Representing files on disk

We know that disks deal with constant size blocks. All files and information about files must be stored in these blocks (usually accessible via a logical block number).

We need to know what we want our file system to look like to the users in terms of its structure.

We also need to know how we can place the required information on the disk devices.

Structure

Data about files and other information about storage on a device is called metadata.

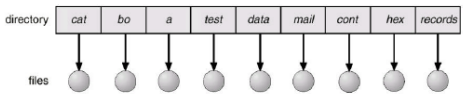
Usually a disk device has one or more directories to store metadata.

These directories can be arranged in different ways:

single level – simple, small systems did this, especially with small disk devices (floppies)

Disadvantages in finding files as the number of files grows (some implementations use a B-tree).

To be workable it requires very long filenames.



Multiple levels

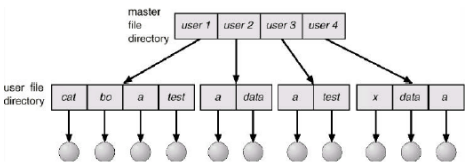
two level – The top level (master file directory) is one entry per user on a multi-user system, the next level (user file directory) looks like a single level system to each user.

Creating a user file directory is usually only allowed for administrators.

User file directories can be allocated like other files. What about the master file directory?

When people log in they are placed within their own directories. Any files mentioned are in that directory.

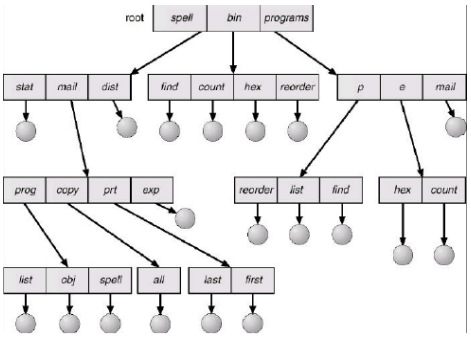
Can use full pathnames to refer to other user's files (if permissions allow it).



Normal file hierarchy

tree – as many directories as required. This facilitates the organisation of collections of files.

Directories are special files. Should users be able to write directly to their own directories?



Sharing files and directories

We commonly want the same data to be accessible from different places in the file hierarchy. e.g. Two people might be working on a project and they both want the project to be in their local directories.

This can be accomplished in several different ways:

- If the data is read only we could just make an extra copy.
- We could make two copies of the file record information (not the data).

There is a problem with consistency.

- There can be one *true* file entry in one directory. Other entries have some reference to this entry.

UNIX symbolic links and Windows shortcuts.

- There can be a separate table with file information. Then all directory entries for the same file just point to the corresponding information in this table.

UNIX and NTFS hard links.

Hard links

UNIX

ln *ExistingFilename NewFilename*

Each directory entry stores a pointer to the file's inode (more on those soon) which holds the real information about the file.

NTFS

fsutil hardlink create *NewFilename ExistingFilename*

Each directory entry holds copies of most file attributes plus a pointer to the file's Master File Table (MFT) file record (more on those soon).

NTFS updates the properties of a hard link only when a user accesses the original file by using the hard link.

Hard links do not have **security descriptors**; instead, the security descriptor belongs to the original file to which the hard link points.

Both only make links on the same volume.

UNIX symbolic links

The file called "linkFile" is actually just a text file with the contents "realFile".

The OS knows to treat it differently from other text files because of the "l" in the attributes on the left hand side.

If the original is moved then UNIX can't do anything about it. You get unable to open errors for example.

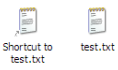
```
$ ls -al
total 1
drwxr-xr-x  2 Robert  None     0 Sep 14 16:59 .
drwxr-xr-x  3 Robert  None     0 Jul 16 16:56 ..
-rw-r--r--  1 Robert  None   304 Sep 12 10:34 .bash_history
-rw-r--r--  1 Robert  None     0 Sep 14 16:59 realFile

$ ln -s realFile linkFile

$ ls -l
total 0
lrwxrwxrwx  1 Robert  None     8 Sep 14 16:59 linkFile -> realFile
-rw-r--r--  1 Robert  None     0 Sep 14 16:59 realFile
```


Windows shortcuts

Make a file. e.g. test.txt.



Make a shortcut to the file.

Rename the original to "test".

By default Windows uses attributes such as file type, size and time of modification to find renamed shortcuts.

Much better under NTFS with the Distributed Link Tracking Client service running.

This keeps a log of changes made to files that have shortcuts. So the file can almost always be found.

It even works in distributed environments (sometimes).

```
15/09/2002  01:25 p.m.               615 Shortcut to test.txt.lnk
15/09/2002  01:24 p.m.                54 test.txt
```

Mac aliases

HFS+ has a unique, persistent identifier for each file or folder.
An alias stores this identifier with the pathname.

Originally the identifier was first used to find the file. Only if this failed was the pathname used.

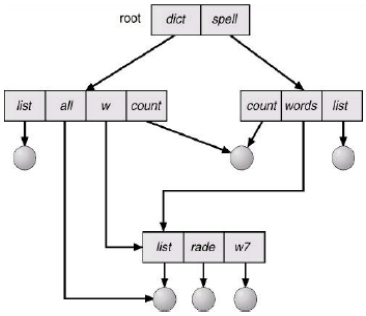
Now in order to work more like symbolic links the pathname is used first. This means that if a file is renamed and a new file with the old name is created both an alias and a symbolic link will find the same file (the new one).

If the pathname does not find the file but the identifier does then the pathname is updated to the current correct value.

Cycles in the directory graph

If we allow directories to appear in multiple places in the file system we can get cycles.

We don't want to fall into infinite loops if we traverse the file system e.g. to search for a file.



We need some way of uniquely identifying directories (the name is not enough, different names point to the same directory).

In UNIX directories can only be linked with symbolic links and algorithms count symbolic links on directories to stop infinite recursion.

Deletion of linked files

With hard links we maintain a count of the number of links. This gets decremented each time one of the links gets deleted. When this finally reaches zero the actual file is deleted.

With symbolic links if a link is deleted we do nothing. If the real file is deleted …

… we could have dangling pointers

… or we could maintain a list of all linked files and go around and delete all of them.

So what is in a directory entry?

The file name – we scan the directory to see if the file exists within it.

We can have the file attributes stored in the directory entry.

At a minimum we need a pointer to the file attributes and location information.

UNIX keeps the file attributes and location information of each file in a separate structure – the inode (Information node or Index node). The inode also keeps a count of the number of *hard* links to the file.

The inode table is an array of inodes stored in one or more places on the disk.

So a UNIX directory isn't much more than a table of names and corresponding inode numbers.

NTFS directory entries

All file and folder information is stored in the MFT (Master File Table).

Each file has at least one file record consisting of the attributes in Lecture 16, see the picture below.

Folders have an indexed table of file information.

Each folder entry includes the file name, a pointer to the file's MFT entry, plus the most commonly referenced file attributes e.g. created and modified dates, and length.

So much of the information in the files MFT record is duplicated in the directory entry. Why?

This explains the hardlink behaviour on slide 6.

An MFT entry for a file or directory:

Before next time

Read from the textbook

12.4 – Allocation methods

12.5 – Free-space management

12.6.1 - Efficiency