CptS 360 (System Programming)
Unit 6: Files and Directories

Bob Lewis

School of Engineering and Applied Sciences
Washington State University

Spring, 2022

Motivation

- Need to know your way around a filesystem.
- A properly organized filesystem is an aid to computing.
- Symbolic links can be very useful critters.

References

- Stevens & Rago Ch. 4

How Do I Find File "Metadata"?

- What *is* metadata?
- *stat(2)*, *lstat(2)*, and *fstat(2)*
  - all return the same information, just different ways of requesting it
- maintained in the file's `inode` (sketch)
- Consider this code snippet:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
...
struct stat *statBuf;
int st = stat("/home/bobl", statBuf);
```

Will this compile? Will it run? Always? If not, how could you make it better?

File Types

- regular file
- directory
- character device
- block device
- FIFO
- socket
- symbolic link

There are macros to identify all of these. (See *stat(2)*.) They act on the "`stat.st_mode`" field.

Finding "Unusual" Files

- try

  ```
  $ ls -l
  ```

  and

  ```
  $ ls --color
  ```

- remember *file(1)*
- Typical places for such files:
  - /tmp
  - /dev

  but these are just conventions.

What ID's are Attached to Files and to Processes?

- Files (and other filesystem objects) have one UID and one GID attached to them.
- Processes have at least six ID's associated with them:
  - real UID and GID
    - established at login
    - rarely changed
  - effective UID, GID, and supplementary GIDs
    - initially == real
    - may be reset by SUID programs
  - saved UID and GID
    - (discussed later)

File Access Permissions I

- Permission bits: (We already spoke of these.)
  - `rwxrwxrwx`
  - The only justification I know of for learning about octal numbers.
- To open any type of file by name requires execute (a.k.a. search) bit access of every directory (including "`.`", even implicitly) contained in the (full) path.
  - This is also necessary for seeking executables in `$PATH`.
- Read permission on a directory lets us read the directory (e.g. for an "`ls`"), execute/search lets us open a specific file.
  - Example: Setting permissions on your home directory.

File Access Permissions II

- Read and write permissions affect open flags `O_RDONLY`, `O_RDWR`, and `O_WRONLY`.
- The `O_TRUNC` flag requires (only) write permission. Logical, right?
- To create a new file in a directory requires write and execute permission on the directory.
- To delete a file, we also need write and execute permission on the directory containing the file.
  - We *don't* need read or write permission on the file itself.
- Execute permission is needed to execute the file, which must be a regular file.

How UIDs/GIDs Affect File Access

- If the EUID (of the *process*) is 0, anything goes.
- If the EUID is the file's owner,

  look at user id bit for appropriate access mode
- Else if the EGID (or a supplementary GID) matches the GID of the file,

  look at the group bit for appropriate access mode.
- Else,

  look at the other bit for the appropriate access mode.

Ownership of New Files (and Directories and ...)

- UID of file = EUID of process
- GID of file = EGID of process,
  - *unless* the "set-GID" bit of the containing directory is set, in which case it's the GID of the directory.
  - This was intended to allow groups to share directory contents.
  - Kind of anachronistic now.

What Can Your Process Do to a File?

access(2)

- Handy function to check accessibility.
- Note that it uses real UID and GID.
- Watch out for possible race condition with *chmod(2)* or *chown(2)*.
- Warning: It is of absolutely no use for the *finddups* PA.

How Can a Process Change Its Umask?

umask(2)

- (previously discussed)

How Can a Process Change a File's Protections?

chmod(2) and *fchmod(2)*

- This includes setting "sticky", set-UID, and set-GID bits.
- sticky bit semantics:
  - On a(n executable) file: vestigial and ignored.
  - On a directory: A file contained within can be renamed or deleted only by the owner of the file, the owner of the directory, and (of course) root.
- Best sticky bit example:
  - the `/tmp` directory: Everybody can write files to it, but not everybody should be able to rename files in or delete files from it.

Can I Change the Owner of a File?

chown(2), *fchown(2)*, and *lchown(2)*

- Can change owner *and* group, but -1 arguments leave that id unchanged.
- Only root can change owner and group of a file arbitrarily.
  - Figures, doesn't it?
- A file's owner can change its group to any other group he/she belongs to.

How Big is This File?

Use one of the *stat(2)* calls.

- `stat.st_size` is the length of the file in bytes.
  - $= 1 +$ offset of the last byte in the file
  - You *might* use this to read a whole file into a buffer.
  - There's a possible race condition. See it?
  - How can you overcome it?
- `stat.st_blocks` is the number of blocks allocated for the file.
- `stat.st_size` may be less than, equal to, or greater than `stat.st_blocks * stat.st_blksize`. Why?
  - block allocation
  - holes

How Do I Shorten a File?

truncate(2) and *ftruncate(2)*

- (see man page)

What is a Filesystem?

- Structure imposed by OS on any block device:
  - hard disk (most often)
  - CD-ROM or DVD-ROM
  - USB stick
  - floppy drive (remember those?)
- What are inodes? (discuss)
- Q: Where are filenames kept in the filesystem?
- Q: Where are paths kept in the filesystem?

How Do I Change the Name of a File?

rename(2)

- obvious purpose
- atomic
- This only works if `oldpath` and `newpath` are on the same filesystem (unlike *mv(1)*).

Can a File Have More Than One Name?

Yes, in two ways:

- ("hard") links
- symbolic links ("symlinks")

(Hard) Links

Two directory entries point to the same data blocks.

- A UNIX filesystem is really a DAG (Directed Acyclic Graph).
- *link(2)*
  - adds an entry to a directory
  - increments the inode's count
  - restrictions: regular files only, not across devices
- *unlink(2)*
  - removes an entry from a directory
  - decrements the inode's count
- Q: When is a file's data removed from the filesystem?

Symbolic Links

A directory entry refers to another file by name.

- *symlink(2)* and *readlink(2)*
- links to any file (directory, device, etc.) on the system
- *open(2)* opens the linked-at file (checking its permission bits)
- internal representation of a symlink:
  - its data (block) only contains the name of the thing being linked to
  - `stat.st_size` of a symlink is `strlen(`*`linkName`*`) + 1` (The "`+ 1`" is for NUL-termination.)
- does not affect link count (so symlinks can be "broken")

What Timestamps are on a File?

Kept with the inode:

- `atime` (a.k.a. actime)
  - last time file was accessed (but not modified)
  - set by (e.g.)
    - *read(2)* (of > 0 bytes)
    - *execve(2)* (later)
    - *pipe(2)* (also later)
- `mtime` (a.k.a. modtime)
  - last time file was modified (but not accessed)
  - set by (e.g.)
    - *write(2)* (of > 0 bytes)
    - *mknod(2)*
    - *truncate(2)*
- ctime
  - last time file status was changed
  - set when inode info is modified (owner, group, etc.)

(We'll demonstrate these on a local – not Dropbox – directory.)

How Do I Change Timestamps?

utime(2) and *utimes(2)*

- allow modification of atime and mtime fields
- *utime(2)* resolution is 1 sec.
- *utimes(2)* resolution is 1 usec (!).
- *utimensat(2)* resolution is 1 nsec (!!).

How Do I Create and Remove Directories?

mkdir(2) and *rmdir(2)*

- just like shell commands
- `rmdir()` won't remove a non-empty directory

remove(3)

- general purpose "clobber anything"
- `unlink()`s a file
- `rmdir()`s a directory
- (still won't remove a non-empty directory)

How Can I Read Data About Directory Contents?

opendir(3), *readdir(3)*, *scandir(3)*, *seekdir(3)*, *telldir(3)*, *rewinddir(3)*, and *closedir(3)*

- `struct dirent`
- *readdir(2)* vs. *readdir(3)*
  "This is not the function you are interested in."
- Don't use low-level (i.e. (2)) directory access functions.
  (There are synonyms. Be sure to include the right header.)

What's My Current Directory and How Do I Change It?

getcwd(2), *chdir(2)*, and *fchdir(2)*

- (see man pages)
- This is actually an attribute of your process, not the filesystem.

Special Device Files

- device numbers
  - major
  - minor
  - use macros to extract them
- `stat.st_dev`
  - device number of a device containing a filesystem
  - device number denotes the filesystem (i.e. its driver)
- `stat.st_rdev`
  - device number of a character or block device file
  - device number denotes the device itself

stat.st_dev vs. stat.st_rdev

compare

```
$ stat /home/bobl
```

and

```
$ stat /dev/sda2
```

with

```
$ stat -f /dev/sda2
```

The former tells you about the file (`stat.st_dev`). The latter tells you about the filesystem. mounted on the raw device (`stat.st_rdev`).

How Can I Make Sure Stuff Has Been Written to Disk?

sync(2) and *fsync(2)*

- No special permission required. (Why not?)
- Use with discretion: potentially time-consuming.