File System Components:

Files: Stores related data.

Directories: Organizes and provides information about the files.

Partitions: Physically or logically separates large collections of directories.

Files are necessary because:

- Virtual address space of a process may not be large enough to hold information.
- Information must be persistent - be available past the termination of a process.
- Concurrent access by multiple processes to the information may be necessary.

Files: The user's perspective

Files: A named collection of related information that is recorded on secondary storage.

Files are an abstract data type:

A sequence of bits, bytes, lines or records whose meaning is defined by the file's creator.

File attributes:

- Name: string of characters.
- Structure: internal organization.
- Type: text, source, object, executable, database, etc.
- Other attributes: time, date, owner, size, protection.

File Naming:

All OSs allow strings of one to eight letters as legal file names - some up to 255 characters, some are case sensitive.

File extensions are often just conventions. Sometimes they are important when the application processes many different types of files, e.g. C compiler.

Windows uses file extension to determine application to launch when user double clicks on the file name.

File Structure: There are several possibilities:

File is an unstructured sequence of bytes. Provides maximum flexibility to users.

Meaning of bytes imposed by application.

UNIX, DOS, Window95.

Other possibilities:

- File is a sequence of fixed-length records.
- File is a tree of records of variable length.

File Type: UNIX and DOS support regular files and directories.

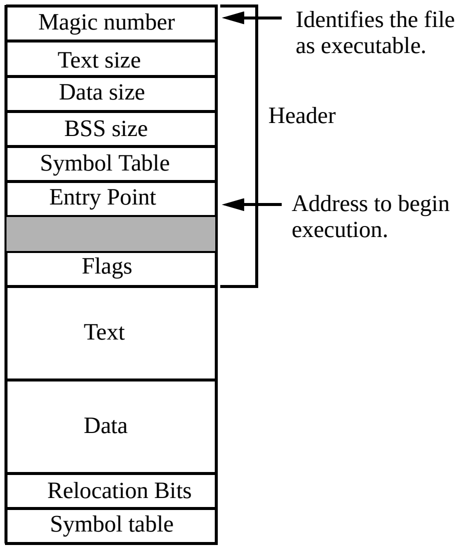
Regular files hold the data.

Directories are **system files** for maintaining the structure of the file system.

UNIX additionally supports character and block special files for modeling serial I/O devices and disks.

Regular files are either ASCII or binary.

Old UNIX format for binaries had 5 sections



Other File Attributes:

Protection, creator, owner.

Read-only, hidden, system file, archive.

Creation time, time of last access, modification time.

Current size.

File Operations:

Create: Sets file attributes, e.g. name, type, protection, etc. No data.

Open: Get and check file attributes. Required before a file can be used.

Truncate: Preserves attributes, contents erased.

Append: Adds data onto end of file, increasing size.

Write: OS writes data at current position in file.

Read: OS reads data.

Reposition or SEEK: Sets OS maintained file pointer.

Get/Set attributes: Returns/sets file attributes such as size, protection, etc.

Rename/Copy: Change name attribute/make a copy under different name.

Close: Free up internal OS table space. Forces data out to the disk.

Delete: Remove the file and free space.

System calls create, open, truncate and append typically return a file pointer. This pointer used in write, read and reposition system calls.

File Access:

Sequential access: Information in file is processed in order from beginning to end.

- Most common mode of access.
- Reads return the next portion of the file and automatically advance the pointer.
- Writes append data onto end of file and advance pointer to new end of file.
- Based on the tape model of a file.

Direct Access: Programs can read and write records in arbitrary order.

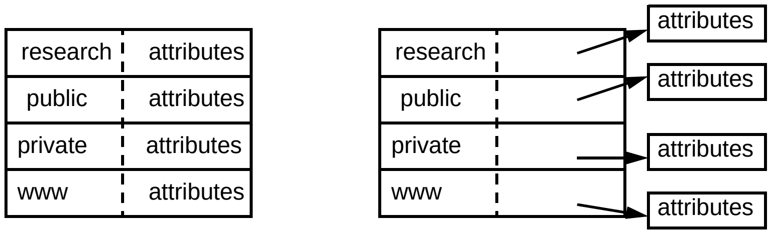
- File made up of fixed-length logical records.
- File viewed as a numbered sequence of blocks or records.
- Read and write operations require an extra parameter, a block or record number.
- Databases are often accessed this way.

Not all OSs support both sequential and direct file access.

Directories: The user's perspective

Contains a number of entries, one per file.

2 possible organizations:



Attributes include the file attributes, e.g. file name, size, etc. + the disk addresses where the data are stored.

When a file is opened, the directory is searched and the attributes copied to a table in main memory.

Directories are organized hierarchically, as a directory tree.

Path Names:

Directory trees require a method of specifying individual files.

UNIX path separator is '/' while DOS/Windows is '\'.

Absolute path name:

A path is specified from the root directory to the file,
e.g., /usr/jimp/421/final_exam.txt

Relative path name:

Used in conjunction with working directory, e.g.
If a program's working directory is /usr/jimp/421
File can be opened as open(final_exam.txt, append).

System calls are available to change a programs working directory.

Special directories, '.' and '..' indicate current working directory and parent directory, e.g.,
'mv ./final_exam.txt, ..' moves final_exam.txt into /usr/jimp.

Directory Operations:

Create: 'mkdir' creates an empty directory except of '.' and '..'.

Delete: 'rmdir' removes a directory as long as it is empty. 'rm -rf *' also removes files and directories, recursively.

OpenDir/CloseDir: Analogous to file operations 'open' and 'close'. Directories must be opened before they can be read.

ReadDir: Returns the next entry in an open directory.

Rename: Analogous to rename operation on files.

Link: Allows a file to appear in more than one directory,
e.g. 'ln -s /usr/jimp/421/final_exam.txt /usr/jimp/'

Unlink: Removes the file (delete) if last reference, otherwise only the directory entry is deleted.

File Systems: The system implementation perspective

Concerned with:

- How files and directories are stored.
- How disk space is managed.
- How to make the system work efficiently and reliably.

Most important aspect of storage: How do we track which disk blocks belong to which files ?

Simplest method: Store file as **contiguous** block of data on the disk.

Advantages:

Simple to implement; save disk address of first block.
Good performance; Entire file can be read in a single operation.

Disadvantages:

Not possible unless maximum file size is known in advance.
External fragmentation results in much wasted space. Compaction is expensive.

Linked List Implementation:

Each file kept as a linked list of disk blocks.

First word of each block is a pointer to next block.

Directory entry stores address of first disk block.

Advantages:

Eliminates external fragmentation.

Disadvantages:

Random access is extremely slow.

Block size is no longer a power of two due to pointer storage. Programs like to read and write power of 2 sized blocks.

Linked List Implementation using an index:

Move the pointer address from disk block into a table:

Assume File A disk block address are 3, 7, 1, 5 and 4.

File A starts here

Physical Block	
0	
1	5
2	
3	7
4	0
5	4
6	
7	1

Table is loaded into main memory.

Adv:

Entire block can be used for data.

Random access is fast - no disk access for each pointer.

Directory entry still contains only starting block.

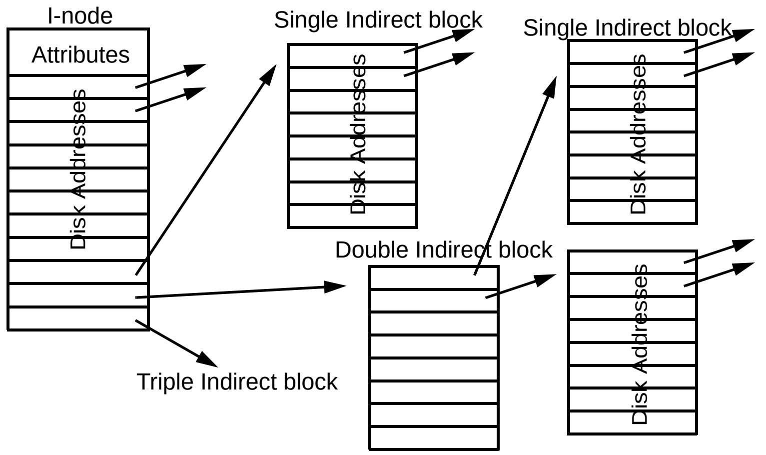
MS-DOS uses this method.

Disadvantage: Entire table must be in memory all the time.

With 1K block, 500MB drive and 4 bytes/entry -> ~2MBytes RAM !

I-node Implementation: The UNIX method.

Each file has a little table called an **i-node**.



Some disk block address stored in i-node itself. Good for small files.

Larger files use single, double and triple indirect block pointers.