UNIT – 4 : FILE SYSTEMS AND ITS IMPLEMENTATION

File System Interface:

The operating system defines a logical storage unit called a file. The files are stored in disk blocks in the disk. The mapping of files onto the disk physical devices is done by the operating system. The physical devices are nonvolatile. Information is stored on different storage media. For example, hard disks, pen drives etc. are used to store information. In a disk, data are stored in small units called as disk blocks. That is, the disk is logically divided into disk blocks in which data are stored. The user need not be aware that there are disk blocks in the disk where information is stored. It is enough for the users to understand information in terms of files.

File:

A file is a named collection of related information that is recorded on secondary storage. The file is the smallest allotment on secondary storage. A file may represent programs or data. That is, a file may be a program file or a data file. The program files can be source programs, objects programs and so on. The data files can have numeric data, alphabetic data, alphanumeric data, binary data and so on.

A file has a defined structure depending on the type of the file. For example, the text file is a sequence of characters organized into lines. A source file has a source program that has a sequence of subroutines and functions, organized as declarations followed by executable statements. An object file has a sequence of bytes organized into blocks understandable by the linker. An executable file has a series of code sections that the loader can bring into the memory and execute.

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:

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

The file system consists of two distinct parts:

- A collection of files that store related data.
- A directory structure that organizes and provides information about all the files in the system.

Issues Handled by File System:

We've seen a variety of data structures where the file could be kept. The file system's job is to keep the files organized in the best way possible.

A free space is created on the hard drive whenever a file is deleted from it. To reallocate them to other files, many of these spaces may need to be recovered. Choosing where to store the files on the hard disc is the main issue with files one block may or may not be used to store a file. It may be kept in the disk's non-contiguous blocks. We must keep track of all the blocks where the files are partially located.

File Concepts:

1. File Name & Extension:

Files are named to make it easy for users to refer to them. A file name is typically a string of characters, such as "filename.cpp", which includes an extension that identifies the file format. Some systems, like Linux, are case-sensitive regarding file names, while others are not. The name of the file is divided into two parts name & extension, separated by a period.

Once a file is named, it becomes independent of the user, process, or system that created it. For example, one user might create a file called "filename.cpp", while another might edit it using its name. The file's owner could also write it to a CD, send it via email, or copy it across a network, and it would still be known as "filename.cpp" on the destination system.

2. Fundamental Components of a File:

A file's attributes vary from one OS to another but typically consist of these:

- **Name:** Name is the symbolic file name and is the only information kept in human-readable form.
- **Identifier:** This unique tag is a number that identifies the file within the file system; it is in the non-human-readable form of the file.

- **Type:** This information is needed for systems that support different types of files or their formats.
- **Location:** This information is a pointer to a device pointing to the file's location on the device where it is stored.
- **Size:** The current file size (which is in bytes, words, etc.), possibly the maximum allowed size, gets included in this attribute.
- **Protection:** Access-control information establishes who can do the reading, writing, executing, etc.
- **Date, Time, and user identification:** This information might be kept for creating the file, its last modification, and its previous use. These data might be helpful in the field of protection, security, and monitoring its usage.

3. File Path: A file path is the unique location of a file within the file system hierarchy. It specifies the directory or directories that must be traversed from the root directory to locate a specific file. File paths can be absolute (starting from the root directory) or relative (relative to the current working directory).

Example:

- Absolute Path (Unix/Linux): **/home/user/documents/example.txt**
- Relative Path: **../images/picture.jpg** (going up one directory, then into the **images** directory)

4. File Size: File size refers to the amount of storage space a file occupies on the storage device. It is typically measured in bytes, kilobytes (KB), megabytes (MB), gigabytes (GB), etc.

5. File Operations: A file is a type of data that is abstract. To define a file properly, we need to consider the various operations that can be performed on it. The operating system provides system calls to create, write, read, reposition, delete, and truncate files.

An operating system has six basic file operations: creating, writing, reading, repositioning, deleting, and truncating files.

- **Creating a file:** There are two steps necessary for creating a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** To write a file, you make a system call to specify the file's name and the information to be written to the file.

- **Reading a file:** To read from a file, you use a system call that specifies the file's name and where the next file block should be placed within memory.
- **Repositioning inside a file:** The directory is then searched for a suitable entry, and the 'current-file-position' pointer is relocated to a given value. Relocating within a file need not require any actual I/O. This file operation is also termed as 'file seek.'
- **Deleting a file:** To delete a file, you have to search the directory for the specific file. Deleting that file or directory releases all file space so other files can reuse that space.
- **Truncating a file:** The user may wish to erase the contents of a file but keep the attributes same. Rather than deleting the file and then recreating it, this utility allows all attributes to remain unchanged — except the file length — and lets the user add or edit the file content.

6. File Access Methods:

File access methods define how files are accessed and read by programs. Common methods include:

- **Sequential Access**: Reading files sequentially from the beginning to end.
- **Random Access**: Accessing files directly at any point without reading sequentially.

7. File Compression: File compression is the process of reducing the size of a file by encoding information in a more efficient way. Compressed files take up less storage space and can be transmitted faster over networks.

8. File sharing:

Operating systems often provide mechanisms for sharing files among multiple users or processes, enabling collaboration and concurrent access to files. File access permissions and locking mechanisms are used to control and coordinate shared access.

9. File backup and recovery:

Backing up files is crucial to prevent data loss in case of hardware failures, software issues, or accidental deletion. File recovery mechanisms are also essential for restoring lost or corrupted files.

10. Files Attributes and Their Operations:

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Java	Truncate
protection	class	Delete
		Close

11. File Types – Name, Extension & Function:

File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing an ASCII or binary file.

File Access Methods:

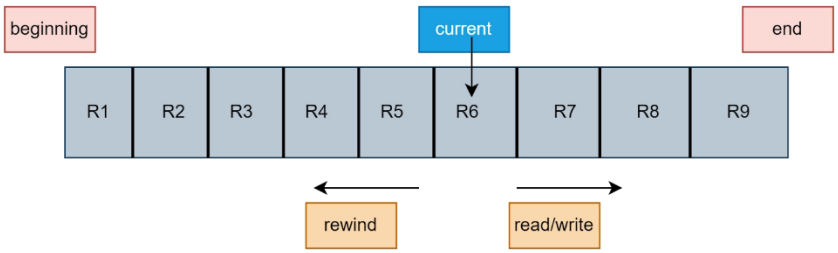
File access methods refer to the different ways in which files can be accessed and manipulated within an operating system's file system. These methods define the rules and mechanisms for reading, writing, and managing file data.

Types of File Access Methods:

1. Sequential Access Method:

It is one of the simplest access methods and is widely used in editors and compilers. You must have seen the audio cassettes. Even they use the same access method.

The files are a collection of records. Accessing the file is equivalent to accessing the records. In the sequential access method, each record is accessed sequentially, one after the other. Consider the below image for more clarity.



The figure represents a file. The current pointer is pointing to the record currently being accessed. In the sequential access method, the current pointer cannot directly jump to any record. It has to "cross" every record that comes in its path. Suppose there are nine records in the file from R1 to R9. The current pointer is at record R6. If we want to access record R8, we have to first access record R6 and record R7. This is one of the major disadvantages of the sequential access method.

The sequential access method has three operations:

- **Read next:** It will read the next record in the file. The file pointer (current pointer) will advance to the next record. It is similar to how we traverse the nodes in a linked list.
- **Write next:** This operation is used when some more information is to be included in the file. A new node (a record) will be added at the end of the file. The end pointer will now point to the node that has been added, marking it as the new end of the file. It is similar to adding a new node at the end of a linked list.
- **Rewind:** This will bring the read and write pointers to the beginning of the file.

Advantages of Sequential Access:

- The sequential access mechanism is very easy to implement.
- It uses lexicographic order to enable quick access to the next entry.

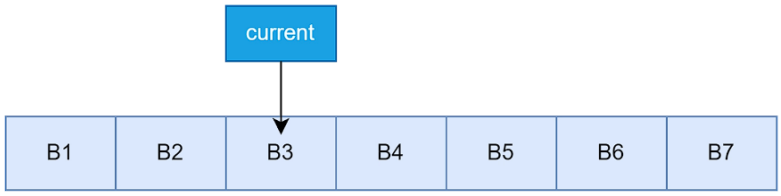
Disadvantages of Sequential Access:

- Sequential access will become slow if the next file record to be retrieved is not present next to the currently pointed record.
- Adding a new record may need relocating a significant number of records of the file.

2. Direct (or Relative) Access Method:

In the direct access method, the files are considered as a sequence of blocks or records, just like the disk was considered to be divided into equal-sized blocks. The benefit of this method is that we can access any block randomly. The direct access method is known as the relative access method. The exact block address is known only to the operating system. When a user wants to access a particular block, he/she provides the relative block number, which the operating system then uses to find the exact block address.

This type of access method is used in database management systems.



The direct access method has the following operations:

- **Read n:** This operation is used to read the nth block. Read 6 would allow us to read block B6.
- **Write n:** This operation is used to write in the nth block.
- **Goto n:** This operation is used to directly access the nth block.

Advantages of Direct/Relative Access:

- The files can be retrieved right away with a direct access mechanism, reducing the average access time of a file.
- There is no need to traverse all of the blocks that come before the required block to access the record.

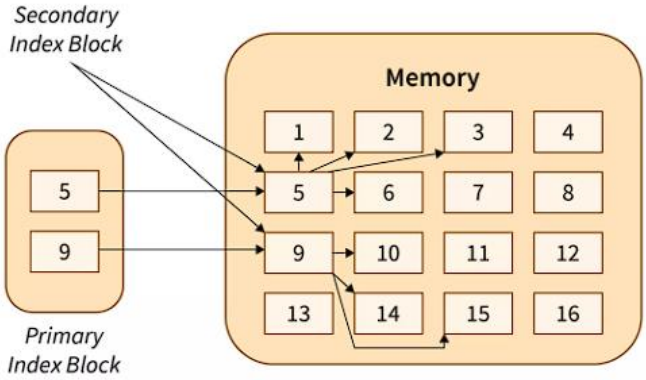
Disadvantages of Direct/Relative Access:

- The direct access mechanism is typically difficult to implement due to its complexity.
- Organizations can face security issues as a result of direct access as the users may access/modify the sensitive information. As a result, additional security processes must be put in place.

3. Index Sequential Access/ Index Access:

It's the other approach to accessing a file that's constructed on top of the sequential access mechanism. This method is practically similar to the pointer-to-pointer concept in which we store the address of a pointer variable containing the address of some other variable/record in another pointer variable. The indexes, similar to a book's index (pointers), contain a link to various blocks present in the memory. To locate a record in the file, we first search the indexes and then use the pointer-to-pointer concept to navigate to the required file.

Primary index blocks contain the links of the secondary inner blocks which contain links to the data in the memory.



Indexed Sequential Access Of File

Advantages of Indexed Sequential Access:

- If the index table is appropriately arranged, it accesses the records very quickly.
- Records can be added at any position in the file quickly.

Disadvantages of Indexed Sequential Access:

- When compared to other file access methods, it is costly and less efficient.
- It needs additional storage space.

4. Relative Record Access:

Relative Record Access is a method to access records in a file based on their relative position rather than their absolute position. Each record is assigned a unique relative record number (RRN) and must have the same fixed size. You can directly access a specific record by calculating its position using its RRN and the fixed record size. It is efficient for random or non-sequential access to records.

Advantages of Relative Record Access:

- Direct access to records based on RRNs, saving time and processing power.
- RRA is simple to implement and understand and suitable for small-scale applications.
- It is efficient for random or non-sequential access to records.
- It doesn't require complex data structures or indexing.

Disadvantages of Relative Record Access:

- It is not ideal for files with frequent updates or variable-length records.
- Records changes may disrupt other records' relative positions, which leads to data inconsistencies.
- It is limited to fixed-size records, which may waste space for smaller records.
- It may not be suitable for complex databases with dynamic data changes.

5. Content Addressable Access:

Content-Addressable Access (CAA) is used in computer systems to retrieve data based on its content rather than its identifier or location. In Content Addressable Access, data is stored with a unique content-based address, such as a hash value derived from the data itself. When you want to retrieve specific data, you provide its content, and the system uses the content-based address to locate and return the matching data.

Advantages of Content-Addressable Access:

- Data retrieval based on content, rather than traditional search methods, is quick and efficient.
- It reduces the need for maintaining complex data structures and indices.
- It suits applications like caching, data deduplication, and database systems.
- It prevents duplicate data entries, enhancing storage efficiency.

Disadvantages of Content-Addressable Access:

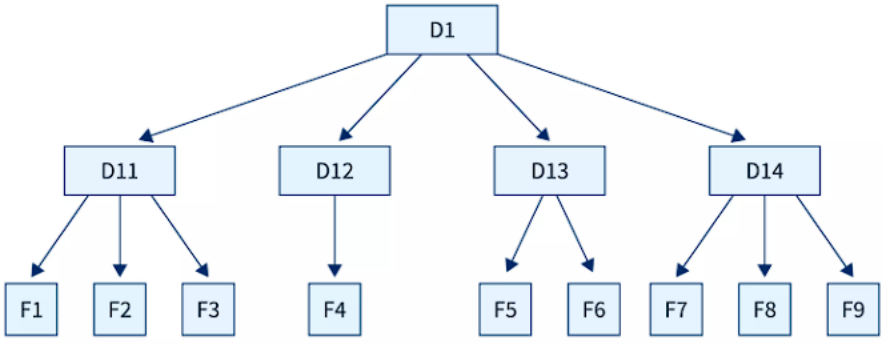
- It handles large datasets with content-based addressing can be computationally expensive.
- The risk of collisions in hash functions may cause data retrieval errors.
- Applications needing ordered data retrieval may not find content-based addressing suitable.
- Selecting an appropriate hash function prevents performance issues and ensures data integrity

Directory Structure:

A directory is a list of files on a disc. The directory structure, as well as the files, are stored on the disc. A few or all of the file attributes mentioned above may be stored in the directory. A directory can be thought of as a file that has the metadata for a bunch of files. A directory should be able to handle a variety of common operations.

The various types of operations on the directories are:

1. Search for a file
2. Create a file
3. Delete a file
4. Rename a file
5. Listing of files
6. File system traversal



Points to Consider while Maintaining a Directory Structure in an Operating System:

- Users should be able to choose file names without restrictions, promoting flexibility.
- Collaboration is enhanced by allowing users to share and access files created by others.

The Directory Structure:

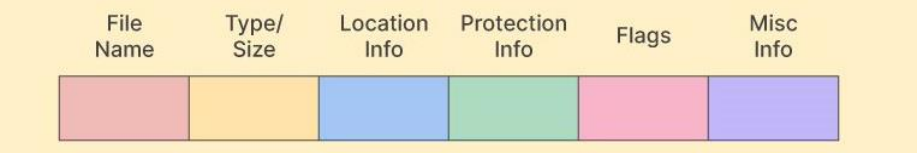
When a user or a process requests a file, the file system searches the directory for the file's entry, and when a match is found, it acquires the file's location. The field File Name provides the name of the concerned file in the directory, the Type field shows the file's type or category, and the Location Info field gives the file's location.

The Flag field provides details about the nature of the directory entry. For instance, a value of D signifies that the entry pertains to a directory, L indicates a link, and M denotes a mounted file system.

The Protection Info field determines the file's accessibility to other users within the system.

The directory's Misc info file contains miscellaneous information such as the file's owner, the date it was created, and the last time it was edited.

There are many ways to organize a common directory structure, with different levels of complexity, flexibility, and efficiency.



Types of Directory Structure:

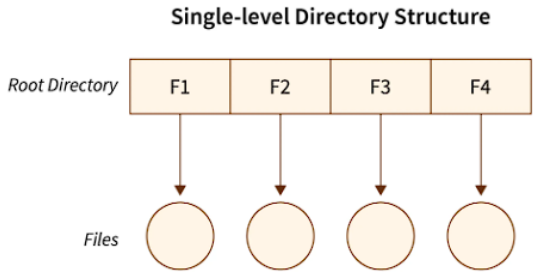
We have mainly five different types of directory structures in OS. Given below are all the five different directory structures and their sets of advantages and disadvantages along with pictorial representation to dive deeper:

1. Single-Level Directory Structure:

The single-level directory structure is the simplest and easiest directory structure out of all the other directories. In this directory structure, all the folders/files are contained under the same directory which is called the root directory. The single-level directory structure gathers all the files under one directory or the root directory, this makes it easy to support and understand.

Now as the different files are under the same-root directory the users are not allowed to create the different sub-directories serving their requirements. This also creates a barrier with the single-level directory as when the number of files increases or more than one user logs into the system both of these need to maintain the standards of giving a unique name to it. This also means that if two users call their files 'apple', then this, in turn, will violate the unique name standardization.

Below is the pictorial representation of The Single-level directory structure in OS:



Advantages:

1. Because it is just a single directory, therefore it is the simplest directory structure.
2. Searching will be faster if the files are smaller in size. Also, the groping capability of files increases.
3. In such a user file directory structure, actions like file creation, searching, file deletion, and updating are quite simple.

Disadvantages:

1. In a single-level directory structure, if the directory is vast, searching will be difficult.
2. We can't group the same type of files in a single-level directory structure.
3. Selecting a unique file name is a little more difficult than in other types.

2. Two-Level Directory Structure:

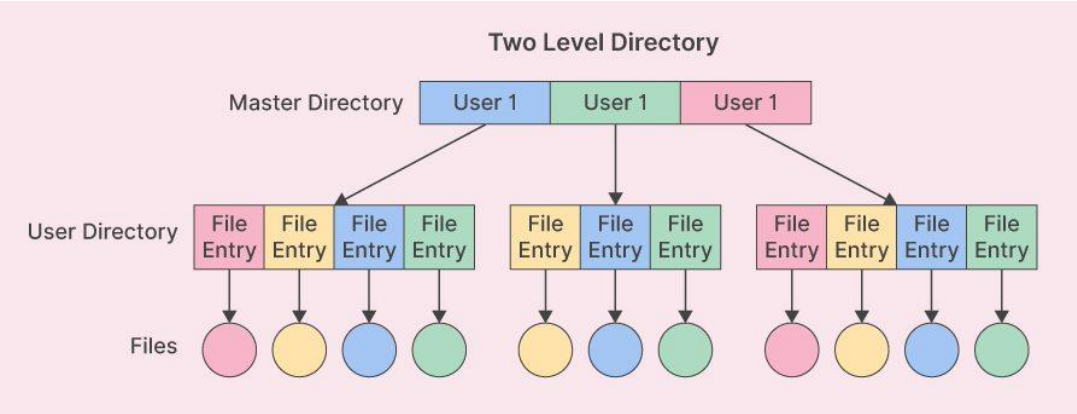
Overcoming the drawbacks posed by the single-level directory structure, i.e., the confusion created by the same file names given by several users - The Two-level directory structure in OS came into the picture.

The two-level directory structure in OS offers a unique solution to the problem caused by single-level that is, this directory structure gives each user the right to have their own user files directory commonly called **User File Directory (UFD)**. The User File Directory or UFD has a similar structure as that of the single level, but each UFD lists only the files of a single user who owns that UFD. To root all the UFDs, the system's **Master File Directory or (MFD)** searches whenever a new user id's logged into the directory structure.

This can also be defined as the two-level directory structure in OS which gives each of its users the right to create a directory directly inside the root directory. Here the directories created by the user are called the UFDs and to check who logged in as a user the Master File Directory or MFDs are responsible for the same.

Here the MFDs are indexed by username or account number which are pointed to the UFDs with each entry point of the user. In a two-level directory structure searching files becomes, even more, easier as there is only one user's list, which is required to be traversed along with having a pathname for each file such as /User-name/directory-name/ which is also defined here.

Below is the pictorial representation of The Two-level directory structure in OS:



Advantages:

1. In a two-level directory, a full path like user-name/directory-name can be given.
2. Different users can have the same directory as well as the file name.
3. Searching files becomes easier.

Disadvantages:

1. One user cannot share a file with another user in a two-level directory without permission.
2. The two-level directory also has the problem of not being scalable.

3. Tree-Structured Directory Structure:

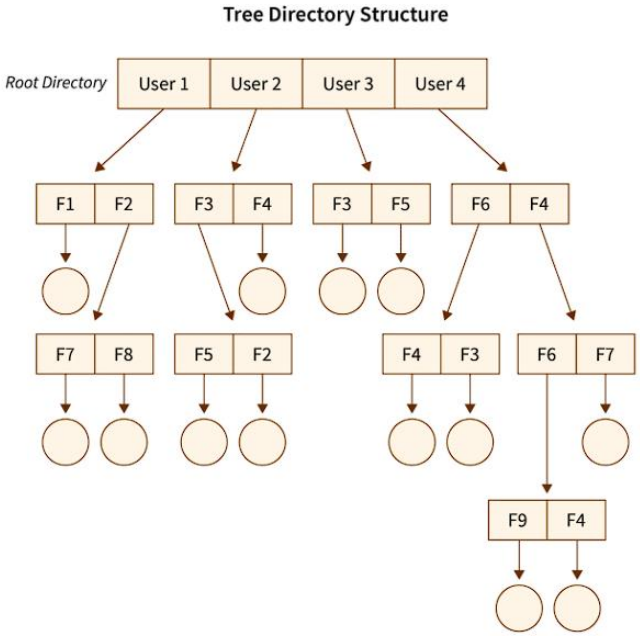
As observed in the two-level directory structure in OS the drawback of users not having the ability to create sub-directories is resolved with The Tree-structured directory structure in OS coming into the picture.

The Tree-Structured directory structure in OS is said to be the most common directory structure among users as it gives the users the capability to create sub-directories under their defined directory. Here we have the natural generalization which extends the directory structure to a tree of arbitrary height whereas, in the case of two-level, it was the tree of two heights. This generalization in tree structure allows the user the ability to create their own subdirectories and organize their files accordingly.

The tree-structured directory structure has separate parent directories for the sub-directories owned by each of their specific users and the parent directories of the users are all under the master-root directory which makes it a tree structure. This helps in total separation between the users which provides complete naming freedom and privacy to users' information.

The system administrator/ UFD admin only has full access to the root directory. In the Tree-structured directory structure searching is quite effective where we use the current working concept that is we can access the file by using two kinds of paths that are either absolute or relative.

Below is the pictorial representation of The Tree-structured directory structure in OS:



Advantages:

1. The directory, which is organized in a tree form, is extremely scalable.
2. Collisions are less likely in the tree-structured directory.
3. Searching in the tree-structure directory is relatively simple because we may utilize both absolute and relative paths.

Disadvantages:

1. Files may be saved in numerous directories if they do not fit into the hierarchical structure.
2. We are unable to share files in a tree-structured directory.
3. Because a file might be found in several folders in a tree-structured directory, it is inefficient.

4. Acyclic-Graph Directory Structure:

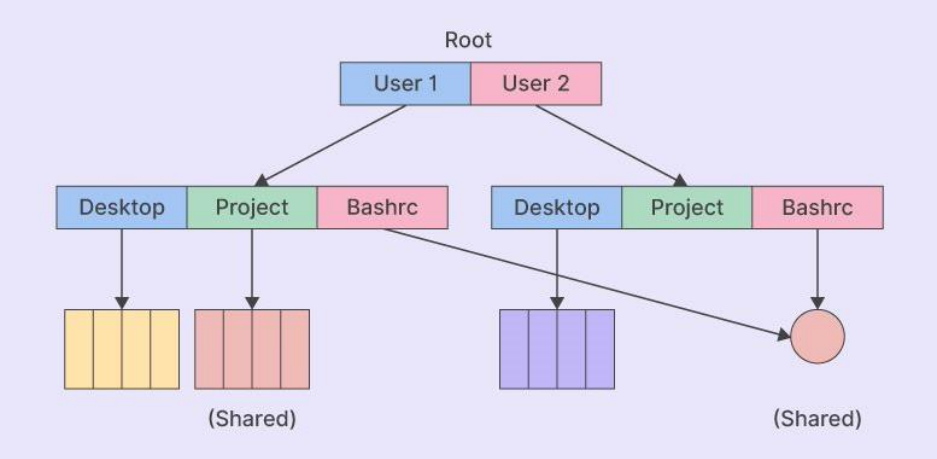
The tree model forbids the existence of the same file in several directories. By making the directory an acyclic graph structure, we may achieve this. Two or more directory entries can lead to the same subdirectory or file, but we'll limit it, for now, to prevent any directory entries from pointing back up the directory structure.

Links or aliases can be used to create this type of directory graph. We have numerous names for the same file, as well as multiple paths to it. There are two types of links:

1. symbolic link or soft link (specify a file path: logical) and
2. hard link (actual link to the same file on the disc from multiple directories: physical).

If we delete the file, there may be more references to it. The file is simply erased via symbolic links, leaving a dangling pointer. A reference count is kept through hard links, and the actual file is only erased once all references to it have been removed.

Below is the pictorial representation of The Acyclic graph directory structure in OS:



Advantages:

1. Acyclic-Graph directory structure provides the ability to share files.
2. Due to different-different paths, searching is simple in the Acyclic-Graph directory structure.

Disadvantages:

1. If the files are linked together, removing them may be difficult.
2. If we use softlink and if the file is removed, all that is left is a dangling pointer.
3. If we use a hardlink when we delete a file, we must also erase all of the references that are associated with it.

5. General-Graph Directory Structure:

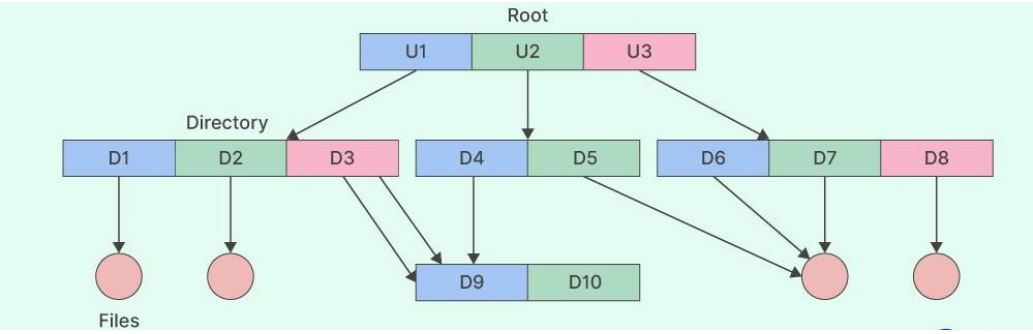
As observed in the acyclic-graph directory structure in OS the drawback of links that are established and need to be either terminated or suspended to reach the files/directory is resolved with The General Graph directory structure in OS taking a vital place in the different types of a directory structure in OS.

In the General Graph directory structure in OS users have the capability to create a cycle of the directory within a directory where we have the power to derive the various directories with the help of more than one parent directory in operating systems. In this type of structure, the users are free to create directories under the root directory along with creating sub-directories under the same structure which can also hold true if the users want to create multiple sub-directories that is, the users are free to create different sub-directories for different file types.

Adding to the above, With the help of the file paths if the users feel the need to access the location of the files, then that is made possible by the General Graph Directory Structure. In this structure, the file paths or paths can be categorized into two broad categories to locate the files in the directory structure as below:

1. **The Absolute Path:** Here, the path of the desired files can be determined by considering the root directory as the base directory.
2. **The Relative Path:** Here, the path of the desired files can be determined by two choices that are, either the file that needs to be retrieved from its directory is considered the base directory, or the user's directory is considered as the base directory.

Below is the pictorial representation of The General Graph directory structure in OS:



Advantages:

1. The general-graph directory structure is more adaptable than the others.
2. In the general-graph directory, cycles are permitted.

Disadvantages:

1. It is more expensive than other options.
2. Garbage collection is required in the general-graph directory.

File System Mounting:

Mounting is a process in which the operating system adds the directories and files from a storage device to the user's computer file system. The file system is attached to an empty directory, by adding so the system user can access the data that is available inside the storage device through the system file manager. Storage systems can be internal hard disks, external hard disks, USB flash drivers, SSD cards, memory cards, network-attached storage devices, CDs and DVDs, remote file systems, or anything else.

Terminologies used in File System Mounting:

- **File System:** It is the method used by the operating system to manage data storage in a storage device. So, a user can access and organize the directories and files in an efficient manner.
- **Device name:** It is a name/identifier given to a storage partition. In windows, for example, "D:" in windows.
- **Mount point**: It is an empty directory in which we are adding the file system during the process of mounting.

Procedure for mounting file system:

To mount the file system, device name and location within the file structure

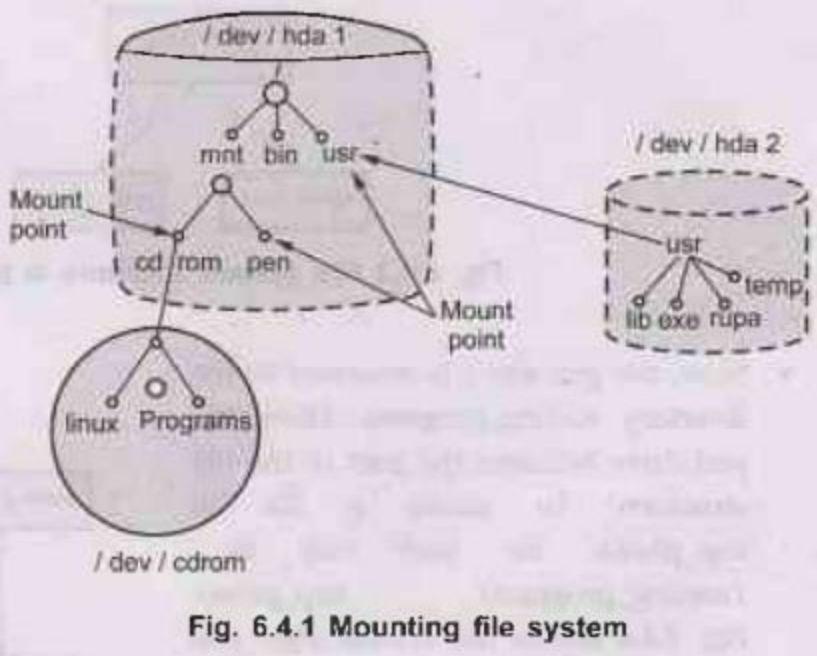


Fig. 6.4.1 Mounting file system

Fig. 6.4.1 shows the mounting a file system on secondary storage device. File system manage mounted directories by using mount table. User can create soft links to files in mounted file system. Mount tables keep track of the actual physical location of the files.

When a file system is mounted, the client can reference files by the local path name. There is no reference to remote host location, although files remain physically located at the remote site.

Busy file system is cannot unmount by using unmount command. A file system is considered busy in following situation:

1. File system is shared in between the users.

2. A program has a file open in that file system.

3. A user is accessing a file/directory in the file system.

Example: Consider a file structure of the pen drive. Following is Fig. 6.4.2 of file

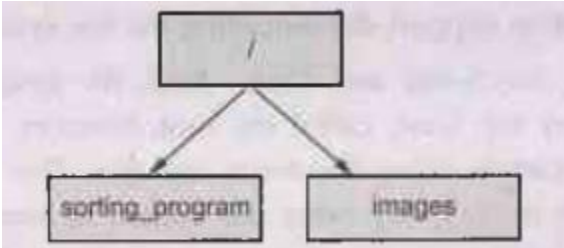


Fig. 6.4.2 File system before mounting

The file system structure in the pen drive is as follows:

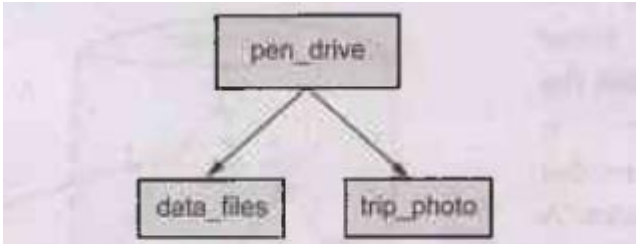


Fig. 6.4.3 File system structure in pen drive

Now, the pen drive is mounted to the directory sorting_program. Here, the pen drive becomes the part of the file

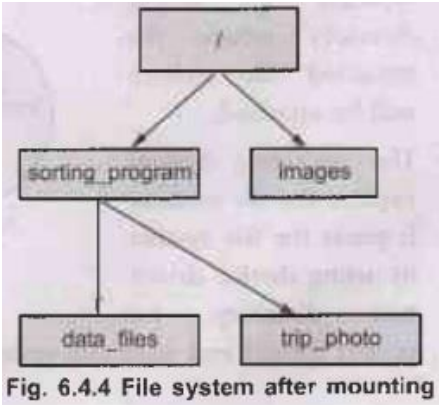


Fig. 6.4.4 File system after mounting

Advantages of mounting:

1. It is a user-oriented naming scheme

2. It makes sense to be able to mount at any level.

Disadvantages of mounting:

1. The hardware is less obvious since devices can be located anywhere in the tree structure

Question: ***Discuss the advantages and disadvantages of supporting links to files that cross mount points.***

Solution:

Advantages:

1. Greater transparency

2. User does not need to be aware of mount points.

Disadvantages:

1. The error condition would expose to the user that a link is a dead link and that the link does indeed cross file system boundaries.

2. The file system containing the link might be mounted while the file system containing the target file might not be.

Directory Implementation:

Directory implementation refers to the methods and data structures used by an operating system to organize and manage the directory structure of its file system.

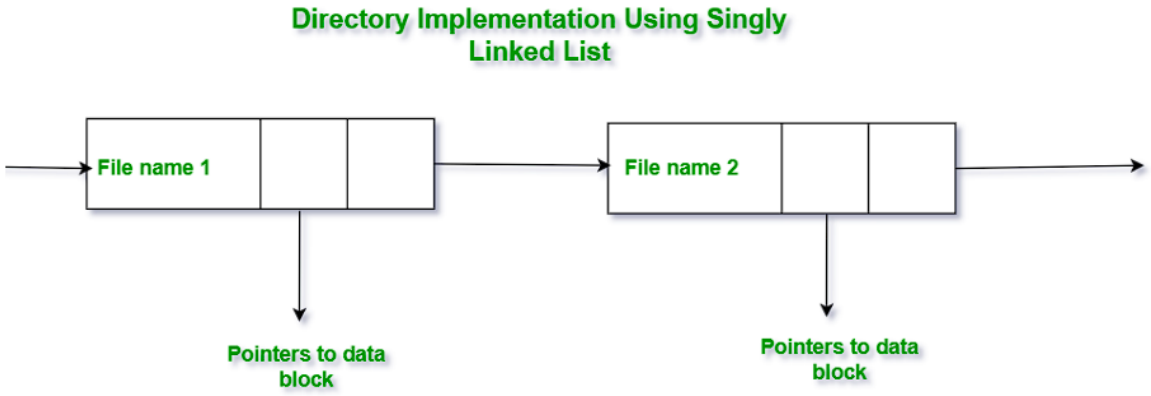
Directory implementation in the operating system can be done using Singly Linked List and Hash table.

The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms.

There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

1. Directory Implementation using Singly Linked List:

The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. Here we implement a directory by using a linear list of filenames with pointers to the data blocks.



- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.
- In order to delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it.
- To reuse the directory entry we can mark that entry as unused or we can append it to the list of free directories.
- To delete a file linked list is the best choice as it takes less time.

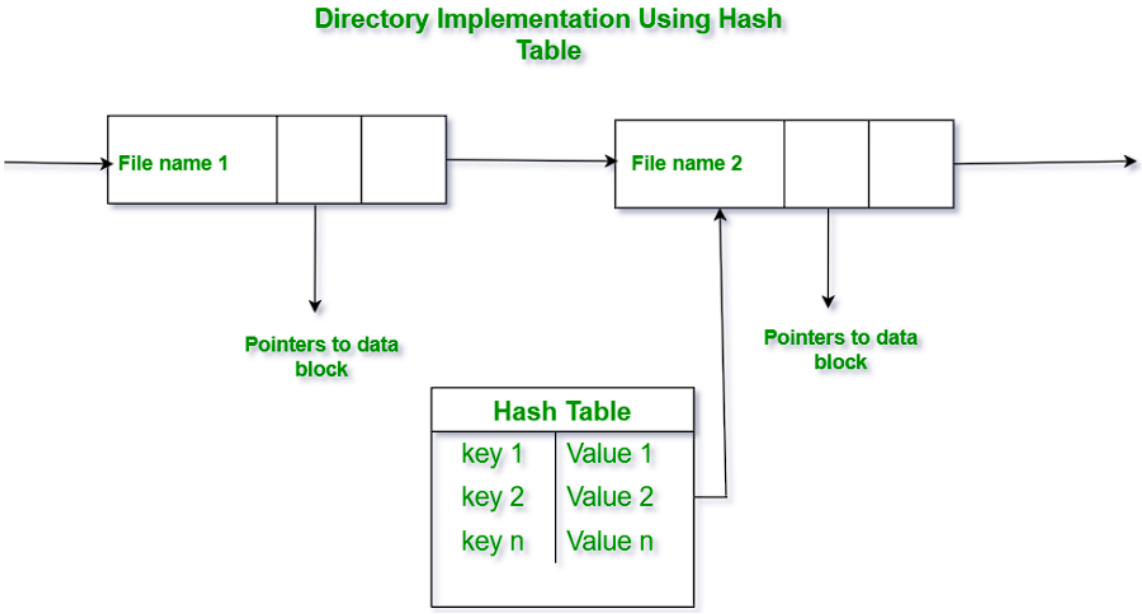
Disadvantage:

The main disadvantage of using a linked list is that when the user needs to find a file the user has to do a linear search. In today's world directory information is used quite frequently and linked list implementation results in slow access to a file. So the operating system maintains a cache to store the most recently used directory information.

2. Directory Implementation using Hash Table:

An alternative data structure that can be used for directory implementation is a hash table. It overcomes the major drawbacks of directory implementation using a linked list. In this method, we use a hash table along with the linked list. Here the linked list stores the directory entries, but a hash data structure is used in combination with the linked list.

In the hash table for each pair in the directory key-value pair is generated. The hash function on the file name determines the key and this key points to the corresponding file stored in the directory. This method efficiently decreases the directory search time as the entire list will not be searched on every operation. Using the keys the hash table entries are checked and when the file is found it is fetched.



Disadvantage:

The major drawback of using the hash table is that generally, it has a fixed size and its dependency on size. But this method is usually faster than linear search through an entire directory using a linked list.

File Allocation Methods:

Whenever a hard disk is formatted, a system has many small areas called blocks or sectors that are used to store any kind of file. File allocation methods are different ways by which the operating system stores information in memory blocks, thus allowing the hard drive to be utilized effectively and the file to be accessed.

Below are the types of file allocation methods in the Operating System –

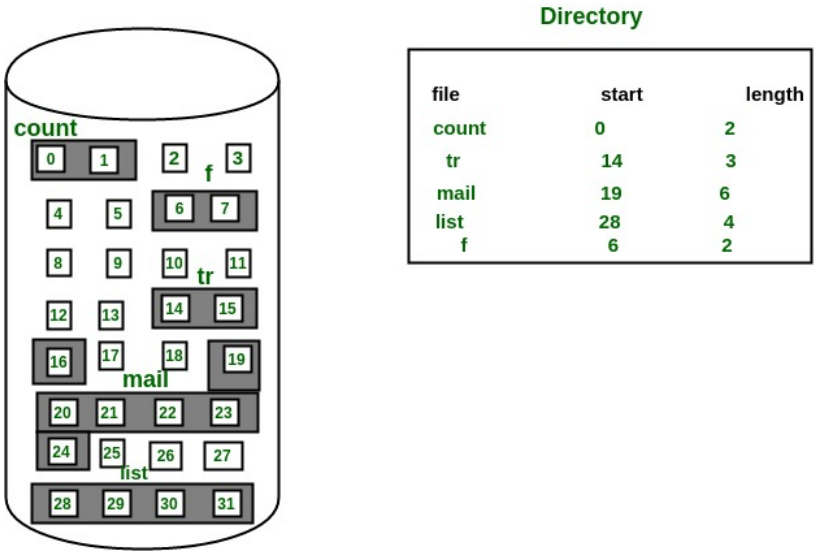
1. Contiguous Allocation:

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

The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



Advantages:

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

Disadvantages:

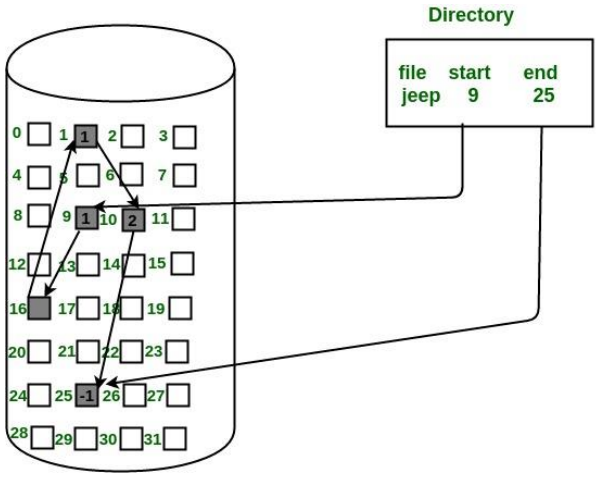
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation:

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

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.

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.



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.

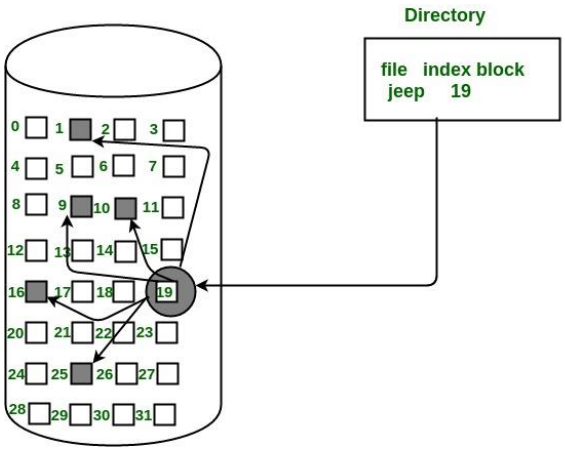
Disadvantages:

- 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 cannot 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.

3. Indexed Allocation:

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

The directory entry contains the address of the index block as shown in the image:



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.
- **For files that are very large**, single index block may not be able to hold all the pointers. Following mechanisms can be used to resolve this:

  1. Linked Scheme: If the file is big then more blocks are required so one index block is insufficient to store all the pointers, therefore to store the pointers two or more index blocks are used where these index boxes are connected using linked file allocation that is each index block stores the pointer to the next index block.

  2. Multilevel Index: In this method, the multiple indexes blocks along with the levels of these blocks. Here, the level 1 block is used for pointing to the level 2 block which points to the blocks occupied by the file. These index blocks can be extended to three or more levels according to the size of the file.

  3. Combined Scheme: In Combined Scheme, a special block is used to store all the information related to the file like name, authority, size, etc. The special block is called **inode** (information-node). Some space of this special block is used to store the information related to the field as mentioned above and the remaining space is used to store the addresses of blocks that contain the actual file. *The **inode** is explained further in detail.*

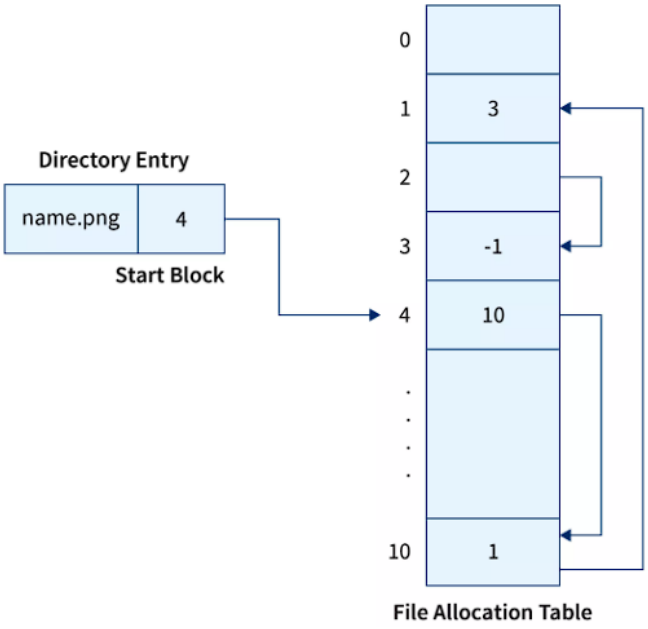
File Allocation Table (FAT):

The File Allocation Table (FAT) overcomes the drawback of Linked File allocation. The random access of a particular block is not possible in the linked file allocation. To access a particular block, it is necessary to access all its previous blocks.

In the file allocation table, all disk block links are collected and maintained. Here the number of head seeks is reduced by caching the file allocation table so that the head does not need to go through all the disk blocks to access one particular block.

The whole process of randomly accessing any block using FAT is completed by reading the desired entry of a block from the file allocation table and accessing that particular block.

The diagrammatic representation of FAT is given below -



Advantages:

- Random Access to the block is possible in FAT.
- One bad/corrupted disk block cannot corrupt all the other blocks.
- It uses all the disk blocks for data as in linked file allocation it needs extra space for pointers.

Disadvantages:

- If entries increase so the FAT size also increases.
- Each entry requires the FAT entry.
- If Entries increase the FAT size increases which increases the size of a block so there are chances of internal fragmentation.

Inode:

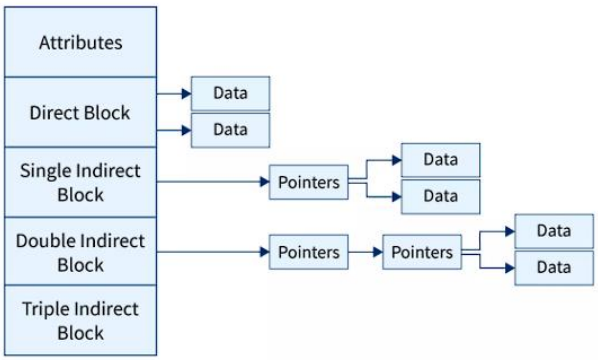
In Operating systems based on UNIX, every file is indexed using **Inode (information-node)**. The inode is the special disk block that is created with the creation of the file system. This special block is used to store all the information related to the file like name, authority, size, etc along with the pointers to the other blocks where the file is stored.

In this special block, some of its space is used to store the information related to the field as mentioned above and the remaining space is used to store the addresses of blocks that contain the actual file.

The first few pointers in Inode point to **direct blocks**, i.e they contain the addresses of the disk blocks containing the file data. A few pointers in inode point to the **indirect blocks**. There are three types of indirect blocks: *single indirect*, *double indirect*, and *triple indirect*.

A **single indirect block** contains nothing but the address of the block containing the file data, not the file itself as shown in the figure below. Furthermore, the **double indirect block** points to the pointers which again point to the pointers which point to the data blocks. Further, it goes in a similar way for **triple indirect block**.

The diagrammatic representation of Inode is given below -



Advantages:

- Accessibility of file becomes easy as all the information like metadata and block address is stored inside inode.
- Read-write and creation timestamps are stored inside the inode.
- Filenames do not affect inodes. In other words, a single file can be copied and renamed without losing its address.

Disadvantages:

- All new files and folders will be rejected as soon as a file system runs out of inodes.
- Upon 100% utilization of the inodes system will start to notice OS restarting, data loss, applications crashing, and more OS-related issues.

Free Space Management:

There is a system software in an operating system that manipulates and keeps a track of free spaces to allocate and de-allocate memory blocks to files, this system is called a **file management system** in an operating system". There is a **free space list** in an operating system that maintains the record of free blocks.

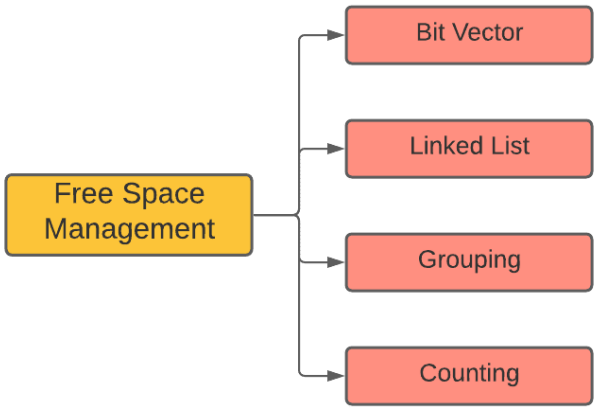
When a file is created, the operating system searches the free space list for the required space allocated to save a file. While deletion a file, the **file system** frees the given space and adds this to the **free space list**.

Commonly Used Free Space Management Techniques:

- **Linked Allocation:** In this technique, files are stored in a linked list manner in disk blocks. Each block contains the address to the next block.
- **Contiguous Allocation:** In this technique, files are stored in a contiguous block in disk space.
- **Indexed Allocation:** In this technique, files are stored in a linked list of disk blocks where the directory entry for the file also contains an index block. The index block contains the addresses of the first few blocks in a linked list.
- **File Allocation Table (FAT):** In this technique, tables are used to track the allocation of disk space. Each entry indicates whether the block is free or allocated, and if allocated, then it indicates to which block it belongs.
- **Volume Shadow Copy:** It is a feature of some operating systems that creates a snapshot of a volume at a specific point in time.

Methods of Free Space Management:

It is not easy work for an operating system to allocate and de-allocate memory blocks (managing free space) simultaneously. The operating system uses various methods for adding free space and freeing up space after deleting a file. There are various methods using which a free space list can be implemented.

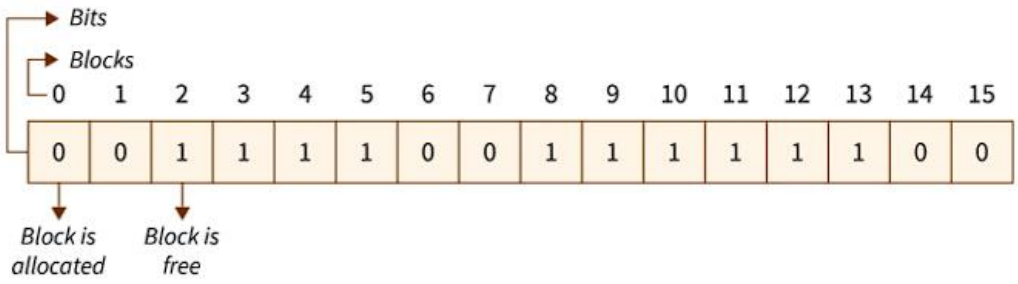


1. Bitmap or Bit Vector:

A bit vector is a most frequently used method to implement the free space list. A bit vector is also known as a **Bit map**. It is a series or collection of bits in which each bit represents a disk block. The values taken by the bits are either **1** or **0**. If the block bit is 1, it means the block is empty and if the block bit is 0, it means the block is not free. It is allocated to some files. Since all the blocks are empty initially so, each bit in the bit vector represents 0.

Let us take an example:

Given below is a diagrammatic representation of a disk in which there are 16 blocks. There are some free and some occupied blocks present. The upper part is showing block number. Free blocks are represented by **1** and occupied blocks are represented by **0**.



"Free block number" can be defined as that block which does not contain any value, *i.e.*, they are free blocks. The formula to find a free block number is:

Block number = (Number of bits per word)* (number of 0-value words) + (offset of first bit)

We will consider the first **8** bits group (00111100011110) to constitute a non-zero word since all bits are not **0** here. **Non-zero word** is that word that contains the bit value **1** (block that is not free). Here, the first non-zero word is the third block of the group. So, the offset will be **3**.

Hence, the **block number** =8∗0+3=3=8∗0+3=3

Advantages of Bit vector method:

- Simple and easy to understand.
- Consumes less memory.
- It is efficient to find free space.

Disadvantages of the Bit vector method:

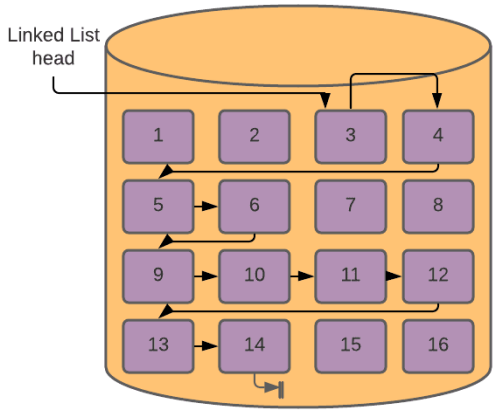
- The operating system goes through all the blocks until it finds a free block. (block whose bit is '0').
- It is not efficient when the disk size is large.

2. Linked List:

A **linked list** is another approach for free space management in an operating system. In it, all the free blocks inside a disk are linked together in a **linked list**. These free blocks on the disk are linked together by a pointer. These pointers of the free block contain the address of the next free block and the last pointer of the list points to null which indicates the end of the linked list. This technique is not enough to traverse the list because we have to read each disk block one by one which requires I/O time.

For example, consider a disk having 16 blocks where block numbers 3, 4, 5, 6, 9, 10, 11, 12, 13, and 14 are free, and the rest of the blocks, i.e., block numbers 1, 2, 7, 8, 15 and 16 are allocated to some files. If we maintain a linked list, then Block 3 will contain a pointer to Block 4, and Block 4 will contain a pointer to Block 5.

Similarly, Block 5 will point to Block 6, Block 6 will point to Block 9, Block 9 will point to Block 10, Block 10 will point to Block 11, Block 11 will point to Block 12, Block 12 will point to Block 13 and Block 13 will point to Block 14. Block 14 will point to null. The address of the first free block, i.e., Block 3, will be stored somewhere in the memory. This is also represented in the following figure-



Advantages:

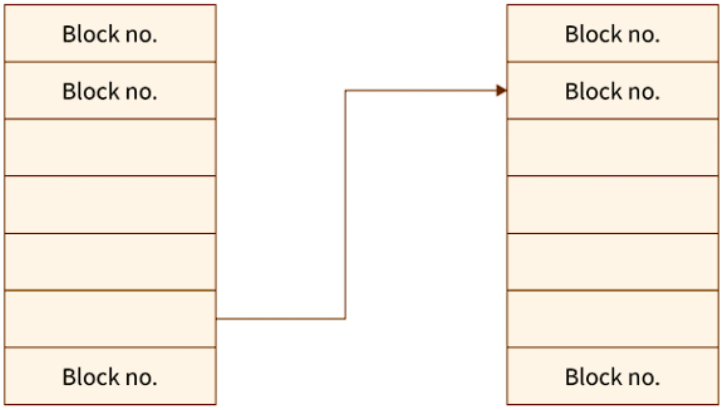
- In this method, available space is used efficiently.
- As there is no size limit on a linked list, a new free space can be added easily.
- External fragmentation is prevented by linked list allocation. As opposed to contiguous allocation, this prevents the wasting of memory blocks.

Disadvantages

- In this method, the overhead of maintaining the pointer appears.
- The Linked list is not efficient when we need to reach every block of memory.
- There is no provision for random or direct memory access in linked list allocation. We must search through the full linked list to locate the correct block if we wish to access a certain file block.

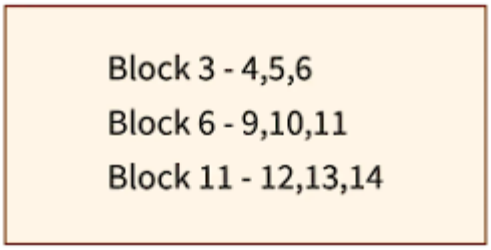
3. Grouping:

The grouping technique is also called the **"modification of a linked list technique"**. In this method, first, the free block of memory contains the addresses of the **n-free** blocks. And the last free block of these **n** free blocks contains the addresses of the next **n** free block of memory and this keeps going on. This technique separates the empty and occupied blocks of space of memory.



Example

Suppose we have a disk with some free blocks and some occupied blocks. The free block numbers are 3,4,5,6,9,10,11,12,13,3,4,5,6,9,10,11,12,13 and 14. And occupied block numbers are 1,2,7,8,15,1,2,7,8,15, and 16 *i.e.*, they are allocated to some files.



When the **"grouping technique"** is applied, block **3** will store the addresses of blocks **4**, **5**, and **6** because **block 3** is the first free block. In the same way, block **6** will store the addresses of blocks **9**, one **0**, and one **1** because block **6** is the first occupied block.

Advantage of the Grouping method

1. By using this method, we can easily find addresses of a large number of free blocks easily and quickly.
2. This method has the benefit of making it simple to locate the addresses of a collection of empty disk blocks.

Disadvantage

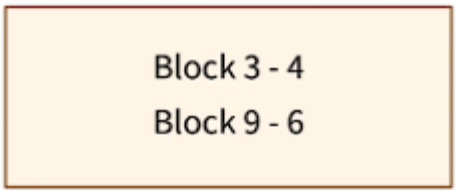
1. We need to change the entire list if one block gets occupied.

4. Counting:

In memory space, several files are created and deleted at the same time. For which memory blocks are allocated and de-allocated for the files. Creation of files occupy free blocks and deletion of file frees blocks. When there is an entry in the free space, it consists of two parameters- **"address of first free disk block (a pointer)"** and **"a number 'n'"**.

Example:

Let us take an example where a disk has 16 blocks in which some blocks are empty and some blocks are occupied as given below:



When the **"counting technique"** is applied, the block number **3** will represent block number **4** because block **3** is the first free block. Then, the block stores the number of free blocks *i.e.* - there are **4** free blocks together. In the same way, the first occupied block number **9** will represent block number **10** and keeps the number of rest occupied blocks i.e.- there are **6** occupied blocks as shown in the above figure.

Advantages

- In this method, a bunch of free blocks takes place fastly.
- The list is smaller in size.

Disadvantage

- In the counting method, the first free block stores the rest free blocks, so it requires more space.

Advantages of Free Space Management Techniques:

- **Efficient use of storage space:** space management techniques help to optimize the use of storage space on the hard disk as well as secondary storage devices.
- **Easy to implement:** some techniques in space management are easy to implement, like linked allocation. It requires less overhead in terms of processing and memory resources.
- **Reduce Fragmentation:** Free space management techniques like contiguous allocation help in reducing disk fragmentation which results in faster access to files.

Disadvantages of Free Space Management Techniques:

- **Complex:** some space management techniques are complex to implement and manage.
- **Performance overhead:** Free space management techniques can contribute to adding some performance overhead because some techniques require frequent updates to the free space list.
- **Vulnerability to corruption:** If the free space list is not properly maintained, there is a chance of corruption.

Efficiency and Performance:

- **Efficiency** depends on:
  - Disk allocation and directory algorithms
  - Type of data kept in file's directory entry
- **Performance** improvement techniques:
  - **Disk cache**
    - Separate section of main memory (in kernel space) for frequently used disk blocks
  - **Virtual disk** (RAM disk)
    - Dedicate a section of main memory as virtual file-system
  - **Free-behind** and **read-ahead** techniques
    - Optimization for sequential access

Backup and Recovery:

- **System failure** (e.g. sudden power outage) may result in
  - Loss of data
  - Inconsistency of data
- **File system recovery** techniques
  - **Consistency checker**
    - Compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
    - Examples: `fsck` in Unix, `chkdsk` in Windows
  - **Back up**
    - Use system programs to *regularily* back up data from disk to another storage device (e.g. magnetic tape or other disk)
    - Recover lost file or disk by **restoring** data from backup

Log Structured File Systems:

A Log-Structured File System (LFS) is a type of file system that manages storage by writing all modifications to a log-like structure. This approach is different from traditional file systems, which update data in place. LFS offers several advantages, including improved performance, reduced fragmentation, and better recovery after crashes. Here's an overview of how Log-Structured File Systems work and their key features:

How Log-Structured File Systems Work:

1. **Log-Based Approach**:
   - LFS organizes data as a log, where all changes to files are written sequentially to the log.
   - Unlike traditional file systems that update data in place, LFS never modifies existing blocks. Instead, it writes new versions of modified blocks to the log.
2. **Segmented Log**:
   - The log is divided into segments of fixed size.
   - When a segment fills up, it is written to disk as a whole unit. This write is done sequentially, which is faster than random writes to different parts of the disk.
3. **Garbage Collection**:
   - As segments become full, LFS performs garbage collection by consolidating valid data from multiple segments into a new, empty segment.
   - This process also includes removing invalid or deleted data.
4. **Metadata Handling**:
   - Metadata (like file system structures such as inodes) is also written to the log.
   - Metadata updates are batched and written to the log, improving efficiency.
5. **Checkpointing**:
   - Periodically, the file system takes a snapshot of its metadata and writes it to stable storage.
   - This allows for quick recovery to a consistent state after a crash. If the system crashes, it can replay the log from the last checkpoint to restore the file system's state.

The following are the data structures used in the **LFS implementation** -

- **Inodes:** As in Unix, inodes contain physical block pointers to files.
- **Inode Map:** This table indicates the location of each inode on the disk. The inode map is written in the segment itself.
- **Segment Summary:** This maintains information about each block in the segment.
- **Segment Usage Table:** This tells us the amount of data on a block.

Advantages of Log-Structured File Systems:

1. **Improved Write Performance**: LFS can write data sequentially, which is faster than random writes. This sequential writing reduces seek times, especially on traditional hard drives.
2. **Reduced Fragmentation**: Since LFS writes data in a log, it avoids fragmentation that can occur in traditional file systems when files are modified and written in non-contiguous blocks. Reading data sequentially from the log helps to reduce seek times and improve read performance.
3. **Better Recovery After Crashes**: Checkpointing and the ability to replay the log allow for quick recovery to a consistent state. The file system can replay the log to bring the file system back to a consistent state after a crash, reducing downtime and the need for lengthy consistency checks.
4. **Efficient Garbage Collection**: LFS can efficiently reclaim space by consolidating valid data and removing invalid or deleted data during garbage collection.
5. **Write-Once, Read-Many (WORM) Support**: LFS is well-suited for write-once, read-many scenarios such as logging and archival systems.

Disadvantages and Considerations:

1. **Overhead**: LFS can have overhead in terms of garbage collection and metadata handling, especially for small writes.
2. **Wear-Leveling for SSDs**: For Solid State Drives (SSDs), LFS needs to consider wear-leveling to ensure even usage of flash memory cells.
3. **Potential for Fragmentation**: While LFS reduces fragmentation within the file system, the log itself can become fragmented over time.