COMP 2718: The File System: Part 1

By: Dr. Andrew Vardy

Adapted from the notes of Dr. Rod Byrne

Outline

- Storage Media
- Block Storage
- Inodes
- Hierarchical directory structure
- File System Navigation — Chapter 2 of TLCL
- Pathnames
- Relative pathnames
- Going home
- References

Storage Media

Files are stored on a wide variety of non-volatile media:

- Magnetic storage devices (hard disk drives, floppy disks, tapes,. . .)
- Flash memory (solid state drives, thumb drives, . . .)
- Optical disks (read-only)

Non-volatile implies that the information is retained even if the device is unpoowered.

Non-volatile memory is typically slower and more expensive per byte than volatile memory (i.e. RAM). Non-volatile memory is also referred to as **secondary memory** while volatile storage is considered **primary memory**. The OS loads files from secondary memory into primary memory in order to use them.

Storage media vary in the number of times information can be written, the speed of access, and in how they are organized:

Media	Write Limit	Access	Organization
Disk	None	Faster	Cylinders
Tape	None	Slowest	Linear blocks
SSD	Limited	Fastest	Block, random
DVD	Limited	Slow	Block, cylinder

For recent cost estimates, see this article.

Block Storage

Most operating systems deal with stored data (files or otherwise) in fixed-size blocks. For example, the default block size on Linux and Mac OS X is 4 kB. Traditionally, the block size was 512 bytes but it has grown with available disk sizes.

The purpose of block storage is to try and keep the file's contents located together on the disk. This is tricky because it is hard to predict which files will grow and by how much. The disadvantage is the wasted storage:

- A 1-byte file will occupy 4 kB
- A 4097-byte file (4kB + 1) will occupy 8kB

WF: Use `ls -s` to determine the number of "blocks" used by files

Note that the values returned are multiples of 512 bytes (not the actual block size).

Inodes

Every file on a UNIX-like file system has an associated **inode**. An inode is a data structure that includes a variety of needed file attributes (e.g. the file's owner, size, permissions, . . .). The inode also has pointers to the blocks that contain the file's contents.

Actually, the inode structure also contains. . .

- pointers-to-pointers to data blocks (for larger files)
- pointers-to-pointers-to-pointers to data blocks (for even larger files)
- pointers-to-pointers-to-pointers-to-pointers to data blocks (for stupendously large files!)

For FreeBSD UNIX with 4 kB block size, the maximum file size is 512 GB (Stallings 2011).

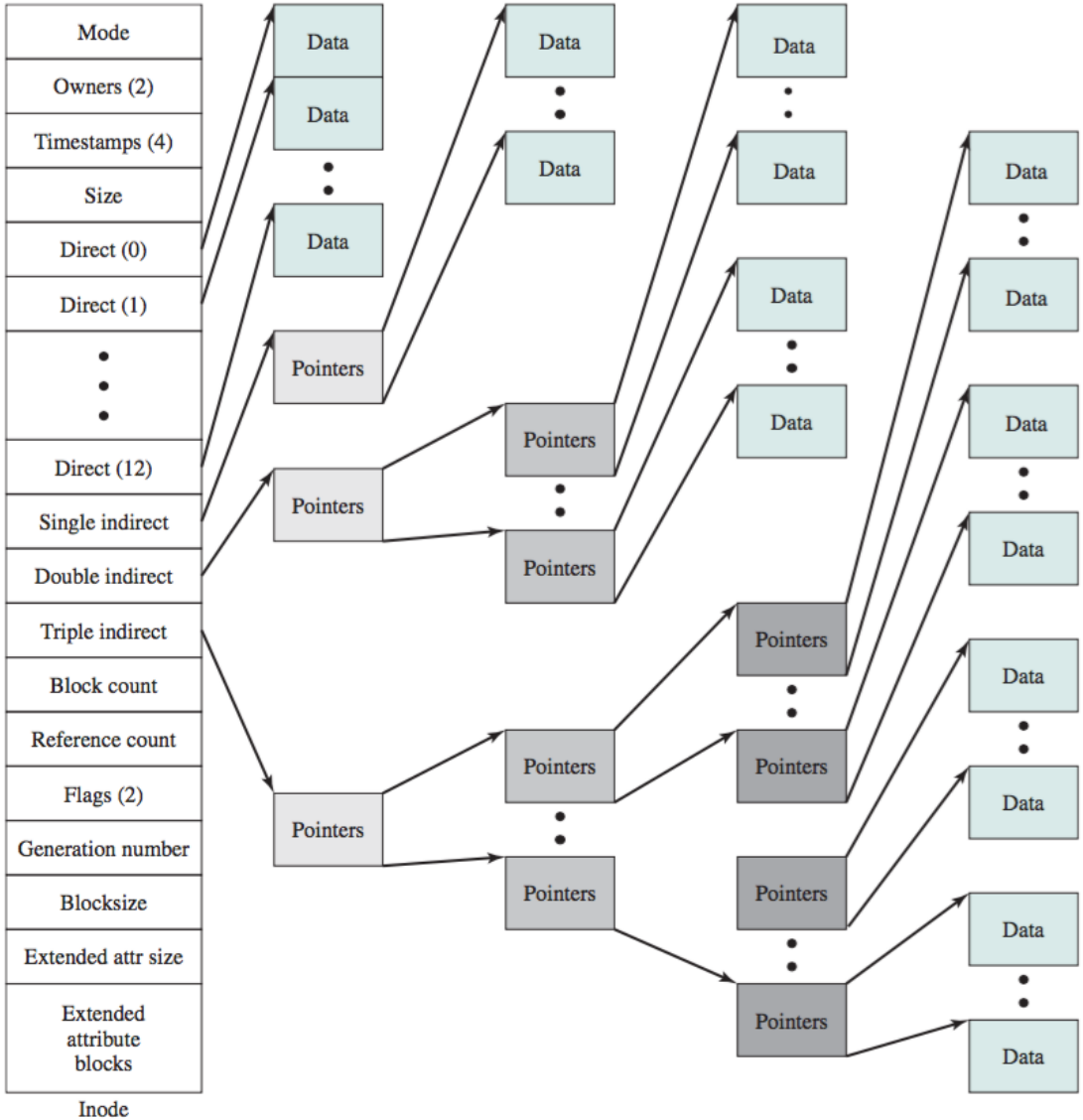

Figure 12.16 **Structure of FreeBSD Inode and File**

Figure 1: Chapter 12 (Stallings 2011)

Just like block size, the OS's use of inodes has consequences... It can run out of them!

WF: Check the number of inodes available with `df -i`

Try adding the `-h` argument to make the output easier to read.

What type of situation would cause us to run out of inodes?

A. Files which are too large?

B. Having too many files?

C. Having too many files with the same name?

D. Having too many files with the same name in the same directory?

Answer: B

Hierarchical directory structure

All modern file systems organize files hierarchically into directories (a.k.a folders) which contain files and other directories.

File systems can be viewed as trees, usually drawn upside-down. Like a real tree, a file system has a directory called the root or /.

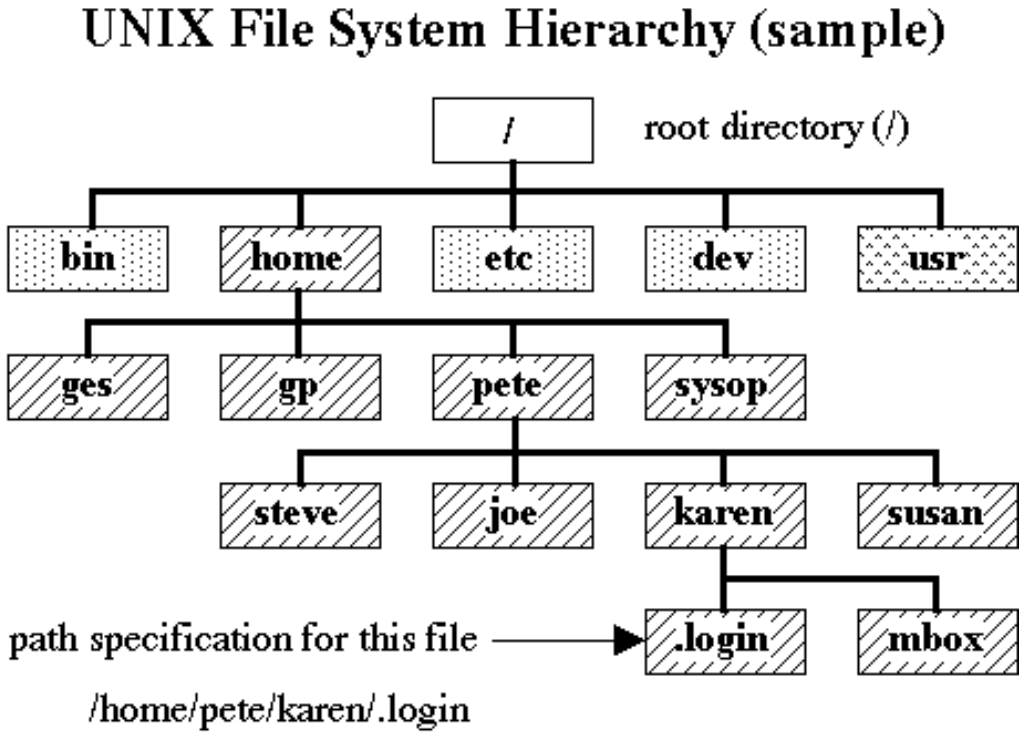

Figure 2: earthsci.stanford.edu/computing/unix

File System Navigation — Chapter 2 of TLCL

We've moved to material covered in chapter 2 of the textbook. The following commands will be introduced:

- pwd: Print name of the current working directory
- ls: List directory contents
- cd: Change directory

The **current working directory** can be dispayed with `pwd`.

When we first log in, the current working directory is the **home directory**. Every user has a home directory which is the only place a regular user can write files and directories.

Use `ls` without arguments to list the files and directories in the current working directory.

We can change directories with `cd`. The following are some common uses:

`cd`	Change to home directory
`cd /`	Change to root directory
`cd -`	Change to previous directory
`cd XYZ`	Change to directory XYZ (which must be in the current directory)
`cd /ABC/XYZ`	Change to directory /ABC/XYZ (current directory is irrelavent)

The last two are examples of using a **relative pathname** and an **absolute pathname**...

Pathnames

A file or directory is located by its **pathname** which is the list of directories starting from the root that enclose the file/directory *plus* the file/directory name itself. For example,

`/home/bob/.bashrc` The file `.bashrc` in the `bob` directory which is inside the `home` directory

`/usr/local/bin` The directory `bin` in the `local` directory which is inside the `usr` directory

The first `/` indicates the root directory. Each of the following `/`'s separates directory names. The rightmost entry is the file (e.g. `.bashrc`) or directory (e.g. `bin`) name.

Both of the above are **absolute pathnames** because they start with the root.

Relative pathnames

Every running program has a current directory. While an absolute pathname starts from the root directory, a relative pathname starts from the current directory. The following special symbols are used:

- `.` The current directory
- `..` The parent directory (one directory closer to the root)

We can illustrate these with the `cd` command and `pwd` commands:

```
$ pwd
/home/av

$ cd ..

$ pwd
/home

$ cd .

$ pwd
/home
```

Notice that `cd .` does nothing.

Here's a rather silly (but legal) relative pathname:

```
$ pwd
/home/av

$ cd ../bob/../av/web/.

$ pwd
/home/av/web
```

Could have just done `cd web`.

Going home

We saw that `cd` without arguments changes to the home directory (e.g. `/home/av`). We can also refer to our home directory with the special symbol `~`.

If we want to refer to another user's home directory we use `~user_name`.

Lets say the current directory is `/usr/share` but we want to list the contents of `/home/av/web/hosts`. We could do the following

```
ls ~/web/hosts
```

If there was a `web/hosts` directory in Bob's home directory, we could do this:

```
ls ~bob/web/hosts
```

References

Stallings, William. 2011. *Operating Systems - Internals and Design Principles (7th Ed.)*. Pitman.