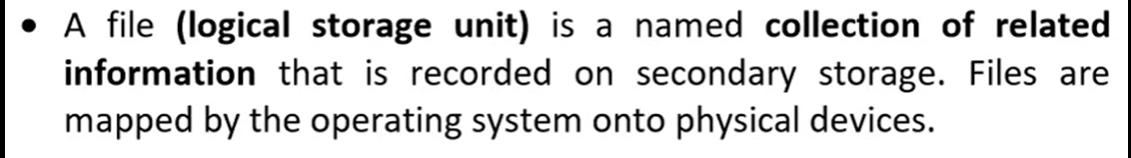
**File Systems in Operating System**

A file can be defined as a data structure which stores the sequence of records.

A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format, although there can be various types of files that help us to store the data.

**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.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From the user’s perspective, a file is the smallest allotment of logical secondary storage.

**Files Attributes And Their Operations**

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

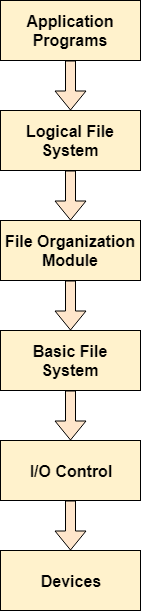
| File type | Usual extension | Function |
| --- | --- | --- |
| Executable | exe, com, bin | Read to run machine language program |
| Object | obj | Compiled, machine language not linked |
| Source Code | C, java, pas, asm, py | Source code in various languages |
| Batch | bat, sh | Commands to the 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 System Structure

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



* When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the **Meta data of the file and directory structure**. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
* Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files**, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.**
* Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
* I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

**File Access Methods in OS**

When we access the data inside the file, it needs to be read into the memory before further execution is done. There are specific ways in which the information inside a file handling can be accessed.

**There are different types of file access methods in OS mentioned below.**

**1 Sequences Method**

This is the most common file access methods in OS. The details in the file are processed in order, one record after the other. All the data must be loaded sequentially for random access inside the given file. Usually, **it contains read, write, and rewind file operations.**

The read operation (read next) reads the data stored on the next position of the pointer and moves the information forward by one read point.

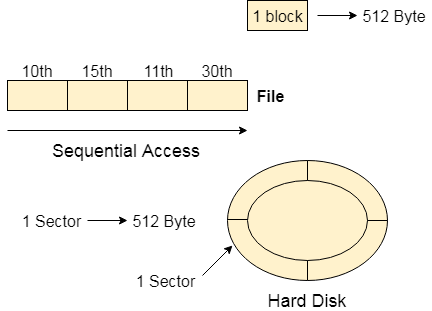
The write operation (write next) adds the data at the end of the file and moves the pointer forward to the newly **registered data**.

The rewind operation moves the pointer backward until the file’s required information appears.

An example of this file access method can be accessing data from a tape.

**Analysis of the sequential access method**

* It takes effort to implement. The work is similar to a linked list.
* Since the records cannot be randomly accessed, it could be more efficient.
* It is a slow method.



**Advantages of the Sequential Method :**

* It is simple to appliance this file access mechanism.
* It uses lexicographic order to access the next entry quickly.
* Simple and easy to implement, requiring minimal hardware and software support.
* Low cost, as it doesn’t require complex indexing or search algorithms.

**Disadvantages of the Sequential Method :**

* If the file data that requires to be accessed following is not present next to the ongoing document, this type of file access method is slow.
* Moving a sizable file block may be necessary to insert a new record.
* More efficient for random access: Sequential access could be more efficient for randomly accessing data within a file, as it requires searching the entire file to find the desired data.
* Limited concurrency: Concurrent access to a sequential file can be challenging, as only one process can access the file simultaneously.

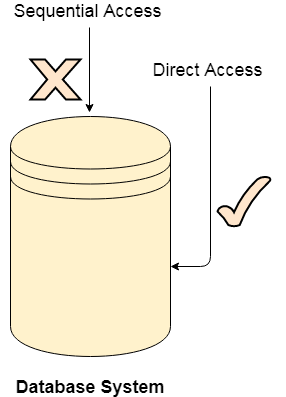
**2 Direct**

The direct method in file access methods in OS is also known as the relative access method. A file is composed of fixed-length logical records called blocks. These blocks can be accessed in any order. The file is accessed as a numbered sequence of blocks or logs. Therefore, we may read block 10, then block 40, and then write block 5.

There are no constraints on the order of reading or writing for a direct-access file. **Direct access files** in file access methods in OS are of great use for intermediate access to large amounts of information, and **most databases are based on this type of file access.**

**Analysis of the direct access method**

* Faster than the sequential access method in file access methods in os.
* Allows random access. Therefore there is no need to traverse all the blocks.
* Implementation is easy.



**Advantages of Direct/Relative Access:**

* The files can be retrieved right away with direct access mechanism, reducing the average access time of a file.
* There is no need to traverse all the blocks before the required block to access the record.
* Suitable for real-time systems that require fast and predictable data access.
* Flexible data retrieval, ideal for accessing data in a non-sequential fashion.

**Disadvantages of Direct/Relative Access:**

* The direct access mechanism is typically challenging to implement due to its complexity.
* Organizations can face security issues due to direct access, as the users may access/modify **sensitive information**. As a result, additional security processes must be put in place.
* The direct access methods in file access methods in OS take lots of time.

**3 Indexed Method**

A direct-access method in file access methods in os is the foundation for this file access technique. An index method in file access methods in os is created for the entire file, which contains pointers to the various blocks inside the disk.

To find data in the file, we first traverse through the index and then use the information to access the file directly. File Access Methods in OS, However, in this file access, extra memory is required for keeping the index file, which burdens the memory resources.

**Analysis of the indexed method**

* It is a modification of the sequential access method in file access methods in os. It allows random access.
* **Apart from the file records, a different index is required to keep track of the blocks.**
* If file size increases, the index may be unable to hold all the pointers due to memory constraints. Hence in such a case multi-level index may be used.

**Advantages of Indexed Sequential Access:**

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

**Disadvantages of Indexed Sequential Access:**

* Compared to other file access methods, it is costly and less efficient.
* It wants additional storage space.

**FILE Directory Structure in OS**

A directory is a container that stores files and folders, organizing them hierarchically. The Directory Structure in OS manages entries of files, including file names, locations, protection info, and more. This structure enables efficient file retrieval.

**Operations in Directory Structure in OS**

* **Creating**: Users can create new files and directories, providing unique names for directories.
* **Searching**: Users can search for specific files or directories within a directory.
* **Deleting**: Unwanted files or empty directories can be deleted.
* **Listing**: Users can retrieve a list of files in a directory.
* **Renaming**: Files and directories can be renamed to reflect content changes.
* **Linking**: Files can be linked to appear in multiple directories.
* **Unlinking:** Removing links from files in multiple directories.

**The 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 :

A diagram of a structure

Description automatically generated

**The Advantages**

* The implementation of a single-level directory structure is simple and easier as compared to other directory structures in OS.
* If the file size is smaller, then the searching of such files with the single-level directory structure becomes simpler.
* The single-level directory structure allows the operations such as searching, creation, deletion and updating as well.

**The Disadvantages**

* As several users can log in at the same system to log their files maintaining a unique name becomes difficult leading to a collision. This also means that if the file with the same name is created then the old file will get destroyed first, then the new file (having the same name ) created will replace it.
* If the size of the files is bigger then searching the files in one root directory of the single-level directory structure will become time-consuming and hence difficult.
* The single-level directory structure restricts the grouping of the same type of files together.

**The 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 :

A diagram of a structure

Description automatically generated

**The Advantages:**

* In the two-level directory structure in OS different users have the right to have the same directory as well as a file name as the user has its own UFD which can give a filename that can match other users but won't cause an issue.
* We can also see that **searching for files become much simpler.**
* As we have a user-defined directory this also provides privacy related to files stored as no user can enter the other user’s directory without permission.
* In a two-level directory structure we cannot group the files which are having the same name into a single directory for a specific user.

**The Disadvantages:**

* In a two-level directory structure a user is not allowed to share files with other users.
* We also find that **scalability** is not present in a two-level directory structure as two files of the same type cannot be grouped together in the same user.
* Here users cannot create subdirectories only one user file directory can be defined under one master file directory.

**The 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 :

sA diagram of a tree directory structure

Description automatically generated

**The Advantages:**

* 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.
* Here we can group the same type of files into one directory.
* In this directory structure the chances of collision of names/types etc are less and hence we can say that the directory structure in OS is scalable.

**The Disadvantages:**

* In the tree-structure directory structure in OS the files cannot be shared between users. Also, the users cannot modify/update the root directory of other users.
* This directory structure in OS as we have to go under multiple directories to access a file we can say that it is said to be inefficient.
* Here each file does not fit into the **hierarchal model** and so we have to save the files into multiple directories.

**The Acyclic Graph Directory Structure**

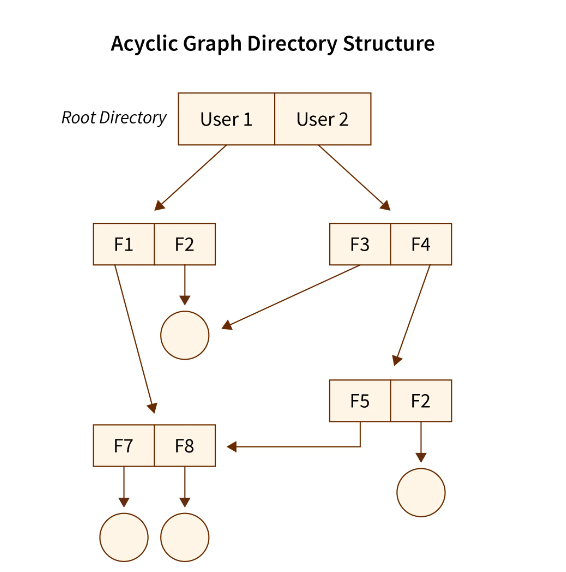
Overcoming the drawbacks posed by the tree-structured directory structure,i.e, the restriction that it cannot have multiple parent directories and also cannot share files between users - The Acyclic Graph directory structure in OS came into the picture.

In the Acyclic Graph directory structure in OS can be defined as the directory structure that allows a directory or a file to have multiple parent directories so that it can be a shared file in a directory that gets pointed by the other user directories which if has the access to that shared file via the links provided to it. It is often said to be a natural generalization of the tree-structured directory.

Mostly, this is used in situations such as, when two users or two programmers are collaborating on a project and they need to access the files. So we have the associated files which are stored in a subdirectory mostly separated from other projects and files of other programmers/users. Now as they are working on a joint project they want the sub-directories to be present in their own directories. Therefore these common sub-directories where two or more users can collaborate would be shared so that the files can be stored in their individual locations and this is where we use Acyclic directories.

There must be a point noted here that the shared file is not the exact copy file, that is if any programmer/user makes some changes in the sub-directory that change will be reflected in both **subdirectories**. Here we can (with the help of aliases or links) create the acyclic type of directory graph where we can also provide different paths to the same file. We define the Links into two kinds, popularly known as The Hard Link and The Symbolic Link.

a. **The Hard Link:** This is also called as the physical link. If we want to delete the files in the acyclic graph directory structures then we need to remove the actual files only if all the references to the files are deleted that is, no link that even references the main file should be established. Here we don’t leave a suspended link.

b. **The Symbolic/Soft-Link:** This is also called as the logical link. If we want to delete the files in the acyclic graph directory structures then we need to simply delete the files and need to keep in mind that only a dangling/ hanging point is left. Here we leave a suspended link.Below is the pictorial representation of The Acyclic graph directory structure in OS : 

**The Advantages:**

* In the **Acyclic Graph** directory structure in OS we can share files between users.
* Here we can search the files easily as compared to the tree-structured directory structure as here we have different-different paths to one file.

**The Disadvantages:**

* In the Acyclic Graph directory structure in OS as we can share the files via linking, so there are chances that in the case when we want to delete a file in a directory it may create a problem. a. Also, even if the link is a **soft link** then after deleting the file we are left with a dangling/suspended point of the link. b. But in the case of **hardlink**, when we delete a file we have to vanish all the references associated with it, which can lead to issues associated with referring back to files in case a requirement arises.

**File Allocation Methods in OS**

**Overview**

The File allocation methods in OS are different ways that are used for storing the file on the hard disk. There are 5 different ways in which we can store the files on the hard disk in such a manner that there is efficient utilization of disk space and the file can be accessed faster by the Operating System.

**What is File Allocation in OS?**

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.

**Types of File Allocation Methods in Operating System.**

* Contiguous File allocation
* Linked File Allocation
* Indexed File Allocation

Let's have an in-detail explanation about each of them,

**Contiguous File Allocation.**

First, let's understand the meaning of contiguous, here contiguous means adjacent or touching. Now let's understand what is contiguous file allocation.

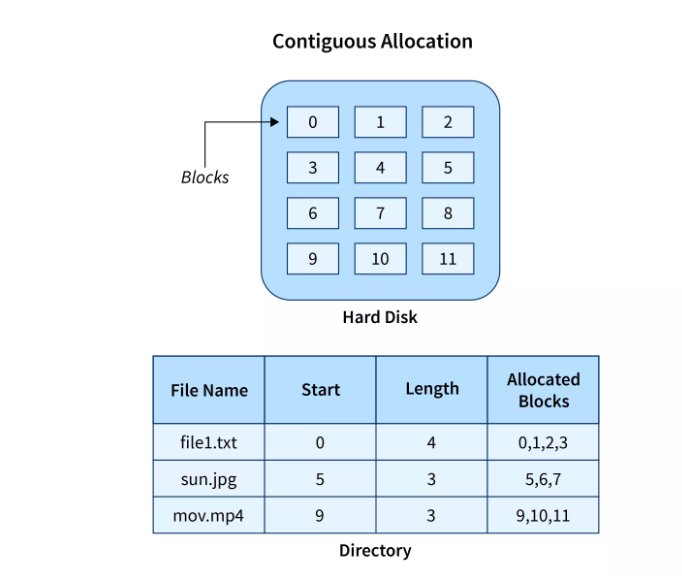
**What is Contiguous File allocation?**

In contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are adjacent.

Assuming a file needs 'n' number of blocks in the disk and the file begins with a block at position 'x', the next blocks to be assigned to it will be x+1,x+2,x+3,...,x+n-1 so that they are in a contiguous manner.

Let's understand this diagrammatically.

**Example**

****

We have three different types of files that are stored in a contiguous manner on the hard disk.

In the above image on the left side, we have a memory diagram where we can see the blocks of memory. At first, we have a text file named file1.txt which is allocated using contiguous memory allocation, it starts with the memory block 0 and has a length of 4 so it takes the 4 contiguous blocks 0,1,2,3. Similarly, we have an image file and video file named sun.jpg and mov.mp4 respectively, which you can see in the directory that they are stored in the contiguous blocks. 5,6,7 and 9,10,11 respectively.

Here the directory has the entry of each file where it stores the address of the starting block and the required space in terms of the block of memory.

**Advantages and Disadvantages**

**Advantages**

* It is very easy to implement.
* There is a minimum amount of seek time.
* The disk head movement is minimum.
* Memory access is faster.
* It supports sequential as well as direct access.

**Disadvantages**

* At the time of creation, the file size must be initialized.
* As it is pre-initialized, the size cannot increase. As
* Due to its constrained allocation, it is possible that the disk would fragment internally or externally.

**Linked File Allocation.**

**What is Linked File Allocation?**

The Linked file allocation overcomes the drawback of contiguous file allocation. Here the file which we store on the hard disk is stored in a scattered manner according to the space available on the hard disk. Now, you must be thinking about how the OS remembers that all the scattered blocks belong to the same file. So as the name linked File Allocation suggests, the pointers are used to point to the next block of the same file, therefore along with the entry of each file each block also stores the pointer to the next block.

Let's understand this better diagrammatically by taking an example.

**Example**

Here we have one file which is stored using Linked File Allocation.

A screenshot of a computer

Description automatically generated

In the above image on the right, we have a memory diagram where we can see memory blocks. On the left side, we have a directory where we have the information like the address of the first memory block and the last memory block.

In this allocation, the starting block given is 0 and the ending block is 15, therefore the OS searches the empty blocks between 0 and 15 and stores the files in available blocks, but along with that it also stores the pointer to the next block in the present block. Hence it requires some extra space to store that link.

**Advantages and Disadvantages**

**Advantages**

* There is no external fragmentation.
* The directory entry just needs the address of starting block.
* The memory is not needed in contiguous form, it is more flexible than contiguous file allocation.

**Disadvantages**

* It does not support random access or direct access.
* If pointers are affected so the disk blocks are also affected.
* Extra space is required for pointers in the block.

**Indexed File Allocation.**

**What is Indexed File Allocation?**

The indexed file allocation is somewhat similar to linked file allocation as indexed file allocation also uses pointers but the difference is here all the pointers are put together into one location which is called **index block**. That means we will get all the locations of blocks in one index file. The blocks and pointers were spread over the memory in the Linked Allocation method, where retrieval was accomplished by visiting each block sequentially. But here in indexed allocation, it becomes easier with the index block to retrieve.

Let's take an example to explain this better.

**Example**

As shown in the diagram below block 19 is the index block which contains all the addresses of the file named **text1**. In order, the first storage block is 9, followed by 16, 1, then 10, and 25. The negative number -1 here denotes the empty index block list as the file text1 is still too small to fill more blocks.

A diagram of a number and a number

Description automatically generated

**Advantages and Disadvantages**

**Advantages**

* It reduces the possibilities of external fragmentation.
* Rather than accessing sequentially it has direct access to the block.

**Disadvantages**

* Here more pointer overhead is there.
* If we lose the index block we cannot access the complete file.
* It becomes heavy for the small files.
* It is possible that a single index block cannot keep all the pointers for some large files.

To resolve this issue, we can use the following approaches:

1. Linked scheme
2. Multilevel Index
3. Combined Scheme

**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**

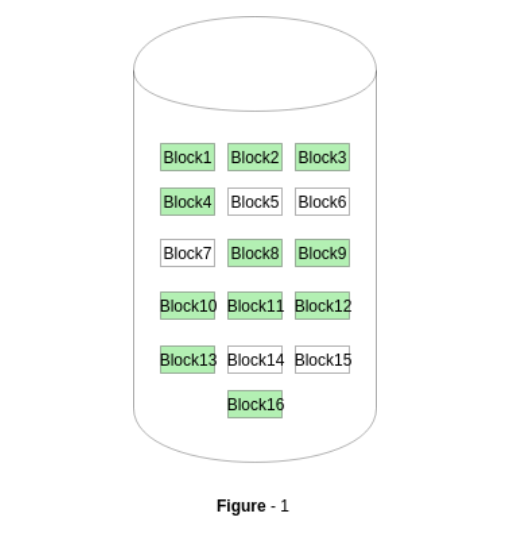
In Combined Scheme, a special block is used to store all the information related to the file like name, authority, size, etc. The special block is called **inode**(information-node). Some space of this special block is used to store the information related to the field as mentioned above and the remaining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

**Free space Disk management in Operating System**

**Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices**

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. **Bitmap or Bit vector –** A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block. The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as:**0000111000000110**.



**Advantages –**

* + Simple to understand.
  + Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

1. **Linked List –** In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

A diagram of a cylinder

Description automatically generated

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list. A drawback of this method is the I/O required for free space list traversal.

1. **Grouping –** This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.
2. **Counting –** This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain:
   * Address of first free disk block
   * A number n

| **Parameters** | **UNIX** | **Windows** |
| --- | --- | --- |
| Basic | It is a command-based operating system. | It is a menu based operating system. |
| Licensing | It is an open-source system which can be used to under General Public License. | It is a proprietary software owned by Microsoft. |
| User Interface | It has a text base interface, making it harder to grasp for newcomers. | It has a [Graphical User Interface](https://www.geeksforgeeks.org/what-is-graphical-user-interface/), making it simpler to use. |
| Processing | It supports Multiprocessing. | It supports Multithreading. |
| File System | It uses Unix File System (UFS) that comprises STD.ERR and STD.IO file systems. | It uses File Allocation System (FAT32) and New technology file system (NTFS). |
| Security | It is more secure as all changes to the system require explicit user permission. | It is less secure compared to UNIX. |
| Data Backup & Recovery | It is tedious to create a backup and recovery system in UNIX, but it is improving with the introduction of new distributions of Unix. | It has an integrated backup and recovery system that make it simpler to use. |
| Hardware | Hardware support is limited in UNIX system. Some hardware might not have drivers built for them. | Drivers are available for almost all the hardware. |
| Reliability | Unix and its distributions are well known for being very stable to run. | Although Windows has been stable in recent years,  it is still to match the stability provided by Unix systems. |
| Case Sensitive | It is fully case-sensitive, and files can be considered separate files | It has case sensitivity as an option. |