UFS & FFS

The original UNIX File system (UFS) pioneered many of the concepts that are wide-spread in filesystems today. UFS allowed files to contain any number of bytes, rather than forcing the file to be blocked into records. UFS was also one of the very first tree-structured filesystems. Instead of having several drives or volumes, each with its own set of directories, UFS introduced the concept of having a master directory called the *root* directory. This directory, in turn, can contain other directories of files.

UNIX and the UFS introduced the concept that *"everything is a file"* – logical devices (/dev/tty), sockets, and other sorts of operating system structures were represented in a filesystem by special files rather than given different naming conventions and semantics.

Finally, UNIX introduced a simple set of function calls, an API, for accessing the contents of files: *open()* for opening a file, *read()* for reading a file's contents, *close()* for closing the file, etc. This API and its associated behavior are part of the POSIX standard specification.

Personnel at the University of California at Berkeley created an improved version of UFS that they named the Fast File System (FFS). Besides being faster and somewhat more robust, FFS had two important innovations: it allowed for long file names, and it introduced the concept of a symbolic link – a file that could point to another file. FFS was such an improvement over the original UFS that AT&T eventually abandoned its filesystem in favor of FFS.

File Contents

UNIX files are an unstructured collection of zero or more bytes of information. A file might contain an email message, a word processor document, an image, or anything else that can be represented as a stream of digital information. In principle, files can be any size, from zero bits to multiple terabytes of data.

Most of the information that you store on a UNIX system is stored as the contents of files. Even database systems such as Oracle or MySQL ultimately store their information as the contents of files.

Inodes

For each set of file contents in the filesystem, UNIX stores administrative information in a structure known as an **inode**, index node. **Inodes** reside on disk and do not have names. Instead they have indices, numbers, indicating their position in the array of inodes on each logical disk.

Each inode on a UNIX system contains:

- The location of the item's contents on the disk
- The item's type, e.g. file, directory, symbolic link
- The item's size, in bytes, if applicable

- The time the file's inode was last modified, typically at file creation, the *ctime*
- The time the file's contents were last modified, the *mtime*
- The time the file was last accessed, the *atime*, for *read(), exec(),* etc
- A reference count, which is the number of names the file has
- The file's owner, UID
- The file's group, GID
- The file's mode bits also called file permissions or permissions bits

The last three pieces of information, stored for each item and coupled with the UID/GID information about executing processes, are the fundamental data that UNIX uses for practically all local operating system security. Solaris along with other flavors of UNIX have extended the inode information to include access control lists.

Directories and Links

When you look at a directory, you see a list of files, the size of each file, and other kinds of information. UNIX directories are much simpler than this. A UNIX directory is nothing more than a list of names and inode numbers. These names are the names of files, directories, and other objects stored in the filesystem. Associated with each name is a numeric pointer that is actually an index on disk for an inode. An inode contains information about an individual entry in the filesystem.

Nothing else is contained in the directory other than names and inode numbers. No protection information is stored there, nor owner names, nor data. This information is all stored with the inode itself. The directory is a very simple relational database that maps names to inode numbers.

UNIX places no restrictions on how many names can point to the same inode. A directory may have 2, 5, or 50 names that each have the same inode number. In like manner, several directories may have names that associate to the same inode. These names are known as *links* or *hard links* to the file. We will talk about symbolic links later. The ability to have hard links is peculiar for the UNIX environment. No matter which hard link was created first, all links to a file are equal.

e.g.

```
ln  <existing file>  <new file>

$  ln  /etc/passwd  grossman
```

The above would create a hard link from grossman in the current working directory to /etc/passwd. Every time that that grossman is referred to, it is really the /etc/passwd file

Because of the way that links are implemented, you do not actually delete a file with the command **rm**. Instead you **unlink** the name – you sever the connection between the filename in a directory and the inode number. If another link still exists, the file will continue to exist on disk. After the last link is removed and the file is closed, the kernel will normally reclaim the storage because there is no longer a method for a user to access it. Internally, each inode maintains a reference count which is the count of how many filenames are linked to the inode. The **rm** command unlinks a filename and reduces the inode's reference count by 1. When the reference count reaches zero, the file is no longer accessible by name.

e.g.

unlink <file name>

```
$ unlink  grossman
```

This would remove grossman from the current working directory.

Every directory has two special names that are always present unless the filesystem is damaged. One entry is ".", and this is associated with the inode for the directory itself. It is self-referential. The second entry is for "..", which points to the parent of the current directory. This is the directory that is next closest to the root in the tree-structured filesystem. Because the root directory does not have a parent directory, in the root directory, the "." directory and the ".." directories are links to the same directory – the root directory.