CS631 - Advanced Programming in the UNIX Environment
–
Files and Directories

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
http://www.cs.stevens.edu/~jschauma/631/

Text Formatting 101

This is a very long text block contnaing sentences with no re al punctuation and no paragraphsText like this is hard to readAlsoreading such lo ng text is no fun, and when the instructor is not having fun when grading that might yield undesirable resluts for the studentEven though this is a programming classit is i mportant to be able to write English text well:programming is communicationand a ll communications requires clarity.The majority of the work in any given program depend s on clear communication of the requirementsof getting answers to questions and of wo rking together with other peopleThereforeyou should practice writing legigible and clear text that is easy to read and understandText should be broken at around 80 charac ters and paragraphs should be created every now and then to make it easier for the readerO therwise,you end up having just this very long text block containing sentences w ith no real punctuation and no paragraphsand as noted abovetext like that is hard to r eadno fun and may lead to a grumpy instructorwhen you really want your instructor t o be in a good mood when he is attempting to make sense of your homework submissions.

HW#1 - some stats

```
7 pwd
5 mkdir
5 echo
4 kill
4 cp
2 test
2 mv
2 cat
1 rm
1 ls
1 ln
1 date
1 chmod
```

HW#1 - some stats

```
$ wc -w * | sort -n | column
     41 HW1-a            625 HW1-h          1295 HW1-n
    239 HW1-b            632 HW1-g          1359 HW1-o
    451 HW1-c            647 hw1-i          1409 HW1-p
    489 HW1-d            742 HW1-j          1750 HW1-q
    494 HW1-e            861 HW1-k
    497 HW1-f            965 HW1-l
    524 HW1-g           1079 HW1-m
$
```

When reading code...

First understand *what* it does.

Then understand *why* it does it.

Only then pay attention to *how* it does it.

When reading code...

- compare source code and documentation; are they in sync?
- compare documentation and reality; are they in sync?
- review manual pages for system- and library calls made; are there (failure or success) cases unaccounted for?
- what follow-up questions do you have?

”More code” does not necessarily imply a ”better program”.

“One of my most productive days was throwing away 1,000 lines of code.”

Ken Thompson

Code Reading

HW#2

`stat(2)` family of functions

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *path, struct stat *sb);
int lstat(const char *path, struct stat *sb);
int fstat(int fd, struct stat *sb);
```

Returns: 0 if OK, -1 on error

All these functions return extended attributes about the referenced file (in the case of *symbolic links*, `lstat(2)` returns attributes of the *link*, others return stats of the referenced file).

`stat(2)` family of functions

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *path, struct stat *sb);
int lstat(const char *path, struct stat *sb);
int fstat(int fd, struct stat *sb);
```

Returns: 0 if OK, -1 on error

All these functions return extended attributes about the referenced file (in the case of *symbolic links*, `lstat(2)` returns attributes of the *link*, others return stats of the referenced file).

```
struct stat {
    dev_t     st_dev;         /* device number (filesystem) */
    ino_t     st_ino;         /* i-node number (serial number) */
    mode_t    st_mode;        /* file type & mode (permissions) */
    dev_t     st_rdev;        /* device number for special files */
    nlink_t   st_nlink;       /* number of links */
    uid_t     st_uid;         /* user ID of owner */
    gid_t     st_gid;         /* group ID of owner */
    off_t     st_size;        /* size in bytes, for regular files */
    time_t    st_atime;       /* time of last access */
    time_t    st_mtime;       /* time of last modification */
    time_t    st_ctime;       /* time of last file status change */
    long      st_blocks;      /* number of 512-byte* blocks allocated */
    long      st_blksize;     /* best I/O block size */
};
```

struct stat: st_mode

The `st_mode` field of the `struct stat` encodes the type of file:

- **regular** – most common, interpretation of data is up to application
- **directory** – contains names of other files and pointer to information on those files. Any process can read, only kernel can write.
- **character special** – used for certain types of devices
- **block special** – used for disk devices (typically). All devices are either *character* or *block special*.
- **FIFO** – used for interprocess communication (sometimes called *named pipe*)
- **socket** – used for network communication and non-network communication (same host).
- **symbolic link** – Points to another file.

Find out more in `<sys/stat.h>`.

struct stat: st_mode

```
$ cc -Wall still-simple-ls.c
$ ./a.out /dev | more
```

`struct stat`: `st_mode`, `st_uid` and `st_gid`

Every process has six or more IDs associated with it:

real user ID real group ID	who we really are
effective user ID effective group ID supplementary group IDs	used for file access permission checks
saved set-user-ID saved set-group-ID	saved by `exec` functions

Whenever a file is *setuid*, set the *effective user ID* to `st_uid`. Whenever a file is *setgid*, set the *effective group ID* to `st_gid`. `st_uid` and `st_gid` always specify the owner and group owner of a file, regardless of whether it is setuid/setgid.

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with O_RDONLY or O_RDWR, need read permission

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with O_RDONLY or O_RDWR, need read permission
- To open a file with O_WRONLY or O_RDWR, need write permission

struct stat: st_mode

`st_mode` also encodes the file access permissions (`S_IRUSR`, `S_IWUSR`, `S_IXUSR`, `S_IRGRP`, `S_IWGRP`, `S_IXGRP`, `S_IROTH`, `S_IWOTH`, `S_IXOTH`). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with `O_RDONLY` or `O_RDWR`, need read permission
- To open a file with `O_WRONLY` or `O_RDWR`, need write permission
- To use `O_TRUNC`, must have write permission

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with O_RDONLY or O_RDWR, need read permission
- To open a file with O_WRONLY or O_RDWR, need write permission
- To use O_TRUNC, must have write permission
- To create a new file, must have write+execute permission for the directory

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with O_RDONLY or O_RDWR, need read permission
- To open a file with O_WRONLY or O_RDWR, need write permission
- To use O_TRUNC, must have write permission
- To create a new file, must have write+execute permission for the directory
- To delete a file, need write+execute on directory, file doesn't matter

struct stat: st_mode

st_mode also encodes the file access permissions (S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH). Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
- To open a file with O_RDONLY or O_RDWR, need read permission
- To open a file with O_WRONLY or O_RDWR, need write permission
- To use O_TRUNC, must have write permission
- To create a new file, must have write+execute permission for the directory
- To delete a file, need write+execute on directory, file doesn't matter
- To execute a file (via exec family), need execute permission

access(2)

```
#include <unistd.h>

int access(const char *path, int mode);
```

Returns: 0 if OK, -1 on error

Tests file accessibility on the basis of the *real* uid and gid. Allows setuid/setgid programs to see if the real user could access the file without it having to drop permissions to do so.

The `mode` paramenter can be a bitwise OR of:

- `R_OK` – test for read permission
- `W_OK` – test for write permission
- `X_OK` – test for execute permission
- `F_OK` – test for existence of file

access(2)

```
$ cc -Wall access.c
$ ./a.out /etc/passwd
access ok for /etc/passwd
open ok for /etc/passwd
$ ./a.out /etc/master.passwd
access error for /etc/master.passwd
open error for /etc/master.passwd
$ sudo chown root a.out
$ sudo chmod 4755 a.out
$ ./a.out /etc/passwd
access ok for /etc/passwd
open ok for /etc/passwd
$ ./a.out /etc/master.passwd
access error for /etc/master.passwd
open ok for /etc/master.passwd
$
```

access(2)

On Mac OS X:

```
$ ls -l a.out
-rwsr-xr-x  1 root  staff  8732 Sep 15 22:35 a.out
$ cc -Wall access.c
ld: can't write output file: a.out for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
$ ls -l a.out
-rwsr-xr-x  1 root  staff  8732 Sep 15 22:35 a.out
$ gcc -Wall access.c
ld: can't write output file: a.out for architecture x86_64
collect2: ld returned 1 exit status
$ ls -l a.out
ls: a.out: No such file or directory
```

access(2)

```
$ sudo dtruss gcc -Wall access.c
[...]
10157/0x48d34:  access("a.out\0", 0x0, 0x0)              = 0 0
10157/0x48d34:  access("a.out\0", 0x2, 0x0)              = -1 Err#13
10157/0x48d34:  unlink("a.out\0", 0x0, 0x100365AF0)      = 0 0
[...]
$ ls -ld . a.out
$ rm -f a.out
```

access(2)

On NetBSD:

```
$ ls -l a.out
-rwsr-xr-x  1 root  users  9399 Sep 26 16:12 a.out
$ cc -Wall access.c
$ ls -l a.out
-rwxr-xr-x  1 jschauma  users  9399 Sep 26 16:51 a.out
$ ktruss cc -Wall access.c
[...]
 14250      1 ld       CALL  __stat30(0x43a1bd,0x7f7fffffd340)
 14250      1 ld       NAMI  "a.out"
 14250      1 ld       CALL  unlink(0x43a1bd)
 14250      1 ld       NAMI  "a.out"
 14250      1 ld       CALL  open(0x43a1bd,0x602,0x1b6)
 14250      1 ld       NAMI  "a.out"
[...]
```

access(2)

On Linux:

```
$ ls -l a.out
-rwxr-xr-x 1 root users 6555 Sep 24 20:35 a.out
$ cc -Wall access.c
$ ls -l a.out
-rwxr-xr-x 1 jschauma users 6555 Sep 24 20:36 a.out
$ strace -f cc -Wall access.c
[...]
[pid 11721] stat("a.out", {st_mode=S_IFREG|0755, st_size=6555, ...}) = 0
[pid 11721] lstat("a.out", {st_mode=S_IFREG|0755, st_size=6555, ...}) = 0
[pid 11721] unlink("a.out")              = 0
[pid 11721] open("a.out", O_RDWR|O_CREAT|O_TRUNC, 0666) = 3
[...]
```

`struct stat`: `st_mode`

Which permission set to use is determined (in order listed):

1. If effective-uid == 0, grant access

struct stat: st_mode

Which permission set to use is determined (in order listed):

1. If effective-uid == 0, grant access
2. If effective-uid == `st_uid`
   2.1. if appropriate user permission bit is set, grant access
   2.2. else, deny access

`struct stat`: `st_mode`

Which permission set to use is determined (in order listed):

1. If effective-uid == 0, grant access
2. If effective-uid == `st_uid`
   2.1. if appropriate user permission bit is set, grant access
   2.2. else, deny access
3. If effective-gid == `st_gid`
   3.1. if appropriate group permission bit is set, grant access
   3.2. else, deny access

struct stat: st_mode

Which permission set to use is determined (in order listed):

1. If effective-uid == 0, grant access
2. If effective-uid == `st_uid`
   2.1. if appropriate user permission bit is set, grant access
   2.2. else, deny access
3. If effective-gid == `st_gid`
   3.1. if appropriate group permission bit is set, grant access
   3.2. else, deny access
4. If appropriate other permission bit is set, grant access, else deny access

struct stat: st_mode

Ownership of new files and directories:

- st_uid = effective-uid
- st_gid = ...either:
  - effective-gid of process
  - gid of directory in which it is being created

umask(2)

```
#include <sys/stat.h>

mode_t umask(mode_t numask);
```

Returns: previous file mode creation mask

`umask(2)` sets the file creation mode mask. Any bits that are *on* in the file creation mask are turned *off* in the file's mode.

Important because a user can set a default umask. If a program needs to be able to insure certain permissions on a file, it may need to turn off (or modify) the umask, which affects only the current process.

umask(2)

```
$ cc -Wall umask.c
$ umask 022
$ touch foo
$ ./a.out
$ ls -l foo*
-rw-r--r--  1 jschauma  staff  0 Sep 26 18:35 foo
-rw-r--r--  1 jschauma  staff  0 Sep 26 18:36 foo1
-rw-rw-rw-  1 jschauma  staff  0 Sep 26 18:36 foo2
-rw-------  1 jschauma  staff  0 Sep 26 18:36 foo3
```

chmod(2), lchmod(2) and fchmod(2)

```
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
int lchmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
```

Returns: 0 if OK, -1 on error

Changes the permission bits on the file. Must be either superuser or *effective uid* == `st_uid`. *mode* can be any of the bits from our discussion of `st_mode` as well as:

- `S_ISUID` – setuid
- `S_ISGID` – setgid
- `S_ISVTX` – sticky bit (aka “saved text”)
- `S_IRWXU` – user read, write and execute
- `S_IRWXG` – group read, write and execute
- `S_IRWXO` – other read, write and execute

chmod(2), lchmod(2) and fchmod(2)

```
$ rm foo*
$ umask 077
$ touch foo foo1
$ chmod a+rx foo
$ ls -l foo*
-rwxr-xr-x  1 jschaumann  staff  0 Sep 15 23:00 foo
-rw-------  1 jschaumann  staff  0 Sep 15 23:00 foo1
$ cc -Wall chmod.c
$ ./a.out
$ ls -l foo foo1
-rwsr--r-x  1 jschaumann  staff  0 Sep 15 23:01 foo
-rw-r--r--  1 jschaumann  staff  0 Sep 15 23:01 foo1
$
```

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

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
```

Returns: 0 if OK, -1 on error

Changes `st_uid` and `st_gid` for a file. For BSD, must be superuser. Some SVR4's let users chown files they own. POSIX.1 allows either depending on `_POSIX_CHOWN_RESTRICTED` (a kernel constant).

owner or *group* can be -1 to indicate that it should remain the same. Non-superusers can change the `st_gid` field if both:

- effective-user ID == `st_uid` and
- *owner* == file's user ID and *group* == effective-group ID (or one of the supplementary group IDs)

`chown` and friends clear all setuid or setgid bits.

Directory sizes (on a system using UFS)

```
$ cd /tmp
$ mkdir -p /tmp/d
$ ls -ld /tmp/d
drwxr-xr-x  2 jschauma  wheel  512 Sep 26 19:35 /tmp/d
$ touch d/a d/b d/c d/d d/e d/f d/g
$ ls -ld /tmp/d
drwxr-xr-x  2 jschauma  wheel  512 Sep 26 19:35 /tmp/d
$ touch d/$(jot -b a 255 | tr -d '[:space:]')
$ ls -ld /tmp/d
drwxr-xr-x  2 jschauma  wheel  512 Sep 26 19:35 /tmp/d
$ touch d/$(jot -b b 255 | tr -d '[:space:]')
$ ls -ld /tmp/d
drwxr-xr-x  2 jschauma  wheel  1024 Sep 26 19:37 /tmp/d
$ rm /tmp/d/a*
$ ls -ld /tmp/d
drwxr-xr-x  2 jschauma  wheel  1024 Sep 26 19:37 /tmp/d
$
```

Directory sizes (on a system using HFS+)

```
$ cd /tmp
$ mkdir -p /tmp/d
$ cd /tmp/d
$ ls -ld
drwxr-xr-x  2 jschauma  wheel  68 Sep 24 18:52 .
$ touch a
$ ls -ld
drwxr-xr-x  3 jschauma  wheel  102 Sep 24 18:52 .
$ echo $((102 / 3))
34
$ touch c
$ ls -ld
drwxr-xr-x  4 jschauma  wheel  136 Sep 24 18:52 .
$ rm c
$ ls -ld
drwxr-xr-x  3 jschauma  wheel  102 Sep 24 18:52 .
$
```

Homework

Reading:

- manual pages for the functions covered
- Stevens Chap. 4.1 through 4.13

Playing:

- in your shell, set your umask to various values and see what happens to new files you create (example: Stevens # 4.3)
- Verify that turning off user-read permission for a file that you own denies you access to the file, even if group- or other permissions allow reading.

Midterm Assignment:

`http://www.cs.stevens.edu/~jschauma/631/f13-midterm.html`