UNIX FILE PERMISSIONS: A FIRST CONTACT

Mode data structure

In Unix every user has a unique *username* and is member of at least one *group*. The username and its primary group are held in the file /etc/passwd, and the secondary groups in /etc/group. Only the administrator can create new groups or add/remove group members.

Every file f has a file descriptor, called *i-node*, which contains all information needed to manage f by the system, such as the *owner* and an associated *group* among others. It also has a set of *permission flags* which is stored in a structured data type of 16 bits, called *mode*. This structure is organized as follows, cf. Fig. 1:

- bits 0–8 : access mode **rwx** (read, write, execute) for owner, group and others (world).
- bits 9–11 : 3 special bits **sst** resp. for owner (**setuid**), group (**setgid**) and others (**sticky**).
- bits 12–15 define the **file type**: ***regular file*** (text or executable), ***directory***, and some others.

				0	0	0	1	1	1	1	0	1	0	0	1
				s	s	t	r	w	x	r	w	x	r	w	x
file type				special			owner			group			others		

Fig. 1. Mode data stucture: permission bits

For regular files **rwx** meaning is obvious. Its also obvious for directories if you know that ***directories*** are data files that hold two pieces of information for each file within: the *file name* and its *inode number*. ***Read*** permission is needed to access the names of files in a directory and ***execute*** (aka ***search***) permission is needed to access the inodes of files in a directory, more precisely:

rwx bits on directories

- ***r-bit:*** grants the ability to read the names of files in the directory (but not retrieve any other information about them, including file type, size, ownership, permissions, etc.)
- ***w-bit:*** grants the ability to modify entries in the directory (e.g. add, rename, delete files). However, w permission is not required in order to modify a file listed in the directory.
- ***x-bit:*** grants the ability to *traverse* its tree in order to access files or subdirectories according to their access rights, but not see files inside the directory (unless the r-bit is set).

Remark 1. The access rights are not inherited: a file F in a directory D is accessed with F's access rights. More precisely, the content of F can be read or modified even if D's rw-bits are off (assuming F's rw-bits are on); and conversely, the name of F can be read or the file F can be renamed or deleted, even if F's rw-bits are off (assuming D's rw-bits are on).

Remark 2. The permissions applied to a specific user are determined in logical precedence (owner > group > others). For example, a member of a group will have the permissions assigned to the group, regardless of those assigned to others (even if others have a more generous access permission; note that this latter case has in fact no sense).

setuid and setgid on executables

When an executable file has been given the ***setuid*** attribute, normal users on the system who have permission to execute this file gain the privileges of the user who owns the file. Similar, if ***setgid*** is on, normal users will gain the privileges of the group set up by the owner.

setuid and setgid on directories

Setting the ***setgid*** permission on a directory causes new files and subdirectories created within it to **inherit its group ID**, rather than the primary group ID of the user who created the file.

setuid is ignored on Unix and Linux systems. But on others, like FreeBSD, its semantics is similar to setgid, namely, to force all new files and sub-directories to be owned by the top directory owner.

sticky on directories

When ***sticky*** bit is set on a directory, only *directory's owner* can rename or delete files in it. Without the sticky bit set, any user with wx permissions for this directory can rename or delete contained files, regardless of owner.

Long listing command `ls -l`

The mode field of a file, say `foo`, can be displayed via the line command `ls -l foo` (assuming the current directory contains file `foo`) :

```
-rwxr-x--x   3   beat   staff   22964   7   oct   21:34   foo
```

This line contains 9 fields and is organized as follows :

- Field 1 : ***file type*** (**–** stands for regular files, **d** for directories, **l** for symbolic links, **c** for character devices (e.g. tty or printer; stored in /dev), **b** for block devices (e.g. disc or display; stored in /dev), and some others. It is followed by the 9 ***rwx*** access bits, here `-rwxr-x--x`
- Field 2 : number of ***hard links***, here `3`
- Fields 3–4 : ***owner*** and an associated ***group***, here `beat staff`
- Field 5 : ***file size*** (in bytes), here `22964`
- Fields 6–8 : ***date*** of last modification (format varies, but always 3 fields), here `7 oct 21:34`
- Field 9 : ***file name***, here `foo`

Once a special bit is assigned, you'll see that the associated x bit is replaced by s or S for owner and group, and t or T for others, depending on wether x is set or not. For example if in the above example we apply the command `chmod +t foo` we will have:

```
-rwxr-x--t   3   beat   staff   22964   7   oct   21:34   foo
```

Default file permission (umask)

Each user has a default set of permissions which apply to all files created by that user. This default can be changed by the user, cf. http://www.cyberciti.biz/tips/understanding-linux-unix-umask-value-usage.html

chmod command

chmod command allows to reset the special and rwx bits. Common examples are:

- `chmod 755 foo` : set foo's rwx bits to 111 101 101 and the special bits to 000
- `chmod 2755 foo` : dito, but the special bits are set to 010
- `chmod u+x foo` : give owner x permission, leaving all other permission flags alone
- `chmod g=u-w foo` : set the group bits equal to the user bits, but clear the group write bit.
- `chmod -R o+r .` : give others r access to directory '.', and everything inside of it (-R = recursive)
- `chmod g+rwxs mydir` : give full group rw access to directory mydir, also setting the set-groupID flag so that directories created inside it inherit the group
- `chmod u=rw, go= foo` : explicitly give user rw access, and revoke all group and others access

chgrp command

chgrp is the command to change the group of a file. Only the owner of a file can change its group, and can only change it to a group of which he is a member. Example:

- `chgrp -R exascale .` : change the ownership of directory '.' to group 'exascale' and everything inside of it (-R = recursive). The person issuing this command must own all the files or it will fail.

ACLs

Most modern Unix system support also ACLs (Access Control Lists), a better adapted method of defining permissions for accessing files on servers.

Exercises

1a. Propose a rwx combination that makes no sense on a regular file, and explain why.
1b. Setting all rwx bits off on a regular file or directory can make sense. Explain.

2. Indicate the values of the ***srwx bits*** on a directory and its regular and directory files acting as a :

   a. ***public whiteboard p:*** only p's owner can add or remove its files, but every one can read and execute them.
   b. ***group whiteboard g:*** only g's owner and group can read, modify and execute its files.
   c. ***mailbox m:*** every one can copy a file, but only m's owner can read, delete and execute its files.

3. Assume that a device containing a file system, e.g. a USB stick or an external disk, is attached to your machine: a) What type of file is mounted ?, b) What does it contain ?, c) Where is it mounted ?