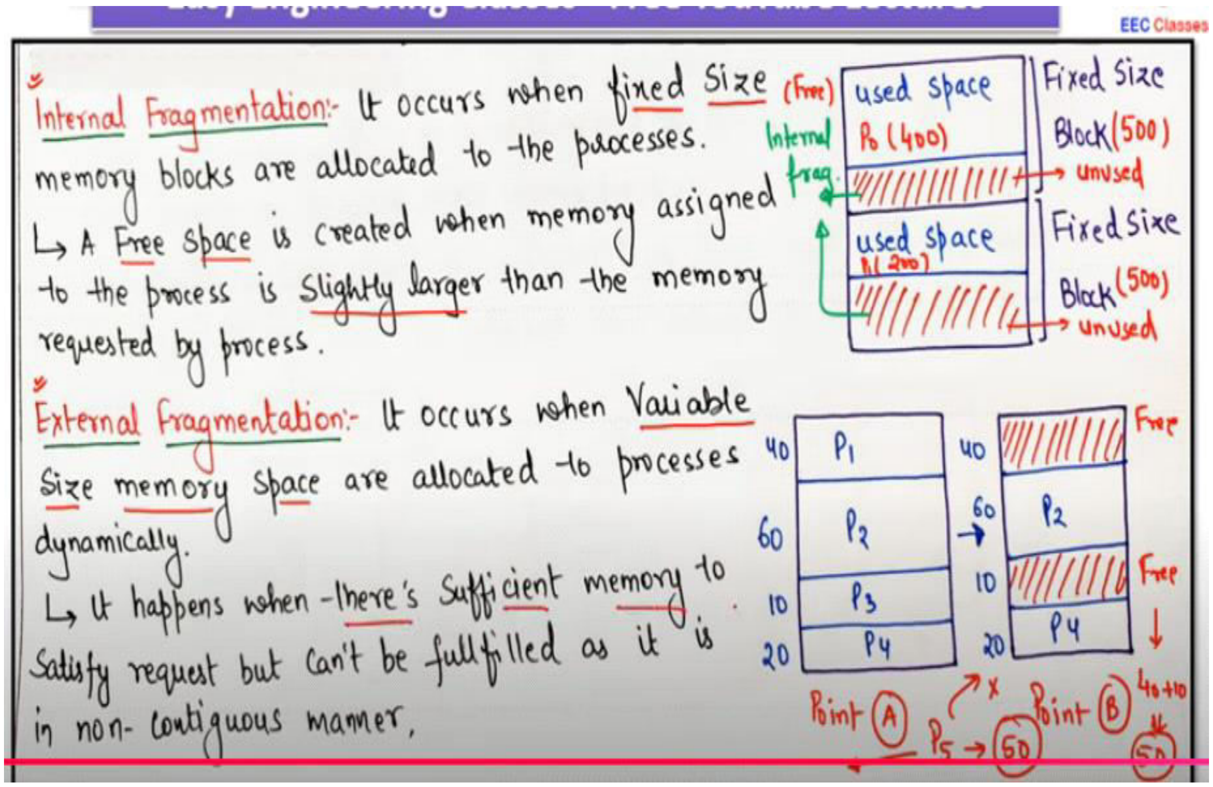
When needed again, the process is "swapped in" (brought back into memory).

Drawback: Swapping can be slow, as moving data between RAM and disk storage is time-consuming compared to RAM-only operations.

## What is Fragmentation?

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

There are two types of fragmentation: internal and external



Internal Fragmentation:- It occurs when fixed Size memory blocks are allocated to the processes.

↳ A Free Space is created when memory assigned to the process is Slightly larger than the memory requested by process.

External Fragmentation:- It occurs when Vaiable Size memory Space are allocated to processes dynamically.

↳ It happens when there's Sufficient memory to Satisfy request but can't be fullfilled as it is in non- contiguous manner,

1. Internal Fragmentation

Meaning: Wasted space inside a memory block.

Cause: When a process doesn't use all the space given to it.

Example:

Imagine you have memory blocks of 100 KB each.

A process only needs 90 KB, but it's given a full 100 KB block.

The extra 10 KB in the block is wasted space, creating internal fragmentation.

2. External Fragmentation

Meaning: Wasted space between memory blocks.

Cause: When small, scattered free spaces form between memory blocks after allocating and freeing memory.

Example:

Suppose you have 500 KB of free memory in total but in small gaps (e.g., 100 KB, 50 KB, 200 KB, 150 KB).

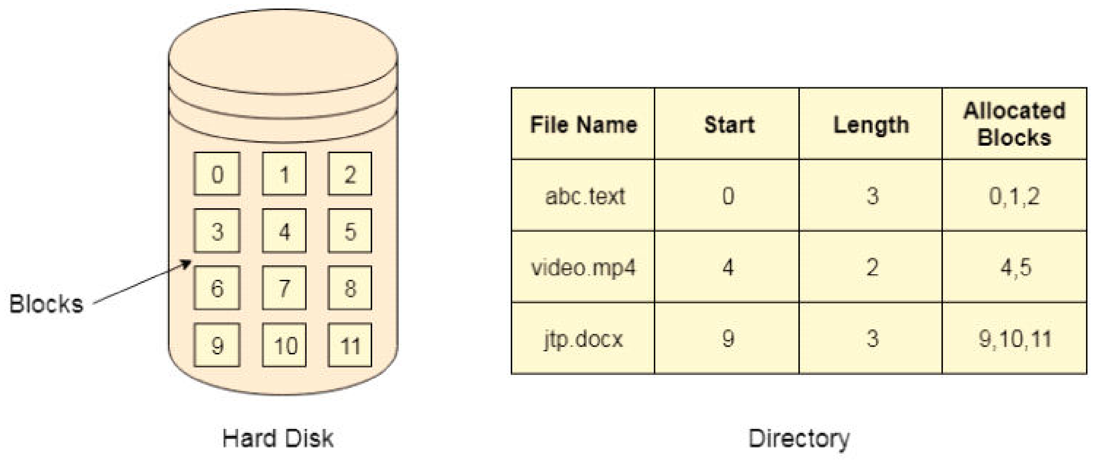
A new process needs 300 KB, but it can't fit into any of these small spaces, even though the total free memory is enough.

These small scattered gaps cause external fragmentation.

**There are mainly two types of memory allocation: contiguous and non-contiguous memory allocation.**

**1. Contiguous Memory Allocation**

In Contiguous Memory Allocation, each process is given a single, uninterrupted block of memory. The entire process is stored in one continuous area in the memory, and each block must be large enough to fit the whole process.



| File Name | Start | Length | Allocated Blocks |
|---|---|---|---|
| abc.text | 0 | 3 | 0,1,2 |
| video.mp4 | 4 | 2 | 4,5 |
| jtp.docx | 9 | 3 | 9,10,11 |

Contiguous Allocation

How It Works:

When a process arrives, the OS searches for a large enough empty memory block to fit the entire process.

Once a suitable block is found, the process is loaded into that contiguous space.

For example, if a process needs 200 MB of memory, the OS finds a 200 MB (or larger) free block and assigns it to that process.

Advantages:

Simple to Manage: Since each process is in a single block, it’s easy for the OS to track and access.

Fast Access: Accessing memory is faster because the CPU doesn’t need to jump between locations.

Disadvantages:

External Fragmentation: As processes are added and removed, gaps form in memory. These gaps may not be usable for new processes if they’re too small, even though the total memory available could be enough.

Limited Flexibility: If there isn’t a contiguous block large enough for a new process, it may be rejected, even if the total free memory is adequate.

Example:

Suppose we have 1 GB of free memory, and a new process needs 300 MB. If there’s a contiguous 300 MB block available, the process is loaded into that space.

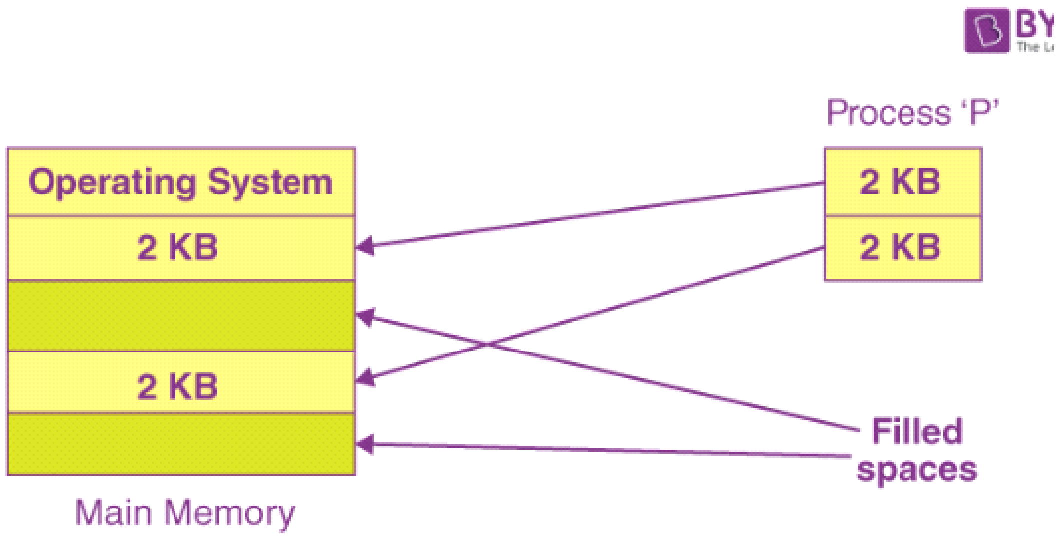
If memory is fragmented, like in this example:

100 MB free block, 50 MB used, 200 MB free, 100 MB used, 500 MB free.

Even though the total free memory is 800 MB, the OS might reject the process because it can’t find one continuous block of 300 MB.

**2. Non-Contiguous Memory Allocation**

In Non-Contiguous Memory Allocation, a process is split into smaller parts (pages or segments) that can be stored in different, non-adjacent locations in memory. This approach uses smaller blocks of memory more flexibly.



How It Works:

The OS divides the process into equal-sized blocks, called pages. Memory is divided into small blocks, called frames.

Each page of the process can be loaded into any available memory frame, even if the frames are not consecutive.

The OS uses a page table to keep track of where each page is stored, allowing the process to be reconstructed when needed.

Advantages:

Better Memory Utilization: Since the OS can use small, scattered free spaces, there's less wasted memory, especially when large blocks aren't available.

Greater Flexibility: Processes are less likely to be rejected due to a lack of large, contiguous memory, as they can fit into smaller spaces.

Disadvantages:

Complex Management: The OS needs to maintain a page table to track each page's physical location, which requires more processing time and resources.

Slower Access: Access might be a bit slower than contiguous allocation due to the need to reference the page table.

Example:

Suppose a process needs 300 MB, divided into three 100 MB pages (Page A1, A2, and A3).

Memory could be arranged like this:

Frame 1: A1 (100 MB)

Frame 5: A2 (100 MB)

Frame 9: A3 (100 MB)

The OS keeps a page table to remember which page is in which frame.

Since each part is stored independently, even if memory has many small gaps, the process can still be loaded.

## Compaction

Compaction is a technique used to manage external fragmentation in memory. It involves rearranging memory contents so that all free memory is grouped together in one large block, making it easier to allocate larger memory requests.

How It Works:

When external fragmentation occurs, the OS shifts processes in memory to remove gaps between them.

After compaction, free memory is consolidated at one end of the memory space, making it possible to fit larger processes.

Advantages:

Reduces external fragmentation by creating a single large free block.

Enables more efficient memory allocation for larger processes.

Disadvantages:

Compaction is a time-consuming operation, as the OS has to move each process in memory.

It may not always be possible, especially if processes are not relocatable.

Example:

Suppose we have memory allocated like this:

Process A: 100 MB, free space: 50 MB, Process B: 150 MB, free space: 100 MB, Process C: 200 MB.

If a new process needs 120 MB, it can’t fit due to fragmented free space.

By shifting Process B and Process C up to remove the gaps, we get 150 MB of free space in one block, allowing the new process to be allocated.

**Virtual Memory** is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.

A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

**There are two main types of virtual memory:**

**Paging**

**Segmentation**

## Paging

Paging is a memory management technique that divides both the process's memory and physical memory into fixed-size blocks, called pages and frames respectively. Each page of a process can be placed in any free frame in memory, allowing non-contiguous memory allocation and reducing external fragmentation.

How It Works:

The OS divides a process into equal-sized pages (e.g., 4 KB each) and divides physical memory into frames of the same size.

Pages of a process are loaded into available frames, even if they're scattered across the memory.

The OS uses a page table to keep track of where each page is stored in memory.

Advantages:

Avoids external fragmentation, as any free frame can be used.

Efficiently utilizes memory by filling in free frames scattered across memory.

Disadvantages:

May have internal fragmentation if a page doesn't fully use its frame.

Adds complexity, as the OS must manage a page table to keep track of each page's location.

Example:

Suppose a process needs 12 KB of memory. If each page is 4 KB, the process will be divided into three pages (P1, P2, P3).

If memory has three available frames scattered across memory (Frame 4, Frame 10, Frame 15), the pages can be stored as follows:

Page P1 → Frame 4

Page P2 → Frame 10

Page P3 → Frame 15

The OS maintains a page table to know which page is in which frame, allowing the process to be reassembled when needed.

## Segmentation

Segmentation is a memory management technique where a process is divided into variable-sized segments based on the logical structure of the program, like code, data, and stack segments. Unlike paging, where all pages are the same size, segments are different sizes depending on the process's needs.

How It Works:

A process is split into segments according to its functional parts (e.g., code, data, stack).

Each segment is allocated memory as a separate, contiguous block, but segments themselves can be scattered in memory.

The OS uses a segment table to keep track of the base address and size of each segment.

Advantages:

Allows memory allocation based on the logical structure of the process, which can lead to more efficient memory usage.

Helps keep related data together in the same segment, making it easier to manage logically related data.

Disadvantages:

Can suffer from external fragmentation because each segment needs a contiguous block of memory.

Requires a segment table, adding to the OS's complexity.

Example:

Suppose a process has three segments:

Segment 1 (Code): 100 KB

Segment 2 (Data): 200 KB

Segment 3 (Stack): 50 KB

In memory, each segment can be placed in different locations:

Segment 1 might be placed starting at address 1000.

Segment 2 might start at address 2000.

Segment 3 might start at address 3000.

The OS maintains a segment table with the base address and size of each segment, allowing it to locate each part of the process in memory.

## What is Demand Paging?

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start. A page fault occurred when the program needed to access a page that is not currently in memory.

The operating system then loads the required pages from the disk into memory and updates the page tables accordingly. This process is transparent to the running program and it continues to run as if the page had always been in memory.

## What is Page Fault?

The term “page miss” or “page fault” refers to a situation where a referenced page is not found in the main memory.

When a program tries to access a page, or fixed-size block of memory, that isn’t currently loaded in physical memory (RAM), an exception known as a page fault happens. Before enabling the program to access a page that is required, the operating system must bring it into memory from secondary storage (such a hard drive) in order to handle a page fault.

In modern operating systems, page faults are a common component of virtual memory management. By enabling programs to operate with more data than can fit in physical memory at once, they enable the efficient use of physical memory. The operating system is responsible for coordinating the transfer of data between physical memory and secondary storage as needed.

## What is Demand Paging?

Demand Paging means that parts of a program are loaded into memory only when the program actually needs them. This saves memory space, but it can also affect the speed of the system.

**How Demand Paging Affects Performance:**

**Page Fault Rate (Missing Data Problem):**

When a program needs a part (page) that isn’t in memory, it creates a page fault, meaning the system has to get that data from the hard drive.

More page faults make the system slower because getting data from the hard drive is slow.

Fewer page faults mean smoother performance, as most needed data is already in memory.

**Disk Access Speed (Hard Drive vs. RAM Speed):**

Hard drives are much slower than RAM, so whenever a page is missing, the program has to wait while the page is loaded from the hard drive into memory.

Faster storage (like SSDs) helps reduce this delay, but it's still slower than RAM.

**Locality of Reference (Using Nearby Data):**

Programs usually access data that is close to other data they just used.

Spatial locality means the program is accessing data near what it just accessed.

Temporal locality means the program is reusing recent data.

Demand Paging works well here because only nearby or recent pages are loaded, which reduces page faults.

**Working Set Size (Current Needed Data):**

The working set is the set of pages the program is using right now.

If the working set can fit in RAM, demand paging is fast because most needed data is already in memory.

If the working set is too big for RAM, the system may start thrashing—constantly loading and unloading pages, which slows everything down.

**Thrashing (Too Many Page Faults):**

Thrashing happens when there are too many page faults, and the system spends most of its time moving pages between RAM and the hard drive.

This causes a huge slowdown because the CPU is often waiting on data instead of doing actual work.

Page Replacement Algorithms are used in operating systems to manage the pages in memory. When a page fault occurs (meaning the requested page is not in memory), the system has to bring in a new page. But if the memory is already full, it must remove an existing page to make space. The Page Replacement Algorithm decides which page to remove.

## Here are a few common, simple algorithms:

1. First-In-First-Out (FIFO)

The oldest page in memory (the one that entered first) is removed first.

Simple but can be inefficient, as the oldest page may still be in use.

Example: If pages are loaded in the order 1, 2, 3, 4, and memory is full, then when page 5 needs to load, page 1 (the oldest) is removed.

2. Least Recently Used (LRU)

This removes the page that hasn't been used for the longest time.

More efficient because it tries to keep frequently used pages in memory.

Example: If pages 1, 2, and 3 are in memory, and page 2 hasn't been used recently, LRU will remove page 2 if a new page needs space.

3. Optimal Page Replacement

This removes the page that will not be used for the longest time in the future.

Most efficient but not practical, as it requires knowing future page requests.

Example: If the system could predict that page 4 won't be used for a long time, it would replace page 4.

4. Least Frequently Used (LFU)

This removes the page that has been used least often.

Keeps frequently accessed pages longer in memory.

Example: If page 1 has been used 10 times and page 2 only twice, LFU would remove page 2.

5. Clock Algorithm (Second-Chance Algorithm)

This gives each page a second chance if it's been used recently.

It cycles through pages in a circular manner, skipping pages that have been used recently and only removing those that haven't been accessed.

Example: Pages are arranged in a circle. If a page was recently used, the algorithm gives it a "second chance" by skipping it.

## **File Management:**

The File System in an operating system manages files and organizes how data is stored, retrieved, and managed. It provides a structured way to save data, access it efficiently, and ensure security.

Here's a breakdown of the main concepts:

1. File Attributes

Each file has several attributes, which are pieces of metadata describing the file. Common file attributes include:

Name: The human-readable name of the file.

Type: The file type or format (e.g., .txt, .jpg, .mp3).

Location: The path or address of the file on the storage device.

Size: The size of the file in bytes.

Protection/Permissions: Defines who can read, write, or execute the file.

Creation/Modification Date: Timestamps showing when the file was created or last modified.

Owner: The user who owns or created the file.

2. File Operations

File systems allow several basic operations on files:

Create: Making a new file with a unique name and an initial location in storage.

Open: Accessing a file so it can be read or modified.

Read: Accessing the data stored within the file.

Write: Adding new data or modifying existing data in the file.

Delete: Removing the file from the storage permanently.

Close: Releasing the file after operations are complete, freeing up resources.

Rename: Changing the name of the file.

Append: Adding data at the end of a file without altering existing content.

3. File Types

File systems support various file types, usually recognized by their extensions, which tell the OS and applications how to handle them:

Text Files (.txt, .md): Contain plain text data.

Binary Files (.exe, .bin): Files that store binary data, usually program executables.

Image Files (.jpg, .png, .bmp): Files that store graphical data.

Audio Files (.mp3, .wav): Files with sound data.

Video Files (.mp4, .avi): Files with visual and audio data.

Document Files (.docx, .pdf): Files for documents with structured data, readable by certain applications.

System Files (.dll, .sys): Files essential for operating system functionality, typically hidden from the user.

4. File Organization & Structure

File systems also define how files are organized within directories and subdirectories, making it easier to store, locate, and manage data on storage devices. Different file systems (like FAT, NTFS, ext4) offer different ways of organizing and securing files.

**The file system in an operating system is responsible for organizing, managing, and accessing data on storage devices. Here are its main functions:**

1. File Creation, Storage, and Management

Create and Store: The file system allows users to create new files and store them efficiently on a storage device.

Manage Attributes: It keeps track of file metadata, such as name, type, size, location, and permissions.

2. Directory Management

Organize Files: The file system organizes files into directories (or folders) and subdirectories, creating a hierarchical structure that's easy to navigate.

Path Resolution: It resolves file paths so that users and programs can easily locate files.

3. File Access and Retrieval

Read and Write Operations: The file system facilitates reading from and writing to files.

Access Control: It enforces file access permissions, ensuring users and applications access files only as authorized (read, write, execute permissions).

4. File Security and Protection

Control Permissions: The file system manages permissions to ensure only authorized users can access or modify files.

Encryption: Some file systems offer built-in encryption to protect data from unauthorized access.

5. File Deletion and Cleanup

Delete Files: It provides functions to delete files and directories.

Reclaim Space: After a file is deleted, the file system reclaims its storage space for future use.

6. Efficient Disk Space Management

Allocate Space: The file system allocates space for files in a way that minimizes fragmentation.

Track Free Space: It keeps a record of free space and reuses it for new files.

7. File Naming and Identification

File Naming Rules: It sets rules for naming files (e.g., allowed characters, length).

Unique Identification: Each file is uniquely identified by its path, enabling quick access.

8. Data Integrity and Recovery

Error Checking: Many file systems include error-checking mechanisms to detect and correct data corruption.

Recovery Tools: In the event of crashes or unexpected shutdowns, file systems may offer recovery options to restore files.

9. Access Control and File Sharing

Control Sharing: The file system allows controlled access to files and supports sharing across users and applications.

Concurrent Access Handling: It manages simultaneous access to files, ensuring data consistency in multi-user environments.

## What is a File System?

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

FAT (File Allocation Table): An older file system used by older versions of Windows and other operating systems.

NTFS (New Technology File System): A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.

ext (Extended File System): A file system commonly used on Linux and Unix-based operating systems.

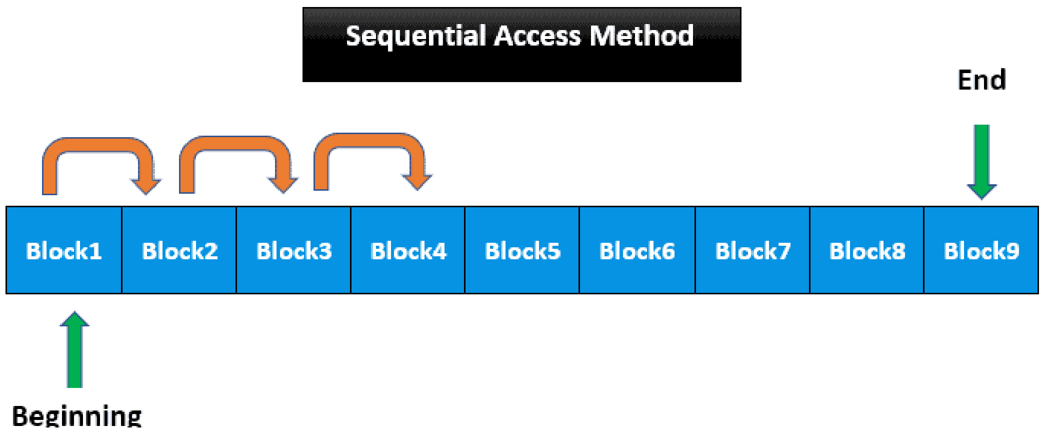
HFS (Hierarchical File System): A file system used by macOS.

APFS (Apple File System): A new file system introduced by Apple for their Macs and iOS devices.

**Access methods are the ways in which data is retrieved or accessed from storage, such as a database, file system, or memory. There are various methods to access data, and they can be broadly categorized as follows:**

1. Sequential Access

Definition: In sequential access, data is accessed in a predefined, linear order, meaning it can only be read or written one record at a time, starting from the beginning.



Use case: This method is ideal when accessing data that must be processed in sequence, such as processing a log file or reading a tape storage device.

Characteristics:

Slower access time since data must be read in order.

Good for large volumes of data that are accessed in sequence.

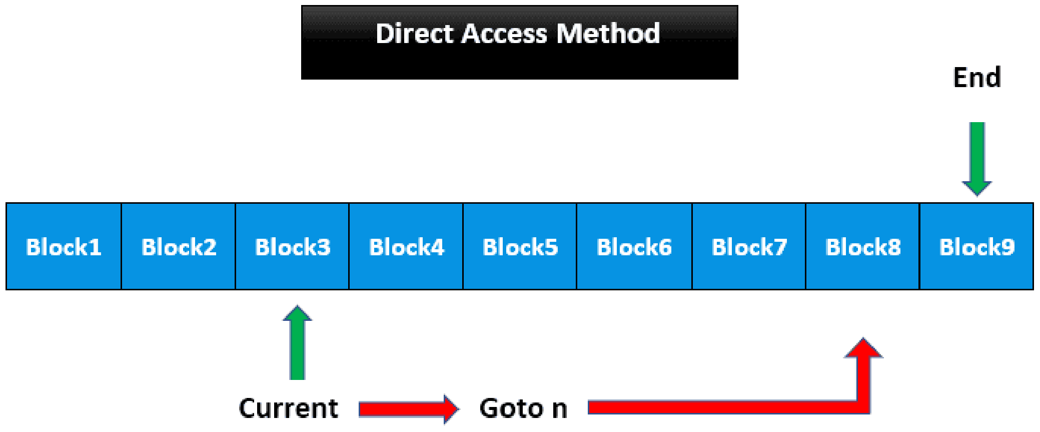
Examples:

Magnetic tapes (old storage medium).

Reading from a text file line by line.

2. Direct (Random) Access

Definition: Direct or random access allows data to be accessed at any point directly, without needing to follow a specific order. The location of the data is known, and it can be read or written immediately.



Use case: Used when fast, non-sequential access to data is needed, like databases or hard disk drives (HDDs).

Characteristics:

Faster access times since data can be accessed directly at any location.

More efficient when accessing specific records out of order.

Examples:

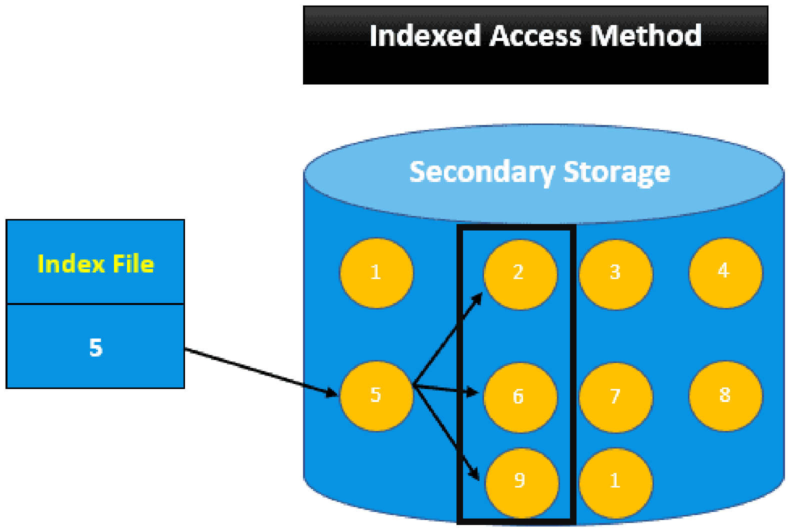
Hard drives (HDDs), SSDs.

Databases with indexed records.

RAM (Random Access Memory).

3. Indexed Access

Definition: In indexed access, an index or lookup table is used to find the location of the data. The index contains references to the actual data, and the system uses the index to retrieve specific records quickly.



Use case: This method is ideal for large datasets that need to be searched quickly, like in databases.

Characteristics:

Faster than sequential access as it involves searching through an index instead of scanning through all data.

Often used in combination with direct access.

Examples:

Relational database indexes (e.g., B-trees).

Filesystems with indexed directories.

4. Hashed Access

Definition: Hashed access uses a hash function to map data to a specific location in memory or storage. This ensures a very fast data retrieval time.

Use case: Used in situations where quick lookups or exact matches are needed.

Characteristics:

Very fast access for operations like searching for a specific record or inserting new records.

Often used in hash tables or hash maps.

Examples:

Hash tables in programming (e.g., Python dictionaries).

Hash-based database indexing.

5. Clustered Access

Definition: Clustered access involves storing related records together, so when a record is retrieved, associated records are likely retrieved as well.

Use case: This method is useful in situations where you want to optimize access to related data, such as in data warehousing.

Characteristics:

Improves performance when accessing related records together.

Can be slower for non-clustered data access.

Examples:

Clustered indexing in databases.

6. Content Addressable Storage

Definition: In content addressable storage, data is accessed by its content or hash value, not by its location.

Use case: This method is commonly used in systems that require integrity verification, such as distributed file systems.

Characteristics:

Data is accessed based on its content rather than its memory address or file path.

Ensures data integrity and is useful in distributed environments.

Examples:

Git repositories.

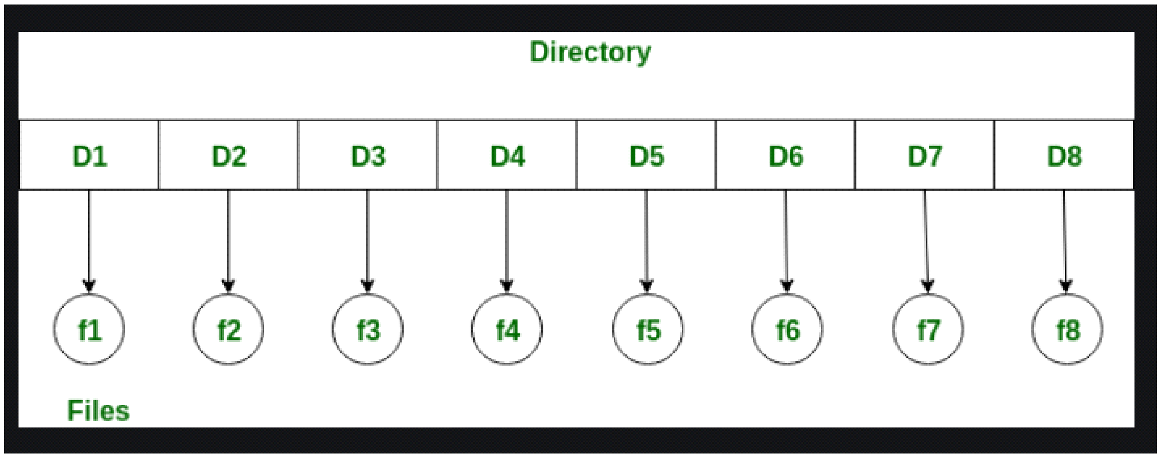
Distributed file systems like IPFS (InterPlanetary File System).

## <u>Different Types of Directory in OS</u>

In an operating system, there are different types of directory structures that help organize and manage files efficiently.

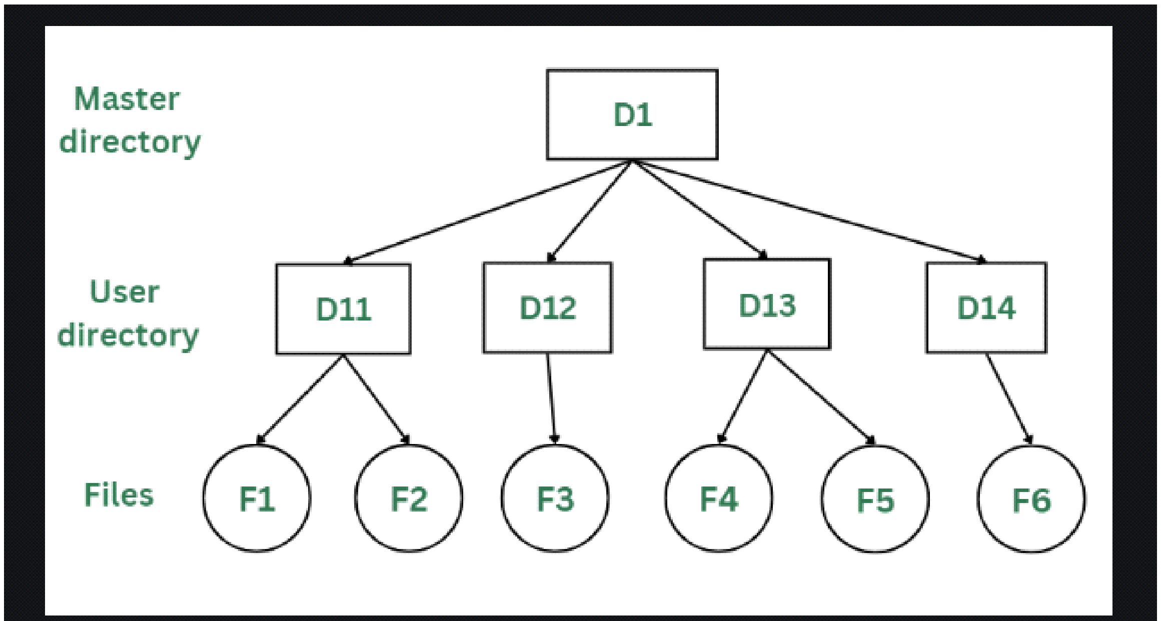
1. Single-Level Directory

All files are stored in a single directory. Access to files is done by referencing the file name directly from this single directory.



2. Two-Level Directory

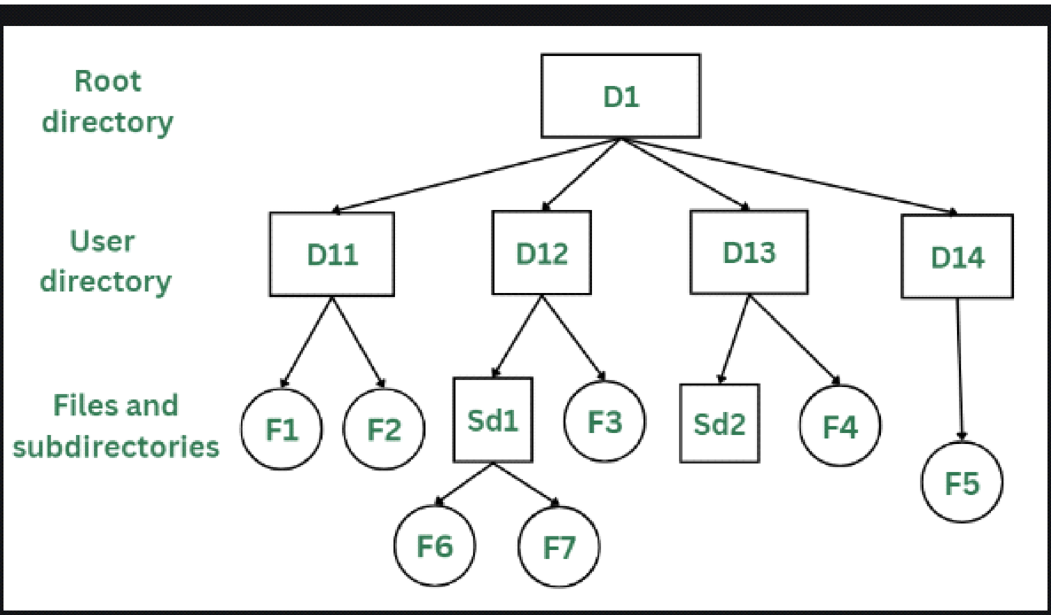
The first level contains user directories, and the second level contains files within each user’s directory. Access is based on both the user directory and the file name.



Two-Levels Directory Structure

3. Tree-Structured Directory (Hierarchical)

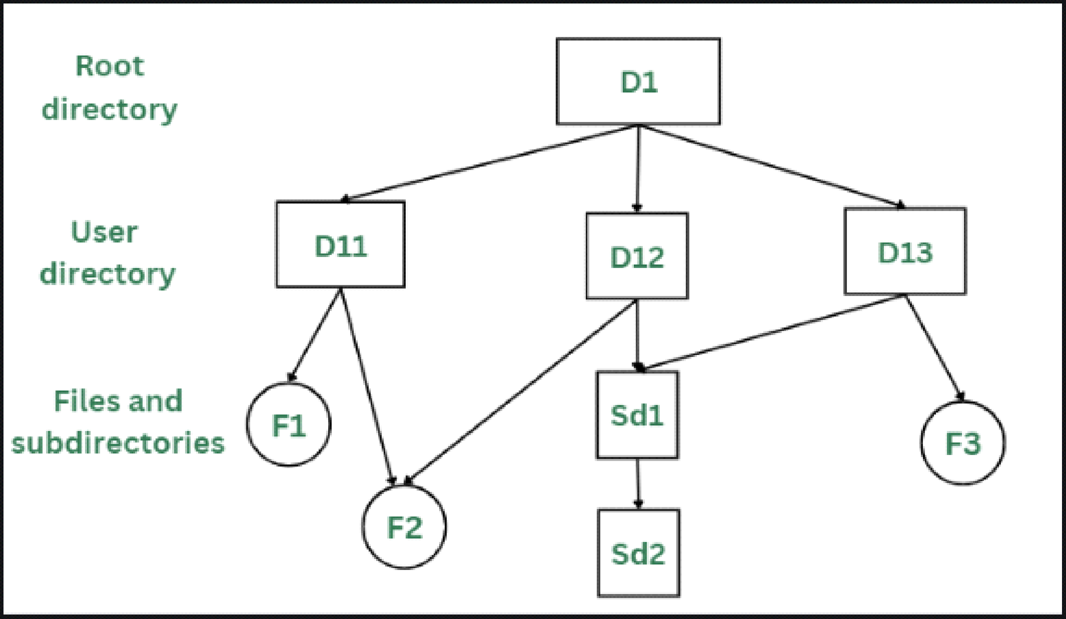
Directories can have subdirectories. Access is done by traversing through the directory levels, starting from the root directory down to the specific file or subdirectory.



Tree/Hierarchical Directory Structure

4. Acyclic Graph Structure

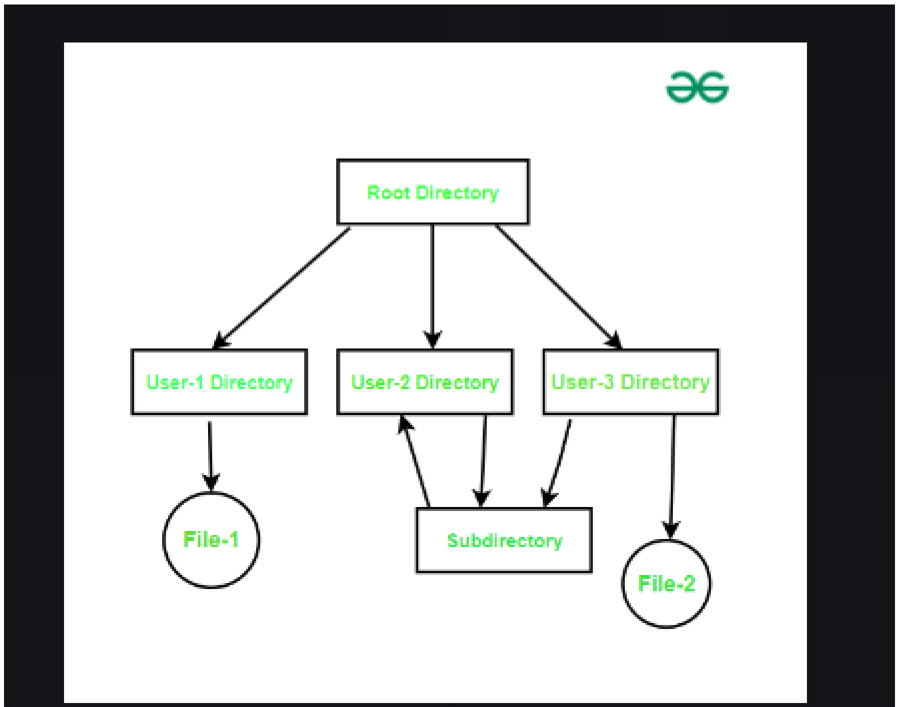
Similar to the tree structure but allows a file or directory to have more than one parent. Access is done by traversing the graph, which may involve cycles or shared subdirectories, ensuring no circular references.



Acyclic Graph Structure

5. General-Graph Directory Structure

Directories and files can have multiple parent directories, allowing more complex relationships. Access involves navigating through a general graph, where files and directories can be reached through different paths.



General Graph Directory Structure

**In operating systems, allocation methods are used to manage how files are stored on disk. These methods determine how the blocks of data for files are allocated and organized on storage media**.

1. Contiguous Allocation

Description: In contiguous allocation, each file is stored in a set of contiguous blocks on the disk. The file occupies a sequence of adjacent blocks, and the operating system keeps track of the starting block and the length of the file.

How it works: When a file is created, the system allocates a contiguous block of space that is large enough to hold the entire file. Accessing the file is straightforward because the file's data is stored in one continuous block.

2. Linked Allocation

Description: In linked allocation, each file is stored in a linked list of blocks. Each block of the file contains a pointer to the next block, which can be located anywhere on the disk.

How it works: The operating system maintains the first block of the file, which contains a pointer to the next block, and so on. Accessing a file involves following the chain of pointers from one block to the next until the entire file is retrieved.

3. Indexed Allocation

Description: In indexed allocation, an index block is used to store the addresses of the blocks that make up a file. Instead of each block pointing to the next one (as in linked allocation), an index block contains a list of all the blocks that are part of the file.

How it works: The operating system maintains an index block for each file, which holds the addresses of all the file's data blocks. Accessing the file involves reading the index block first to find the data blocks scattered throughout the disk.