III File-System Interface – File concept

File: A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Attributes

Different OS keep track of different file attributes, including:

Name - Some systems give special significance to names, and particularly extensions (.exe, .txt, etc.), and some do not. Some extensions may be of significance to the OS (.exe), and others only to certain applications (.jpg)

Identifier

Type - Text, executable, other binary, etc.

Location - on the hard drive.

Size

Protection

Time & Date

User ID

File Operations

The file ADT supports many common operations:

- Creating a file
- Writing a file
- Reading a file

File Structure

Some files contain an internal structure, be known to the OS.

For the OS to support particular file formats increases the size and complexity of the OS.

UNIX treats all files as sequences of bytes, with no further consideration of the internal structure. (With the exception of executable binary programs, which it must know how to load and find the first executable statement, etc.)

Macintosh files have **two *forks* - a *resource fork*, and a *data fork***. The resource fork contains information relating to the UI, such as icons and button images, and can be modified independently of the data fork, which contains the code or data as appropriate.

A File Structure should be according to a required format that the operating system can understand.

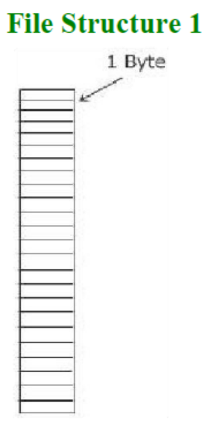
A **file** has a certain defined structure according to its type.

A **text file** is a sequence of characters organized into lines.

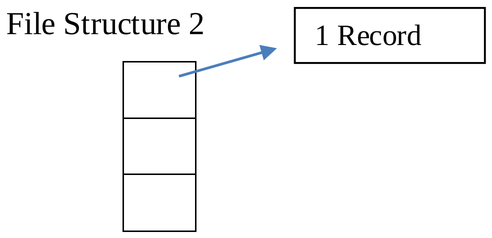
A **source file** is a sequence of procedures and functions.

An **object file** is a sequence of bytes organized into blocks that are understandable by the machine.

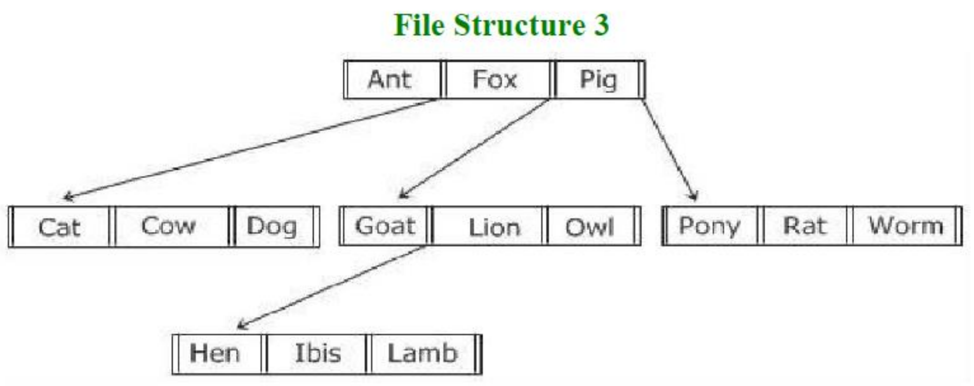
Files can be structured in several ways in which three common structures are given in this tutorial with their short description one by one.



Here, as you can see from the above figure, the file is an unstructured sequence of bytes. Therefore, the OS doesn't care about what is in the file, as all it sees are bytes.



Now, as you can see from the above figure that shows the second structure of a file, where a file is a sequence of fixed-length records where each with some internal structure. Central to the idea about a file being a sequence of records is the idea that read operation returns a record and write operation just appends a record.



Now in the last structure of a file that you can see in the above figure, a file basically consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is stored on the field, just to allow the rapid searching for a specific key.

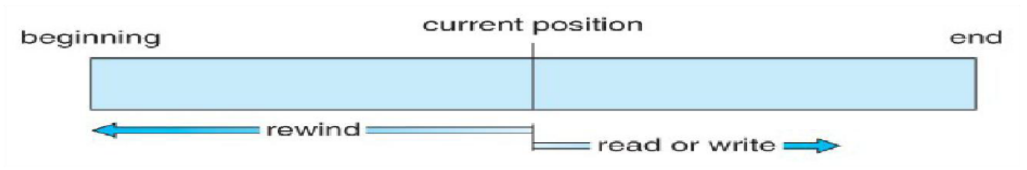
Access Methods

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.



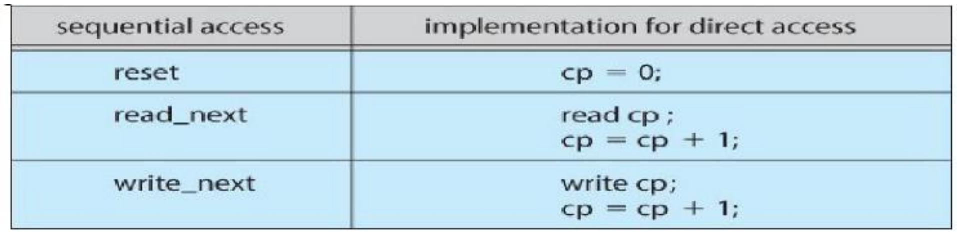
Sequential-access file.

Direct/Random access

Random access file organization provides, accessing the records directly.

Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.

The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.



sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

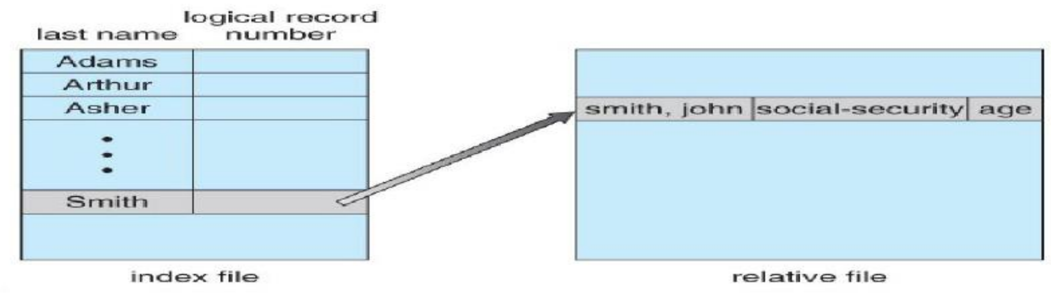
Simulation of sequential access on a direct-access file.

Indexed sequential access

This mechanism is built up on base of sequential access.

An index is created for each file which contains pointers to various blocks.

Index is searched sequentially and its pointer is used to access the file directly.



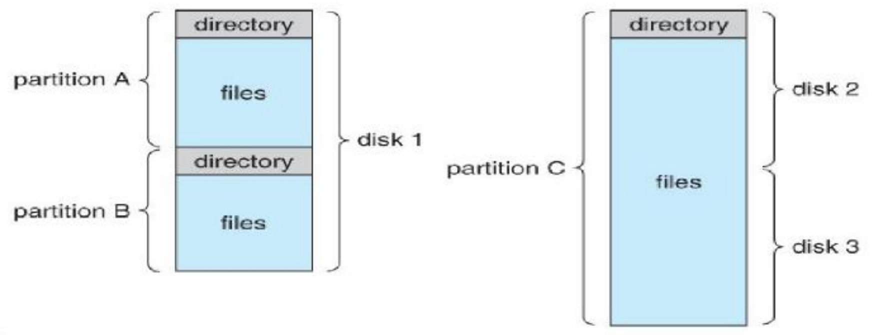
Example of index and relative files.

Directory Structure:

A directory is a container that is used to contain folders and file. It organizes files and folders into hierarchical manner.

Storage Structure

A disk can be used in its entirety for a file system. Alternatively a physical disk can be broken up into multiple ***partitions, slices, or mini-disks***, each of which becomes a virtual disk and can have its own file system(or be used for raw storage, swap space, etc.)Or, multiple physical disks can be combined into one *volume*, i.e. a larger virtual disk, with its own file system spanning the physical disks.



A typical file-system organization

Directory Overview

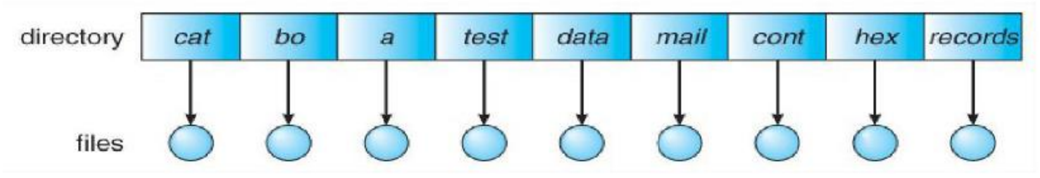
Directory operations to be supported include:

- Search for a file
- Create a file - add to the directory
- Delete a file - erase from the directory
- List a directory - possibly ordered in different ways.
- Rename a file - may change sorting order
- Traverse the file system.

<u>Directory organization</u>

Single-Level Directory

Simple to implement, but each file must have a unique name. The simplest method is to have one big list of all the files on the disk. The entire system will contain only one directory **which is supposed to mention all the files present in the file system. The** directory **contains one entry per each file present on the file system.**



Single-level directory.

Two-Level Directory

Each user gets their own directory space.

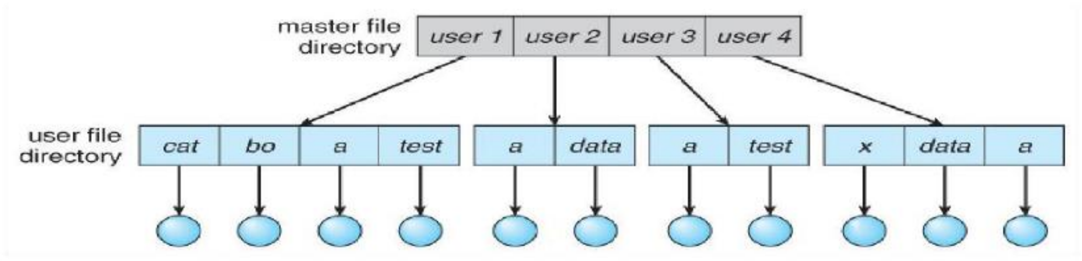
File names only need to be unique within a given user's directory.

A master file directory is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.

A separate directory is generally needed for system (executable) files.

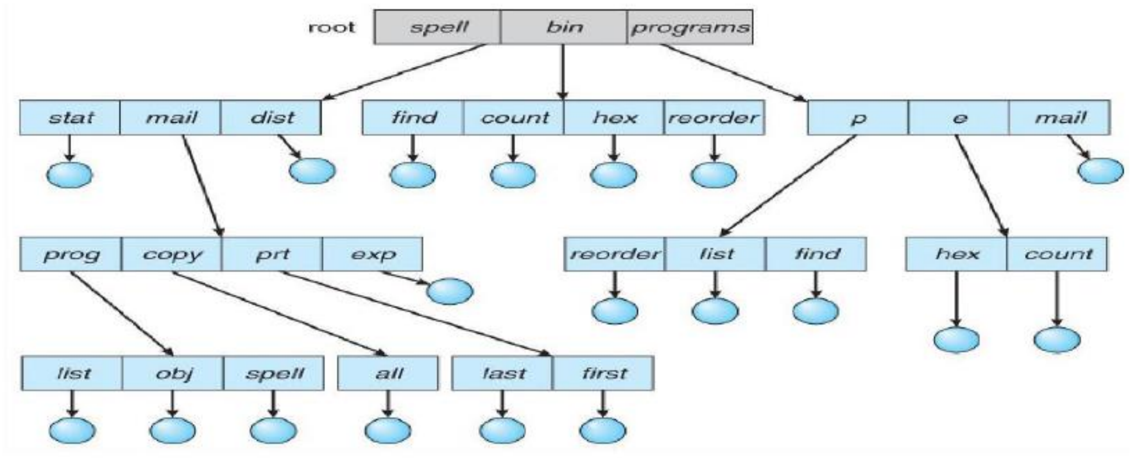
Systems may or may not allow users to access other directories besides their own

- If access to other directories is allowed, then provision must be made to specify the directory being accessed.
- If access is denied, then special consideration must be made for users to run programs located in system directories. A *search path* is the list of directories in which to search for executable programs, and can be set uniquely for each user.



Two-level directory structure.

Tree-Structured Directories



An obvious extension to the two-tiered diractory structure, and the one with which we are all most familiar. Each user / process has the concept of a ***current directory*** from which all (relative) searches take place. Files may be accessed using either absolute pathnames (relative to the root of the tree) or relative pathnames (relative to the current directory.)Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands. One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

Acyclic-Graph Directories

When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user / process), it can be useful to provide an acyclic-graph structure. (Note the *directed* arcs from parent to child.)

UNIX provides two types of *links* for implementing the acyclic-graph structure. (See "man ln" for more details.)

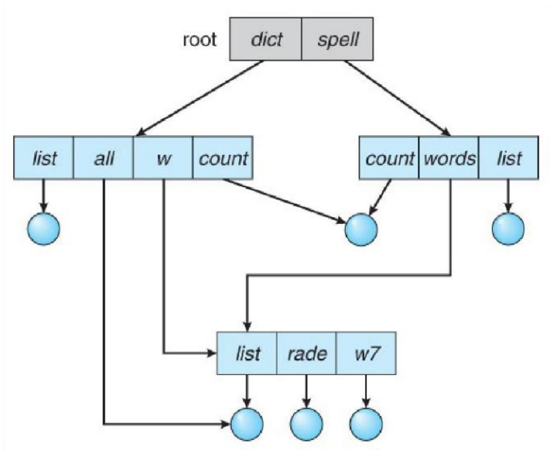
- **A *hard link*** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same file system.
- **A *symbolic link*** that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other file systems, as well as ordinary files in the current file system.

Windows only supports symbolic links, termed *shortcuts.*

Hard links require a *reference count*, or *link count* for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.

For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:

- One option is to find all the symbolic links and adjust them also.
- Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
- What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?

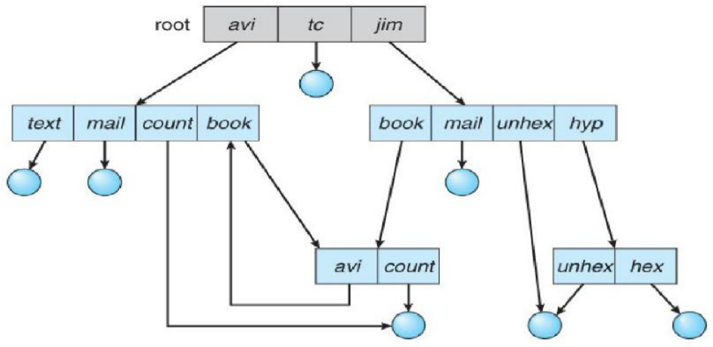


Acyclic-graph directory structure.

General Graph Directory

If cycles are allowed in the graphs, then several problems can arise:

- Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories.)
- Sub-trees can become disconnected from the croesmt of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem.



General graph directory.