Unit-7

File System Interface Management

File concept

A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes. It is a method of data collection that is used as a medium for giving input and receiving output from that program.

In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user. Every File has a logical location where they are located for storage and retrieval.

Objective of file management

- It provides I/O support for a variety of storage device types.
- Minimizes the chances of lost or destroyed data
- Helps OS to standardized I/O interface routines for user processes.
- It provides I/O support for multiple users in a multiuser systems environment

File structure

A File Structure needs to be predefined format in such a way that an operating system understands. It has an exclusively defined structure, which is based on its type.

Three types of files structure in OS:

- A text file: It is a series of characters that is organized in lines.
- An object file: It is a series of bytes that is organized into blocks.
- A source file: It is a series of functions and processes.

File type

It refers to the ability of the operating system to differentiate various types of files like text files, binary, and source files.

The system uses the extension to indicate the type of the file and the type of operations that can be done on that file. Only a file with a .com, .exe, or .sh extension can be executed, for instance. The .com and .exe files are two forms of binary executable files, whereas the .sh file is a shell script containing, in ASCII format, commands to the operating system. Application programs also use extensions to indicate file types in which they are interested. For example, Java compilers expect source files to have a .java extension, and the Microsoft Word processor expects its files to end with a .doc or .docx extension. These extensions are not always required, so a user may specify a file without the extension (to save typing), and the application will look for a file with the given

name and the extension it expects. Because these extensions are not supported by the operating system, they can be considered "hints" to the applications that operate on them.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Fig: Common file types

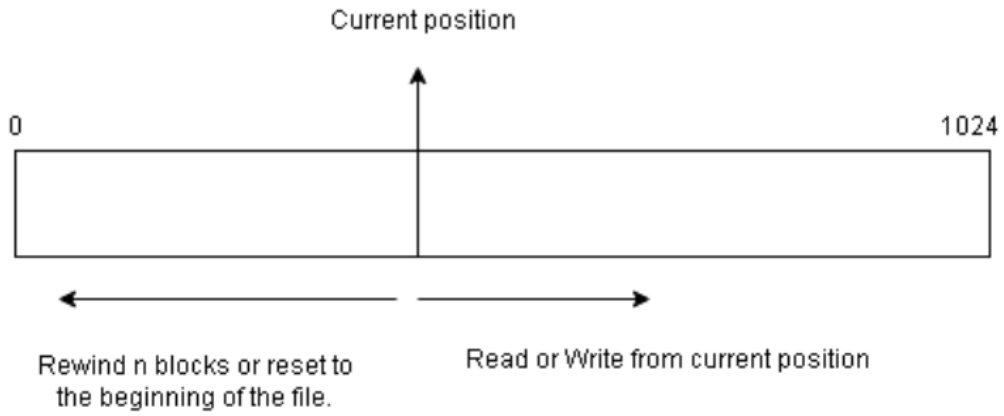
File access methods

The information stored in a file must be accessed and read into memory. Though there are many ways to access a file, some system provides only one method, other systems provide many methods, out of which you must choose the right one for the application.

1. **Sequential Access Method**

In this method, the information in the file is processed in order, one record after another. For example, compiler and various editors access files in this manner.

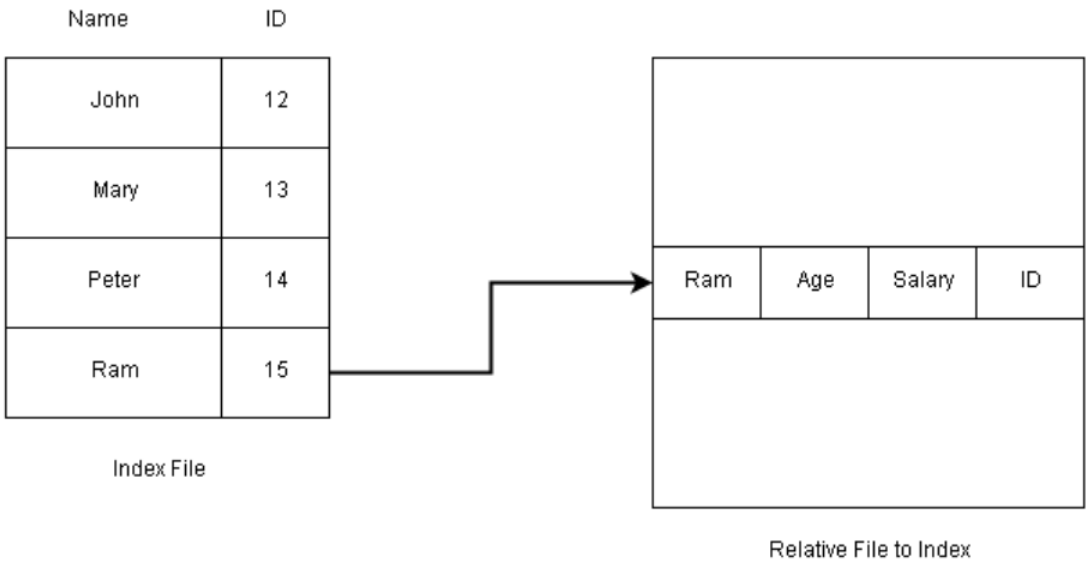
The read-next – reads the next portion of a file and updates the file pointer which tracks the I/O location. Similarly, the write-next will write at the end of a file and advances the pointer to the new end of the file.



Sequential access to File

2. Direct Access Method

The other method for file access is direct access or relative access. For direct access, the file is viewed as a numbered sequence of blocks or records. This method is based on the disk model of file. Since disks allow random access to file block.



Direct Access Method Using Index

You can read block 34, then read 45, and write in block 78, there is no restriction on the order of access to the file. The direct access method is used in database management. A query is satisfied immediately by accessing large amount of information stored in database files directly. The database maintains an index of blocks which contains the block number. This block can be accessed directly and information is retrieved.

3. Indexed sequential access method

It is the other method of accessing a file which is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

File allocation methods

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

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

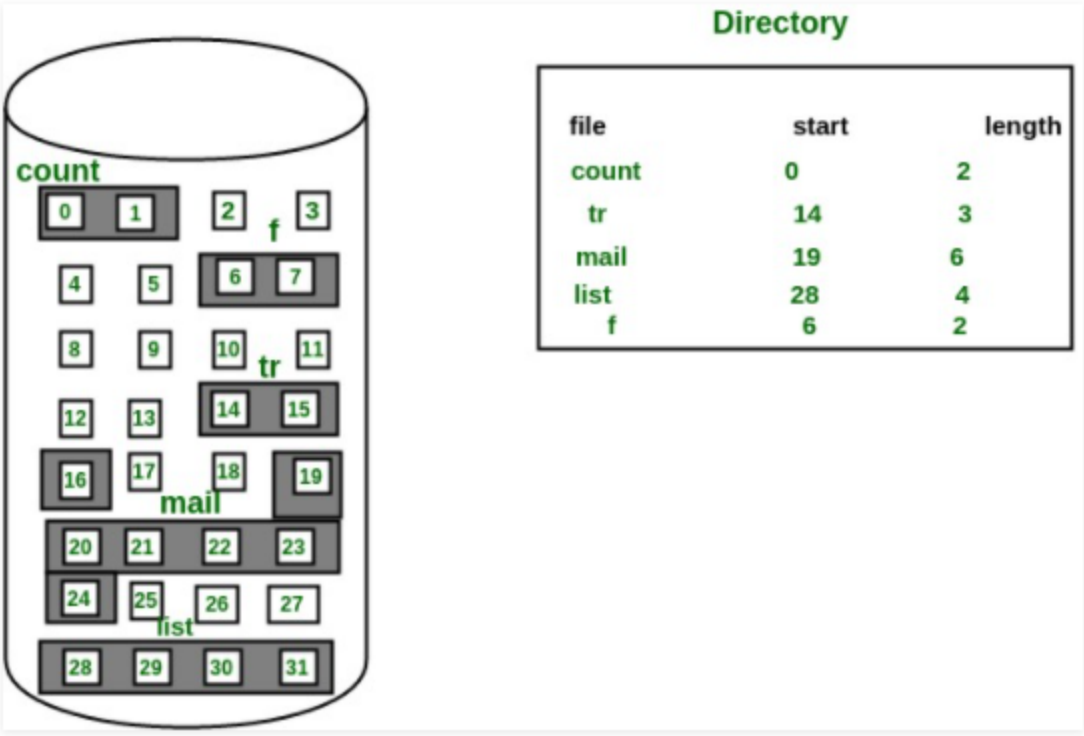
- Efficient disk space utilization.
- Fast access to the file blocks.

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:

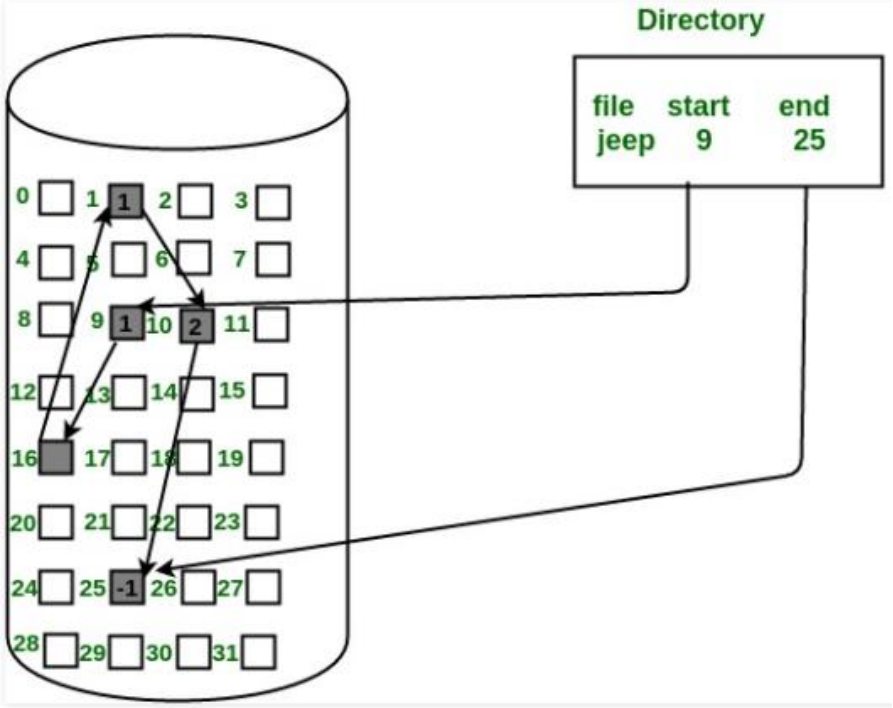
- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

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:

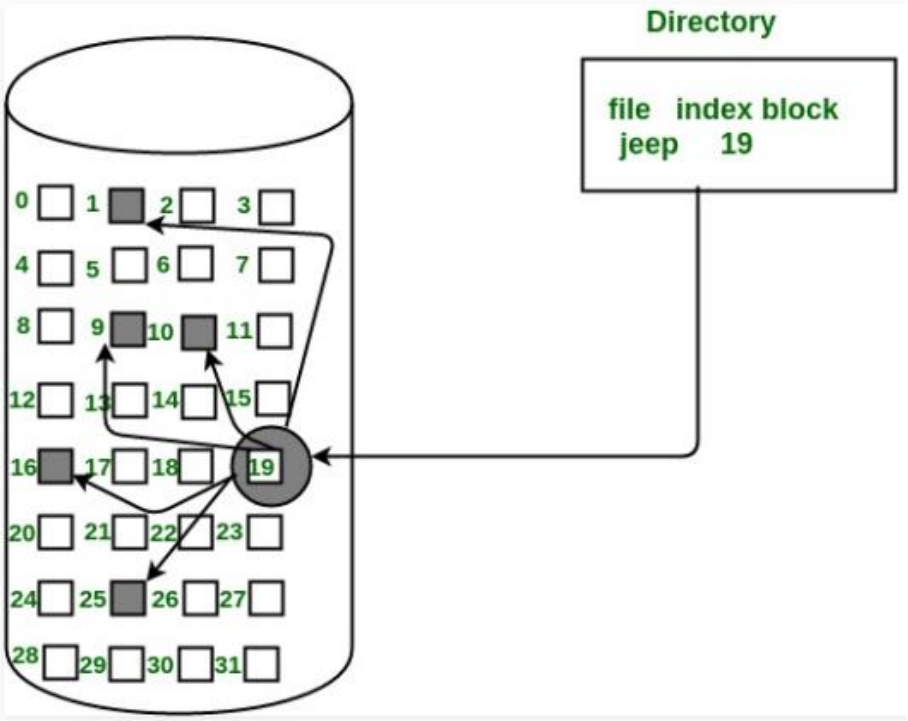
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

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 can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

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 i^{th} entry in the index block contains the disk address of the i^{th} file block. The directory entry contains the address of the index block as shown in the image:



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.

File attributes

A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters, such as example.c. Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not. When a file is named, it becomes independent of the process, the user, and even the system that created it. For instance, one user might create the file example.c, and another user might edit that file by specifying its name. The file's owner might write the file to a USB disk, send it as an e-mail attachment, or copy it across a network, and it could still be called example.c on the destination system. A file's attributes vary from one operating system to another but typically consist of these:

- Name:The symbolic file name is the only information kept in human readable form.
- Identifier: This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- Type: This information is needed for systems that support different types of files.

- Location: This information is a pointer to a device and to the location of the file on that device.
- Size: The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- Protection: Access-control information determines who can do reading, writing, executing, and so on.
- Time, date, and user identification: This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File operations

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. Let's examine what the operating system must do to perform each of these six basic file operations. It should then be easy to see how other similar operations, such as renaming a file, can be implemented.

- **Creating a file:** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file-position pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity.
- **Repositioning within a file:** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.
- **Deleting a file:** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry
- **Truncating a file:** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

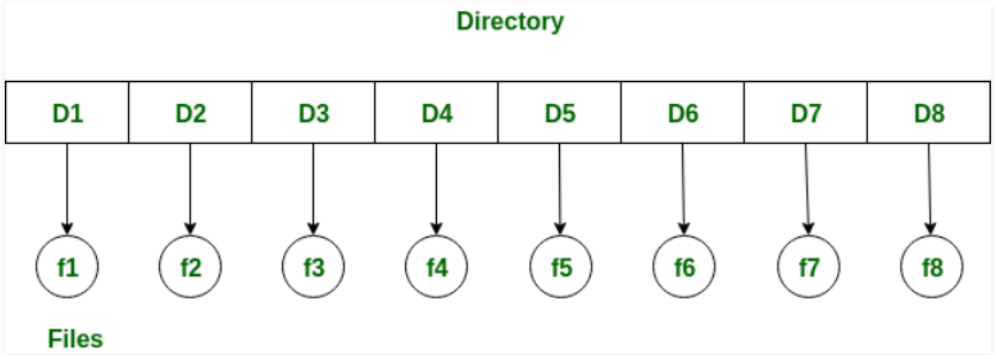
Directories

A Directory is the collection of the correlated files on the disk. In simple words, a directory is like a container which contains file and folder. In a directory, we can store the complete file attributes or some attributes of the file. A directory can be comprised of various files. With the help of the directory, we can maintain the information related to the files.

There are several logical structures of directory, these are given as below:

1. **Single-level directory**

The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.
A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.



Advantages

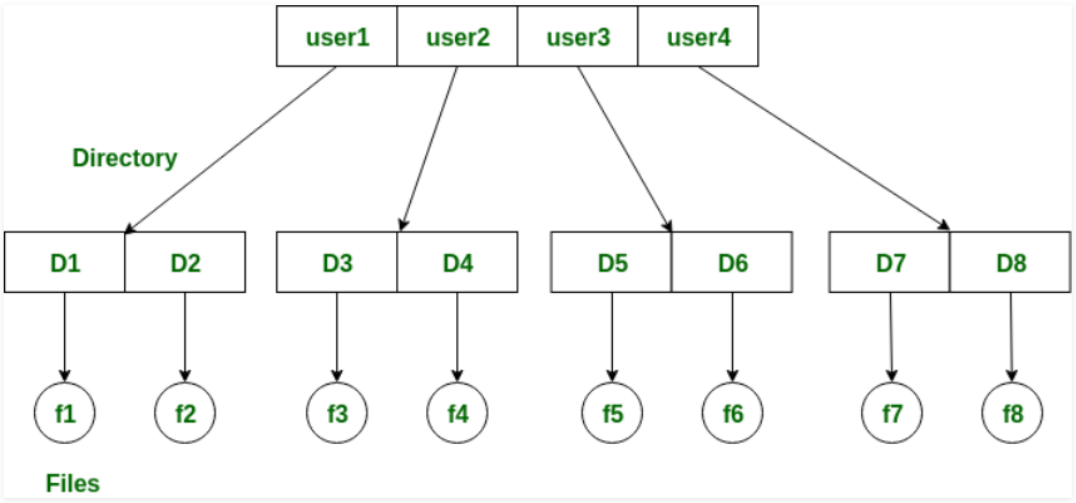
- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

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

2. **Two-level directory**

In the two-level directory structure, each user has their own user files directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. system's master file directory (MFD) is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



Advantages:

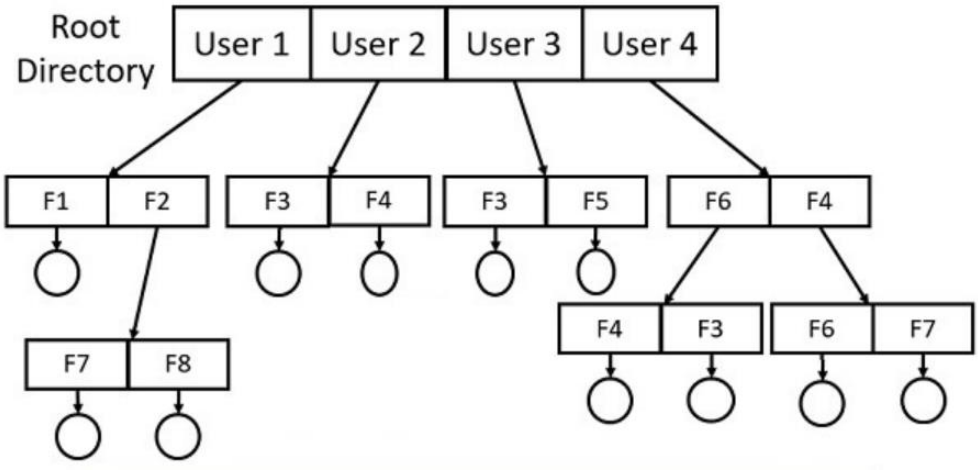
- We can give full path like /User-name/directory-name/.
- Different users can have the same directory as well as the file name.
- Searching of files becomes easier due to pathname and user-grouping.

Disadvantages:

- A user is not allowed to share files with other users.
- Still, it not very scalable, two files of the same type cannot be grouped together in the same user.

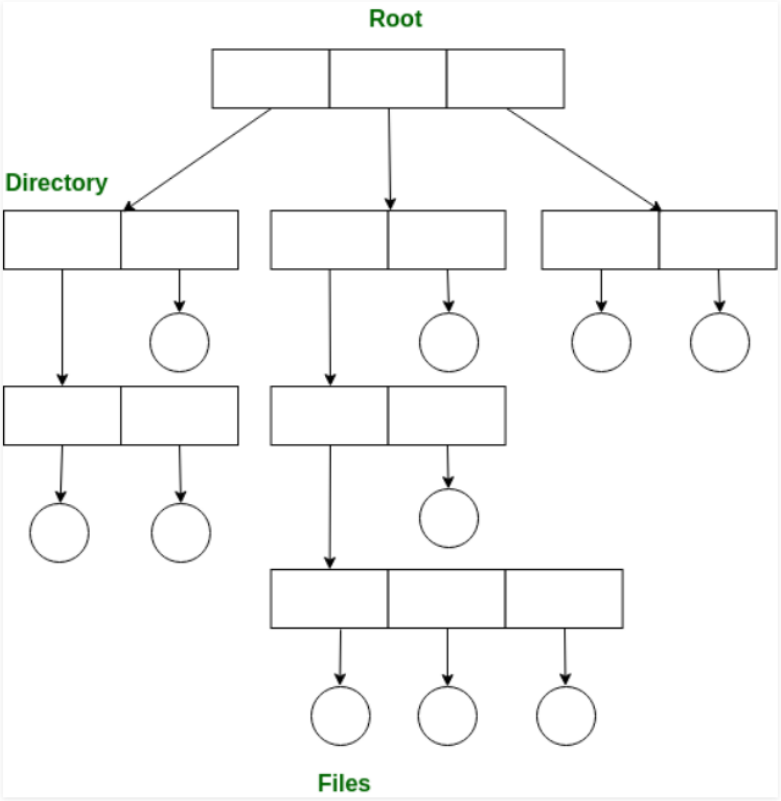
3. Hierarchical directory

In Hierarchical directory structure, the users can create directories under the root directory and can also create sub-directories under this structure. As the user is free to create many sub-directories, it can create different sub-directories for different file types.



4. Tree structure directory

Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows the user to create their own subdirectories and to organize their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

Advantages:

- Very general, since full pathname can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute paths as well as relative.

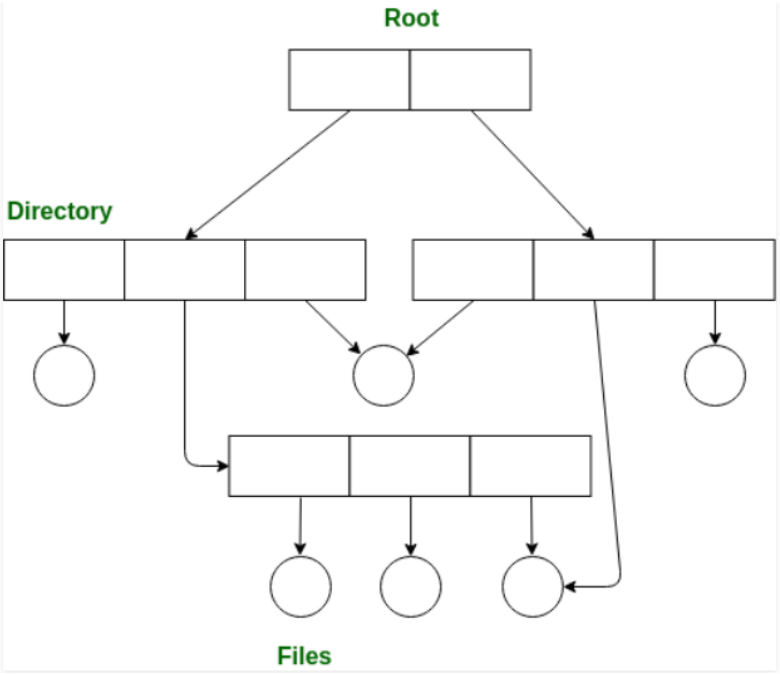
Disadvantages:

- Every file does not fit into the hierarchical model; files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

5. **Acyclic graph directory**

An acyclic graph is a graph with no cycle and allows us to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.
It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.

It is the point to note that the shared file is not the same as the copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:
- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:
- We share the files via linking, in case deleting it may create the problem,
- If the link is a soft link then after deleting the file we left with a dangling pointer.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.

Path names

1. **Absolute Path name –** In this method, each file is given an **absolute path** name consisting of the path from the root directory to the file. As an example, the path **/usr/ast/mailbox** means that the root directory contains a subdirectory usr, which in turn contains a subdirectory ast, which contains the file mailbox.

Absolute path names always start at the root directory and are unique.

In UNIX the components of the path are separated by '/'. In Windows, the separator is ''.
Windows usrastmailbox
UNIX /usr/ast/mailbox

2. **Relative Path name –** This is used in conjunction with the concept of the **working directory** (also called the **current directory**). A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory.
   For **example**, if the current working directory is /usr/ast, then the file whose absolute path is /usr/ast/mailbox can be referenced simply as mailbox.
   In other words, the UNIX command :**cp/usr/ast/mailbox/usr/ast/mailbox.bak**
   and the command : **cp mailbox mailbox.bak**
   do exactly the same thing if the working directory is /usr/ast.

Operations on directory

1. Searching

A directory can be **searched** for a particular file or for another directory. It can also be searched to list all the files with the same name.

2. Creating

A new file can be created and inserted to the directory or new directory can be created keeping in mind that its name must be unique under that particular directory.

3. Deleting

If a file is no longer needed by the user, it can be deleted from the directory. The entire directory can also be deleted if it is not needed. An empty directory can also be deleted. When a directory is empty it is resembled by dot and dotdot.

4. List a directory

List of all the files in the directory can be retrieved and also the contents of the directory entry, for each file in a list. To read the list of all the files in the directory, it must be opened and after reading the directory must be closed to free up the internal tablespace.

5. Renaming

The name of the file or a directory represents the content it holds and its use. The file or directory can be renamed in case, the content inside or the use of file get changed. Renaming the file or directory also changes its position inside the directory.

6. Link

The file can be allowed to appear in more than one directory. Here, the system call creates a link between the file and the name specified by the path where the file is to appear.

7. Unlink

If the file is unlinked and is only present in one directory its directory entry is removed. If the file appears in multiple directories, only the link is removed.

Protection

When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection). Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed. File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally.
Protection can be provided in many ways. For a single-user laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. In a larger multiuser system, however, other mechanisms are needed.

Types of Access

The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus, we could provide complete protection by prohibiting access. Alternatively, we could provide free access with no protection. Both approaches are too extreme for general use. What is needed is controlled access. Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

- Read. Read from the file.
- Write. Write or rewrite the file.
- Execute. Load the file into memory and execute it.
- Append. Write new information at the end of the file.
- Delete. Delete the file and free its space for possible reuse.
- List. List the name and attributes of the file.

Other operations, such as renaming, copying, and editing the file, may also be controlled. For many systems, however, these higher-level functions may be implemented by a system program that makes lower-level system calls. Protection is provided at only the lower level. For instance, copying a file may be implemented simply by a sequence of read requests. In this case, a user with read access can also cause the file to be copied, printed, and so on.

Access control

The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement identity dependent access is to associate with each file and directory an access-control list (ACL) specifying user names and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

This approach has the advantage of enabling complex access methodologies. The main problem with access lists is their length. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:

- Owner. The user who created the file is the owner.
- Group. A set of users who are sharing the file and need similar access is a group, or work group.
- Universe. All other users in the system constitute the universe.

Access control matrix

An access control matrix is a table that defines access permissions between specific subjects and objects. A matrix is a data structure that acts as a table lookup for the operating system. It is a matrix that has specific access permissions defined by user and detailing what actions they can perform.

	F1	F2	F3	Printer
D1	read		read	
D2				print
D3		read	execute	
D4	read write		read write	