File Systems

A file system needs to satisfy these general requirements.

We need some way of storing information

independently from a running program

so it can be used at a later time

permanently (or an approximation to it)

non-volatile storage

so it can be shared with other programs or users

An infinite variety of data is to be stored

The more information the OS knows about the data the more it can facilitate use of the data.

e.g. Executable files

Naming the data

The data needs to be stored and retrieved easily. We need a way to name the data.

What is a file?

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

The information is usually related in the sense that it is used by a particular program (or is a particular program).

Secondary storage is usually a disk drive but it could be a tape drive, or any other non-volatile device (SSD)

In some systems “files” are not always files e.g. In UNIX devices are “files” and so are some operating system data structures (e.g. /proc in Linux)

```
ls -l /proc
```


File system operations 1

On most systems these commands need security authorisation to perform and they work on the file as a whole.

Create

Need to specify information about the file:

- the name
- the file type (or some representation of the program associated with this file)
- do we need to specify the size of the file? (certainly helps with keeping storage contiguous but is usually regarded as an unnecessary restriction)

Creation needs to do something to the associated device - at least write to some structure (a directory).

Some systems allow transitory files to not be recorded permanently in secondary storage.

File system operations 2

Delete

Remove the file. Return the space used by the file. Zero the space?

What if the deletion was a mistake?

Deletion might keep the file in a “going away” area from which it can be retrieved.

Versioning file systems maintain multiple versions over time. They can also be used to track down intruders (self-securing storage)

Move

Moving a file can be performed in different ways depending on the before and after locations.

If both locations are on the same device the data doesn’t have to be copied and then the original deleted. Instead change information about the file.

File system operations 3

Copy

Most file systems preserve attributes (including last modified times) when a copy is made. This way a file can be last modified before it was created.

In some situations we can use copy on write rather than making complete copies. The file information needs to point to the original data.

Change attributes

We will see the different sorts of attributes shortly. Some of these should be changeable, others should be secured.

File system operations - read

These operations work on the files contents.

We need to know where the information is on the device.

Must specify what data to read, how much, and where to put it.

Sequential access

data is retrieved in the same order it is stored

there is a current position pointer somewhere

- may be stored within the using program
- may be stored within the file system (but separately for each process)
- has ramifications for distributed systems

Direct (or random) access

Easy on a disk device. Even easier on a solid state device.

The read specifies exactly where it wants to get the data from.

- it could be a byte offset
- or a record number

Some file systems let you specify record length when you create a file. Others leave all such control up to individual programs.

File system operations - write

Very similar to read but commonly requires the allocation of extra space.

Direct (random) access writes can create holes.

- The program "seeks" to the new position.
- If this is outside the bounds of the file then we have two choices.
- allocate all of the intervening space and fill it with some null value
- mark the intervening space in the directory as not allocated – this is known as a *sparse* file

If a process tries to read from the empty space of a sparse file it gets the null value for the record.

If a process writes to the empty space then real blocks are allocated and written to.

Design decisions

Files need to contain vastly different types of information.

Some of this information is tightly structured with lines, records etc.

Should the file system allow flexibility in how it deals with differently structured files?

At the bottom level the file system is working with discrete structures (**sectors** and **blocks**) of a definite size.

The most common solution is to treat files as a stream of bytes and any other structure is enforced by the programs using the files.

The work has to be done somewhere.

Some operating systems provide more facilities than others for dealing with a variety of file types.

File attributes

Information about the files.

These vary widely because of the previous design decisions.

Standard ones

file name – the full name includes the directories to traverse to find this file. How much space should the system allocate for a name? Many systems use a byte to indicate the file name length and so are limited to 255 characters. There are usually limitations on the characters you can use in filenames.

location – where is the file stored, some pointer to the device (or server) and the positions on the device

size of the file – either in bytes, blocks, number of records etc

owner information – usually the owner can do anything to a file

other access information – who should be allowed to do what

dates and times – of creation, access, modification

and file types…

File type

The more the system knows about file types the more it can perform appropriate tasks.

e.g.

Executable binaries can be loaded and executed.

Text files can be indexed.

Pictures can have thumbnails generated from them.

Files can automatically be opened by corresponding programs.

Also the system can stop the user doing something stupid like printing an mp3 file.

All operating systems "know" about executable binary files.

They have an OS specific structure – information for the loader about necessary libraries and where different parts should be loaded and where the first instruction is.

Dealing with file types

Windows deals with different file types using a simple extension on the file name.

The extensions are connected to programs and commands in the system registry.

But there is nothing to stop a user changing an extension (except a warning message).

UNIX uses magic numbers on the front of the file data.

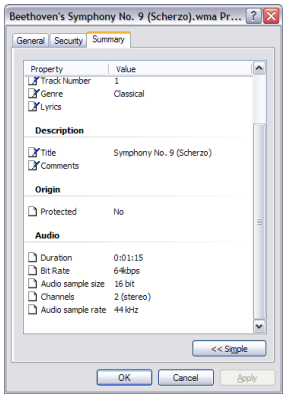
If the file is executed the magic number can be used to invoke an interpreter for example.

Windows does something similar with extra information about files.

Try using the `file` command on Linux (or Mac).

And `od` e.g.

```
od -a -N 32 README.rtf
```

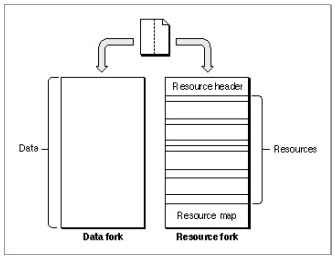


Macintosh solution

The Macintosh used 8 bytes to identify file types (4 for the creator and 4 for the type).

Not normally visible to the user. Therefore harder to change by accident.

Also more **structure** - each file has two components (one can be empty).



Resource fork (/rsrc)

Program code (originally), icons, menu items, window information, preferences.

Data fork

Holds the (possibly unstructured) data, program code

e.g. the text of a word processing document.

Each program includes in its resource fork a list of all the types of files it can work with.

In MacOS X if using the Unix File System the resource fork has merged back into the data fork.

NTFS files

NTFS takes a very general approach to file attributes.

NTFS views each file (or folder) as a set of file attributes.

The most important one is usually the data attribute.

New attributes can be added.

Attribute Type	Description
Standard Information	Information such as timestamp and link count.
Attribute List	Locations of all attribute records that do not fit in the MFT record.
File Name	Additional names, or hard links, can be included as additional file name attributes.
Data	File data. NTFS supports multiple data attributes per file. Each file typically has one unnamed data attribute. A file can also have one or more named data attributes, each using a particular syntax.

More NTFS attributes

Attribute Type	Description
Object ID	A volume-unique file identifier. Used by the distributed link tracking service. Not all files have object identifiers.
Logged Tool Stream	Similar to a data stream, but operations are logged to the NTFS log file just like NTFS metadata changes.
Reparse Point	Used for mounted drives and archives.
Index Root	Used to implement folders and other indexes.
Index Allocation	Used to implement the B-tree structure for large folders and other large indexes.
Bitmap	Used to implement the B-tree structure for large folders and other large indexes.
Volume Information	Used only in the $Volume system file. Contains the volume version.
Volume Name	Used only in the $Volume system file. Contains the volume label.

Alternate Data Streams (NTFS)

```
>dir
10/09/2004  03:22 p.m.    <DIR>          .
               0 File(s)              0 bytes
               2 Dir(s)  14,992,101,376 bytes free

>echo "this is an ADS attached to the 'ads test folder'"
> :ads0.txt

>dir
10/09/2004  03:23 p.m.    <DIR>          .
               0 File(s)              0 bytes
               2 Dir(s)  14,992,101,376 bytes free

>echo "this is an ADS attached to 'file1.txt'" >
file1.txt:ads1.txt

>dir
10/09/2004  03:25 p.m.    <DIR>          .
10/09/2004  03:25 p.m.                 0 file1.txt
               1 File(s)              0 bytes
               2 Dir(s)  14,992,101,376 bytes free

>echo "this is another ADS attached to 'file1.txt'" >
file1.txt:ads2.txt

>dir
10/09/2004  03:25 p.m.    <DIR>          .
10/09/2004  03:26 p.m.                 0 file1.txt
               1 File(s)              0 bytes
               2 Dir(s)  14,992,101,376 bytes free

>more < :ads0.txt
"this is an ADS attached to the 'ads test folder'"

>more < file1.txt:ads1.txt
"this is an ADS attached to 'file1.txt'"

>more < file1.txt:ads2.txt
"this is another ADS attached to 'file1.txt'"
```


Before next time

Read from the textbook

11.3 Directory and Disk Structure

19.5.1 NTFS Internal Layout