

Discussion on Files (continued)

Ref: Chapter 3 of [HGS].

User and Group IDs:

- Each user is given an ID (integer) called `uid`. (All system programs use `uid` instead of the user's login name.)
- The special `uid 0` (zero) is given to the super user (root).
- Each user also belongs to at least one group. Each group has an ID (integer) called `gid`.
- When a file is created, the `uid` and the `gid` of the user who created the file are stored as part of the information regarding the file.

More on file modes:

- Recall: Permissions (modes) can be specified as octal values (for user, group and other).

- Symbolic form also possible (constants defined in `<sys/stat.h>`):

<code>S_IRUSR</code>	<code>S_IWUSR</code>	<code>S_IXUSR</code>	<code>S_IRWXU</code>
<code>S_IRGRP</code>	<code>S_IWGRP</code>	<code>S_IXGRP</code>	<code>S_IRWXG</code>
<code>S_IROTH</code>	<code>S_IWOTH</code>	<code>S_IXOTH</code>	<code>S_IRWXO</code>

Sample code segment:

```
mode_t fmode;
int fd;
fmode = S_IRWXU | S_IRGRP | S_IROTH;
fd = open("file", O_WRONLY, fmode);
if (fd == -1) {
    fprintf("Error in open\n");
    exit(1);
}
```

Permission bits for executable files:

- Three extra permission bits, usually for executable files.
 - Set `uid` on execution:

`04000` `S_ISUID`

– Set gid on execution:

02000 S_ISGID

– Save-text-image (“sticky bit”):

01000 S_ISVTX

Purpose of set uid bit:

- If S_ISUID permission for a file is set, the **effective uid** of the process executing the file is the uid of the owner of the file (rather than the uid of the user who started the execution).
- The process is given file access privileges of the file owner.

Example: The passwd program.

- Program used to change a user’s password.
- Changing password requires permission to write to a file containing password information. (Usually, only root has permission to write to this file.)
- The executable file for the program is owned by root but has its S_ISUID permission set.

- When a normal user runs the program, the program is given the effective user id of the owner (root). So, entries in the password file can be modified.

Other examples: The following programs also (generally) have their S_ISUID permission set.

- Program to print files (usual name: lpr).
- Programs that update files containing top scorer information for some games.

Set gid: Similar to set uid.

Sticky bit:

- Original purpose: An executable with the sticky bit set is saved in the swap area of the disk (so that it can loaded into memory quickly).
- With virtual memory, this bit is redundant for executable files.

File creation mask:

- When a file is created, permissions specified by mode are always modified using the 9-bit quantity called umask.
- Generally, value of umask is specified in a start up file such as .cshrc.
- Value can be found using the shell command umask. (Usual default value: 022)
- Actual mode used for the file is given by

`given_mode & ~umask`

- Setting umask to 022 prevents the open system call from ever creating a file that can be written by anyone other than the owner.
- System call for umask:

```
mode_t umask (mode_t newmask);
```
- The call changes umask to the value specified by the parameter and returns the old value of umask.

- **Reading assignment:** Program example on page 45 of [HGS].

Additional comments regarding open:

- The call to open checks whether the specified access flags satisfy the file permissions.
- If not, the call returns -1 and errno is set to EACCES (“permission denied” error).
- Special flag O_EXCL prevents creation of file if the file already exists.
- If flag is given as

`O_WRONLY | O_CREAT | O_EXCL`

and the file exists, open fails and errno is set to EEXIST.

System call chmod:

```
int chmod (const char *pathname,  
           mode_t newmode);
```

- Can be used to change the permission bits of a file (including the special permission bits for executable files).
- The parameters represent the file name and the new permission bits respectively.
- The call succeeds only when issued by the owner of the file or the superuser (root).
- The value of `umask` is not used by `chmod`.
- Returns 0 if successful and -1 otherwise.

System call `chown`:

```
int  chown (const char *pathname,
            uid_t  newowner,
            gid_t  newgroup);
```

- Can be used to change the owner and the group of a file.
- The parameters represent the file name, the uid of the new owner and the gid of the new group respectively.

- If the `newowner` parameter is -1, the owner is not changed. A similar comment applied to the `newgroup` parameter.
- Caution: Once the owner is changed, the change cannot be undone by the previous owner.
- The call succeeds only when issued by the owner of the file or the superuser (root); otherwise, `errno` is set to `EPERM`.
- Returns 0 if successful and -1 otherwise.

Hard Links and Symbolic Links:

- Link: A mechanism that allows a file to be referred to by different names.
- Important: There is only one copy of the file; the links refer to the same copy.
- Useful in allowing different software teams to work together.
- Shell command for creating a hard link:

```
% ln  oldfile  secondname
```

- Shell command for creating a symbolic link:

```
% ln -s oldfile anothername
```

- Caution: If a file is deleted, the symbolic links for the file become dangling pointers.

Hard and Symbolic links: Differences

- A normal user cannot create a hard link to a directory; a user may create a symbolic link to a directory.
- A hard link must be to a file within the same file system. A soft link may go across file systems.
- The number of hard links to a file is stored as part of the information regarding the file. The system does not store information about the number of symbolic links.

System call link:

```
int link (const char *original_path,  
          const char *new_path);
```

- Creates a hard link to an existing file.
- The parameters represent the name of an existing file and that of the link respectively.
- The link may be in a different directory.
- Returns 0 if successful and -1 otherwise.

Some comments regarding unlink:

- The call to `unlink` only removes the link named in the call.
- The number of hard links to the file is decremented by 1.
- The disk space allocated to the file is reclaimed only if both of the following conditions hold:
 - (a) The hard link count has dropped to zero.
 - (b) No process has the file open.

System call symlink:

```
int  symlink (
        const char *real_name,
        const char *sym_name);
```

- Creates a symbolic link to an existing file.
- The parameters represent the name of an existing file and that of the symbolic link respectively.
- The link may be in a different file system.
- Returns 0 if successful and -1 otherwise.

System calls stat and fstat:

```
int  stat (const char *pathname,
           struct stat *buf);
int  fstat (int filedes,
           struct stat *buf);
```

- Allow us to obtain information about files.

- In both calls, the second parameter is a pointer to the buffer where information about the file can be stored. (The space for the buffer must have been allocated before the call.)
- stat: The first parameter represents the name of an existing file.
- fstat: The first parameter represents the file descriptor of an open file.
- Returns 0 if successful and -1 otherwise.

Data Members of struct stat: Handout 12.1.
(See also page 54 of [HGS].)

Program example: Handout 12.2. (Study also the example on pages 56–57 of [HGS].)