**File system**

**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 Structure**

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.
* When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

**File Type**

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

**Ordinary files**

* These are the files that contain user information.
* These may have text, databases or executable program.
* The user can apply various operations on such files like add, modify, delete or even remove the entire file.

**Directory files**

* These files contain list of file names and other information related to these files.

Special files

* These files are also known as device files.
* These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

* **Character special files** − data is handled character by character as in case of terminals or printers.
* **Block special files** − data is handled in blocks as in the case of disks and tapes.
* **File Attributes**
* **Name** – only information kept in human-readable form
* **Identifier** – unique tag (number) identifies file within file system
* **Type** – needed for systems that support different types
* **Location** – pointer to file location on device
* **Size** – current file size
* **Protection** – controls who can do reading, writing, executing
* **Time, date, and user identification** – data for protection, security, and usage monitoring
* Information about files are kept in the directory structure, which is maintained on the disk

**File organization /Access Mechanisms**

File organization/access mechanism refers to the manner in which the records of a file may be accessed.

**Types of file organization**

* Sequential access File organization
* Direct/Random access File organization
* Indexed sequential access File organization

**Sequential access file organization**

In sequential access file organization is all 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.

**Direct/Random access file organization**

* 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.

**Indexed sequential access file organization**

* 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.

**Space Allocation**

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

* Contiguous Allocation
* Linked Allocation
* Indexed Allocation

**Contiguous Allocation**

* Each file occupies a contiguous address space on disk.
* Assigned disk address is in linear order.
* Easy to implement.
* External fragmentation is a major issue with this type of allocation technique.

**Linked Allocation**

* Each file carries a list of links to disk blocks.
* Directory contains link / pointer to first block of a file.
* No external fragmentation
* Effectively used in sequential access file.
* Inefficient in case of direct access file.

**Indexed Allocation**

* Provides solutions to problems of contigous and linked allocation.
* A index block is created having all pointers to files.
* Each file has its own index block which stores the addresses of disk space occupied by the file.
* Directory contains the addresses of index blocks of files.

**Blocking and Buffering**

**Block:** smallest amount of data that can be read from or written to secondary storage at one time. Often generalized to mean any chunk of data that can be treated as a unit (for reading, writing, organizing). We will distinguish between disk blocks (physical) and program defined blocks (logical).

**Blocking:** the process of grouping several components into one block

**Clustering:** grouping file components according to access behaviour

**Considerations affecting block size:**

1. size of available main memory
2. space reserved for programs (and their internal data space) that use the files
3. size of one component of the block
4. characteristics of the external storage device used

**Buffering:** Software interface that reconciles blocked components of the file with the program that accesses information as single components.

A buffering interface is of one of two types:

**blocking routine**

**deblocking routine**.

**Blocking Routine:** stores components from the program into a buffer (in main memory)

**Deblocking Routine:** accesses one block from the file (,places it in memory) and sends one component at a time to the program.

# File descriptor

a **file descriptor** (**FD**, less frequently **fildes**) is an abstract indicator used to access a [file](https://en.wikipedia.org/wiki/File_(computing)) or other [input/output](https://en.wikipedia.org/wiki/Input/output) [resource](https://en.wikipedia.org/wiki/System_resource), such as a [pipe](https://en.wikipedia.org/wiki/Pipe_(Unix)) or [network socket](https://en.wikipedia.org/wiki/Network_socket).

File descriptors form part of the [POSIX](https://en.wikipedia.org/wiki/POSIX) [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface). A file descriptor is a non-negative [integer](https://en.wikipedia.org/wiki/Integer), generally represented in the [C](https://en.wikipedia.org/wiki/C_(programming_language)) programming language as the type int (negative values being reserved to indicate "no value" or an error condition).

Each Unix [process](https://en.wikipedia.org/wiki/Process_(computing)) (except perhaps a [daemon](https://en.wikipedia.org/wiki/Daemon_(computer_software))) should expect to have three standard POSIX file descriptors, corresponding to the three [standard streams](https://en.wikipedia.org/wiki/Standard_streams):

|  |  |  |  |
| --- | --- | --- | --- |
| **Integer value** | **Name** | **<**[**unistd.h**](https://en.wikipedia.org/wiki/Unistd.h)**> symbolic constant**[[1]](https://en.wikipedia.org/wiki/File_descriptor#cite_note-1) | **<**[**stdio.h**](https://en.wikipedia.org/wiki/Stdio.h)**> file stream**[[2]](https://en.wikipedia.org/wiki/File_descriptor#cite_note-2) |
| 0 | [Standard input](https://en.wikipedia.org/wiki/Stdin) | STDIN\_FILENO | Stdin |
| 1 | [Standard output](https://en.wikipedia.org/wiki/Stdout) | STDOUT\_FILENO | Stdout |
| 2 | [Standard error](https://en.wikipedia.org/wiki/Stderr) | STDERR\_FILENO | Stderr |

Operations on file descriptors

The following lists typical operations on file descriptors on modern [Unix-like](https://en.wikipedia.org/wiki/Unix-like) systems. Most of these functions are declared in the <unistd.h> header, but some are in the <fcntl.h> header instead.

**Creating file descriptors**

* [open](https://en.wikipedia.org/wiki/Open_(system_call))()
* creat()
* socket()
* accept()
* socketpair()
* pipe()
* opendir()

**Deriving file descriptors**[[edit](https://en.wikipedia.org/w/index.php?title=File_descriptor&action=edit&section=4)]

* dirfd()
* fileno()

**Operations on a single file descriptor**

* [read](https://en.wikipedia.org/wiki/Read_(system_call))(), [write](https://en.wikipedia.org/wiki/Write_(system_call))()
* readv(), writev()
* pread(), pwrite()
* recv(), send()
* recvmsg(), sendmsg() (including allowing sending FDs)
* sendfile()
* fdopen()
* ftruncate()
* fsync()

## File descriptors as capabilities

## Unix file descriptors behave in many ways as [capabilities](https://en.wikipedia.org/wiki/Capability-based_security). They can be passed between processes across [Unix domain sockets](https://en.wikipedia.org/wiki/Unix_domain_socket) using the [sendmsg](https://en.wikipedia.org/w/index.php?title=Sendmsg&action=edit&redlink=1)() system call. Note, however, that what is actually passed is a reference to an "open file description" that has mutable state (the file offset, and the file status and access flags). This complicates the secure use of file descriptors as capabilities, since when programs share access to the same open file description, they can interfere with each other's use of it by changing its offset or whether it is blocking or non-blocking, for example. In operating systems that are specifically designed as capability systems, there is very rarely any mutable state associated with a capability itself.

# Directory structure

A **directory structure** is the way an [operating system](https://en.wikipedia.org/wiki/Operating_system)'s [file system](https://en.wikipedia.org/wiki/File_system) and its files are displayed to the user. Files are typically displayed in a [hierarchical tree structure](https://en.wikipedia.org/wiki/Hierarchical_tree_structure).

**Directory Structure**

There are many types of directory structure in Operating System. They are as follows :-  
  
1) Single Level Directory  
2) Two Level Directory  
3) Tree Structured Directory  
4) Acyclic Graph Directory  
5) General Graph Directory

**1) Single Level Directory**

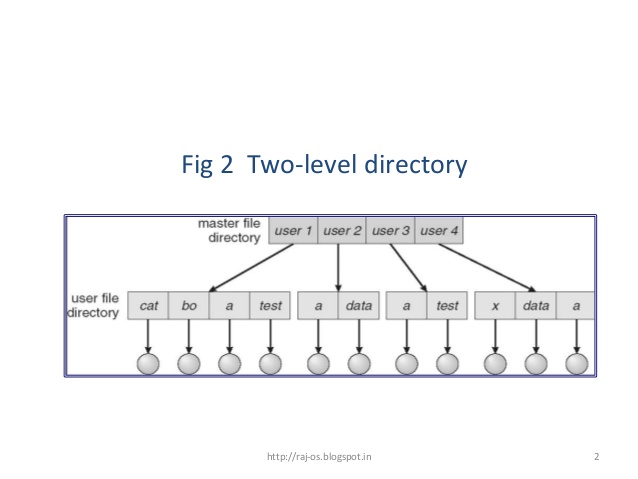
## In a single level directory system, all the files are placed in one directory.

## This is very common on single-user OS's. A single-level directory has significant limitations, however, when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If there are two users who call their data file "test", then the unique-name rule is violated. Although file names are generally selected to reflect the content of the file, they are often quite limited in length. Even with a single-user, as the number of files increases, it becomes difficult to remember the names of all the files in order to create only files with unique names shown in Fig

## 1911_Single level directory.png

***Limitations of Single Level Directory***  
  
a) Since all files are in the same directory, they must have unique name.  
  
b) If two user call their data free test, then the unique name rule is violated.  
  
c) Files are limited in length.  
  
d) Even a single user may find it difficult to remember the names of all files as the number of file increases.  
  
e) Keeping track of so many file is daunting task.

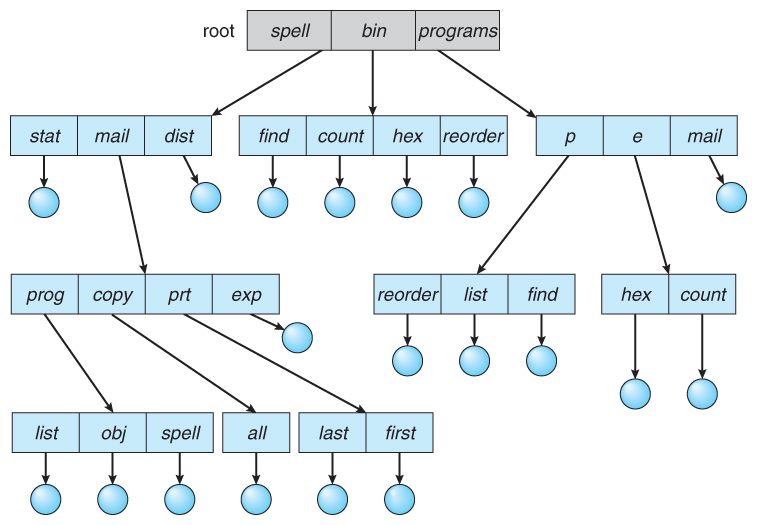
**2) Two Level Directory**



i) Each user has Its own User File Directory (UFD).  
  
ii) When the user job start or user log in, the system Master File Directory (MFD) is searched. MFD is indexed by user name or Account Number.  
  
iii) When user refers to a particular file, only his own UFD is searched.  
  
Thus different users may have files with same name. To have a particular file uniquely, in a two level directory, we must give both the user name and file name.  
  
*A two level directory can be a tree or an inverted tree of height 2*  
  
The root of a tree is Master File Directory (MFD).   
  
Its direct descendents are User File Directory (UFD). The descendents of UFD's are file themselves.  
  
The files are the leaves of the tree. 

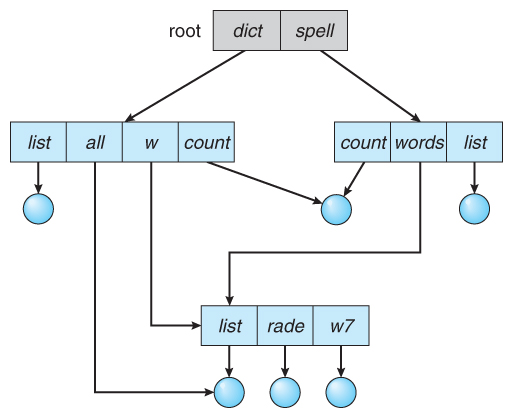
***Limitations of Two Level Directory***  
  
The structure effectively isolates one user from another.

**3) Tree Structured Directory**



A directory (or Sub directory) contains a set of files or sub directories. All   
directories has the same internal format.  
  
i) One bit in each directory entry defines the entry.  
  
ii) Special calls are used to create and delete directories.  
  
iii) Each process has a current directory. Current directory should contain most of the files that are of current interest to the process.  
  
iv) When a reference is made to a file, the current directory is searched.  
  
v) The user can change his current directory whenever he desires.  
  
vi) If a file is not needed in the current directory then the user usually must either specify a path name or change the current directory.   
**Paths**can be of two types :-  
  
**a) *Absolute Path***  
Begins at root and follows a path down to the specified file.  
  
**b) *Relative Path***  
Defines a path from current directory.  
  
vii) Deletions if directory is empty, its entry in the directory that contains it can simply deleted. If it is not empty : One of the Two approaches can be taken :-  
a)User must delete all the files in the directory.  
  
b)If any sub directories exist, same procedure must be applied.   
The UNIX rm command is used.  
MS dos will not delete a directory unless it is empty.

**4) Acyclic Graph Directory**



Acyclic Graph is the graph with no cycles. It allows directories to share sub directories and files. With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the another.

## General graph directory:

## Image result for general graph directory diagram

## The general graph directory is formed by adding links into an existing tree structure. It overcomes the problem of acyclic graph by allows the cycles in a directory. Thus it avoids the searching of a component twice in a subdirectory

## Blocks

In unix ,afile system has four structural sections known as blocks:the boot block,the super block,the inode block,and the datablock.

The boot block,super block and inode blocks are fixed at the beginning of the disk.they occupy the same locations on the disk even when the disk is recoganized.

These blocks are shown in the figure

---------------------------------------------------------------

| | | | | | | | | | | | | | |

| B. B. | S. B. | Inodes | | | ... | Data Blocks |

| | | | | | | | | | | | | | |

---------------------------------------------------------------

Each filesystem contains:

1.a *boot block* located in the first few sectors of a file system. The boot block contains the initial bootstrap program used to load the operating system.

Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.

2.a *super block* describes the state of the file system: the total size of the partition, the block size, pointers to a list of free blocks, the inode number of the root directory, magic number, etc.

3.a linear array of *inodes* (short for ``index nodes''). There is a one to one mapping of files to inodes and vice versa. An inode is identified by its ``inode number'', which contains the information needed to find the inode itself on the disk

Thus, while users think of files in terms of file names, Unix thinks of files in terms of inodes.

4.*data blocks* blocks containing the actual contents of files

An inode is the ``handle'' to a file and contains the following information:

* file ownership indication
* file type (e.g., regular, directory, special device, pipes, etc.)
* file access permissions. May have setuid (sticky) bit set.
* time of last access, and modification
* number of links (aliases) to the file
* pointers to the data blocks for the file
* size of the file in bytes (for regular files), major and minor device numbers for special devices.

**Unix**

The Unix operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

* Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
* There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
* Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
* A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture

basic block diagram of a Unix system −



The main concept that unites all the versions of Unix is the following four basics −

* **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
* **Shell** − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.
* **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
* **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

**Unix file system**

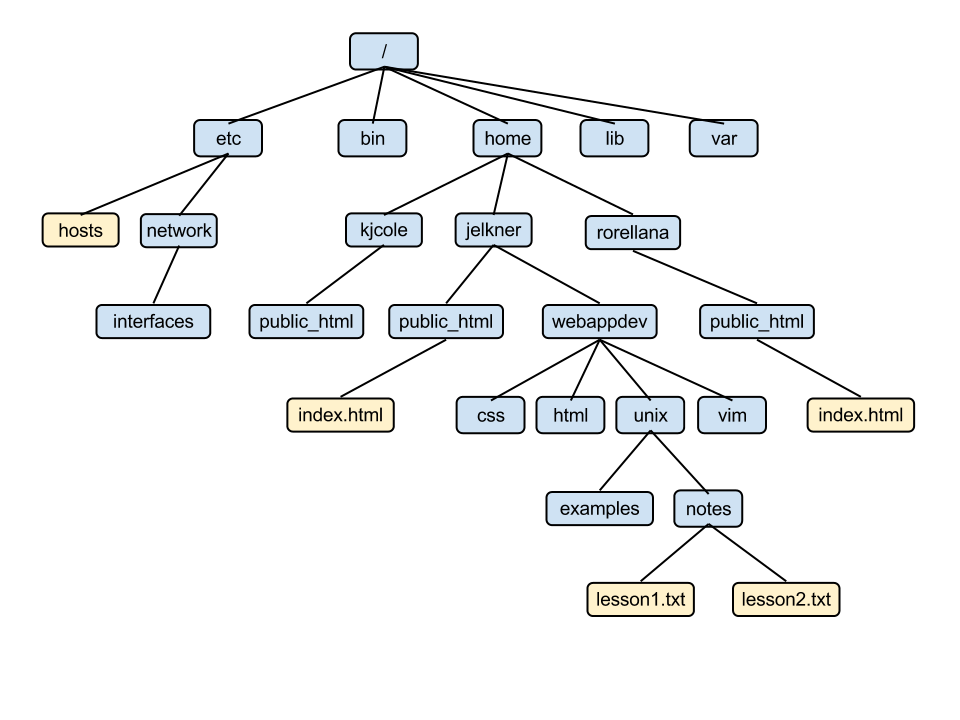
A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.

Your hard drive can have various partitions which usually contain only one file system, such as one file system housing the **/file system** or another containing the **/home file system**.

One file system per partition allows for the logical maintenance and management of differing file systems.

Everything in Unix is considered to be a file, including physical devices such as DVD-ROMs, USB devices, and floppy drives.

Directory Structure



Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A Unix filesystem is a collection of files and directories that has the following properties −

* It has a root directory (**/**) that contains other files and directories.
* Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an **inode**.
* By convention, the root directory has an **inode** number of **2** and the **lost+found** directory has an **inode** number of **3**. Inode numbers **0** and **1** are not used. File inode numbers can be seen by specifying the **-i option** to **ls command**.
* It is self-contained. There are no dependencies between one filesystem and another.

The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix −

|  |  |
| --- | --- |
| **S.No.** | **Directory & Description** |
| 1 | **/**  This is the root directory which should contain only the directories needed at the top level of the file structure |
| 2 | **/bin**  This is where the executable files are located. These files are available to all users |
| 3 | **/dev**  These are device drivers |
| 4 | **/etc**  Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages |
| 5 | **/lib**  Contains shared library files and sometimes other kernel-related files |
| 6 | **/boot**  Contains files for booting the system |
| 7 | **/home**  Contains the home directory for users and other accounts |
| 8 | **/mnt**  Used to mount other temporary file systems, such as **cdrom** and **floppy** for the **CD-ROM** drive and **floppy diskette drive**, respectively |
| 9 | **/proc**  Contains all processes marked as a file by **process number** or other information that is dynamic to the system |
| 10 | **/tmp**  Holds temporary files used between system boots |
| 11 | **/usr**  Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others |
| 12 | **/var**  Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data |
| 13 | **/sbin**  Contains binary (executable) files, usually for system administration. For example, ***fdisk*** and ***ifconfig*** utilities |
| 14 | **/kernel**  Contains kernel files |

s