System Programming

Prof. Chuan-Ju Wang
Dept. of Computer Science
University of Taipei

Files and Directories

`stat(2)` Family of Functions

- 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.

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

int stat(const char *restrict pathname, struct
➥ stat *restrict buf);

int fstat(int filedes, struct stat *buf);

int lstat(const char *restrict pathname, struct
➥ stat *restrict buf);
```

All three return: 0 if OK, 1 on error

* http://en.wikipedia.org/wiki/Restrict

`stat(2)` Family of Functions

```
struct stat {
  mode_t    st_mode;      /* file type & mode (permissions) */
  ino_t     st_ino;       /* i-node number (serial number) */
  dev_t     st_dev;       /* device number (file system) */
  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 */
  blksize_t st_blksize;   /* best I/O block size */
  blkcnt_t  st_blocks;    /* number of disk blocks allocated */
};
```

Figure 2.20. Some common primitive system data types

Type	Description
caddr_t	core address (Section 14.9)
clock_t	counter of clock ticks (process time) (Section 1.10)
comp_t	compressed clock ticks (Section 8.14)
dev_t	device numbers (major and minor) (Section 4.23)
fd_set	file descriptor sets (Section 14.5.1)
fpos_t	file position (Section 5.10)
gid_t	numeric group IDs
ino_t	i-node numbers (Section 4.14)
mode_t	file type, file creation mode (Section 4.5)
nlink_t	link counts for directory entries (Section 4.14)
off_t	file sizes and offsets (signed) (lseek, Section 3.6)
pid_t	process IDs and process group IDs (signed) (Sections 8.2 and 9.4)
ptrdiff_t	result of subtracting two pointers (signed)

Each member is specified by a primitive system data type (see Steven Section 2.8).

File Types

- Encoded in the `st_mode` member of the `stat` structure.
  1. Regular – most common, interpretation of data is up to application
  2. Directory – contains names of other files and pointer to information on those files. Any process can read, only kernel can write.
  3. Character special – used for certain types of devices
  4. Block special – used for disk devices (typically). All devices are either character or block special.
  5. FIFO – used for interprocess communication (sometimes called named pipe)
  6. Socket – used for network communication and non-network communication (same host).
  7. Symbolic link – Points to another file.

The file type can be determined with the macros:

Figure 4.1. File type macros in <sys/stat.h>

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link
S_ISSOCK()	socket

stat.h = (/usr/include/i386-linux-gnu/sys) - VIM

stat.h

```
126
127 /* Test macros for file types.  */
128
129 #define __S_ISTYPE(mode, mask)  (((mode) & __S_IFMT) == (mask))
130
131 #define S_ISDIR(mode)    __S_ISTYPE((mode), __S_IFDIR)
132 #define S_ISCHR(mode)    __S_ISTYPE((mode), __S_IFCHR)
133 #define S_ISBLK(mode)    __S_ISTYPE((mode), __S_IFBLK)
134 #define S_ISREG(mode)    __S_ISTYPE((mode), __S_IFREG)
135 #ifdef __S_IFIFO
136 # define S_ISFIFO(mode)  __S_ISTYPE((mode), __S_IFIFO)
137 #endif
138 #ifdef __S_IFLNK
139 # define S_ISLNK(mode)   __S_ISTYPE((mode), __S_IFLNK)
140 #endif
141
142 #if defined __USE_BSD && !defined __S_IFLNK
143 # define S_ISLNK(mode)  0
144 #endif
```

Find out more in /usr/include/i386-linux-gnu/sys/stat.h

Figure 4.3. Print type of file for each command-line argument

```
#include "apue.h"

int
main(int argc, char *argv[])
{

    int         i;
    struct stat buf;
    char        *ptr;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            err_ret("lstat error");
            continue;

        }
        if (S_ISREG(buf.st_mode))
            ptr = "regular";
        else if (S_ISDIR(buf.st_mode))
            ptr = "directory";
        else if (S_ISCHR(buf.st_mode))
            ptr = "character special";
        else if (S_ISBLK(buf.st_mode))
            ptr = "block special";
        else if (S_ISFIFO(buf.st_mode))
            ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))
            ptr = "symbolic link";
        else if (S_ISSOCK(buf.st_mode))
            ptr = "socket";
        else
            ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }
    exit(0);
}
```

- Define `_GNU_SOURCE` to include the definition of the `S_ISSOCK` macro.
  - glibc* does not make the GNU extensions available automatically.
  - If a program depends on GNU extensions or some other non-standard functionality, it is necessary to
    (1) compile it with the C compiler option -D_GNU_SOURCE
    (2) put #define _GNU_SOURCE at the beginning of your source files, before any C library header files are included.

```
$ ./a.out /etc/passwd /etc /dev/initctl /dev/log /dev/tty \
> /dev/scsi/host0/bus0/target0/lun0/cd /dev/cdrom
/etc/passwd: regular
/etc: directory
/dev/initctl: fifo
/dev/log: socket
/dev/tty: character special
/dev/scsi/host0/bus0/target0/lun0/cd: block special
/dev/cdrom: symbolic link
```

* GLIBC, the GNU C Library: http://www.gnu.org/software/libc/

`mkdir(2)` and `rmdir(2)`

- Directories are created with the `mkdir` function and deleted with the `rmdir`.

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

int mkdir(const char *pathname, mode_t mode);

                                Returns: 0 if OK, 1 on error
```

- Creates a new, empty (except for . and .. entries) directory.
- Access permissions specified by *mode*.

```
#include <unistd.h>

int rmdir(const char *pathname);

                                Returns: 0 if OK, 1 on error
```

- If the link count is 0 (after this call), and no other process has the directory open, directory is removed.
- Directory must be empty (only . and .. remaining)

Reading Directories

```
#include <dirent.h>

DIR *opendir(const char *pathname);

        Returns: pointer if OK, NULL on error
struct dirent *readdir(DIR *dp);

        Returns: pointer if OK, NULL at end of
                                directory or error
void rewinddir(DIR *dp);

int closedir(DIR *dp);

                Returns: 0 if OK, 1 on error
```

- `rewinddir` resets an open directory to the beginning so `readdir` will again return the first entry.

```
struct dirent {
  ino_t d_ino;                  /* i-node number */
  char  d_name[NAME_MAX + 1];   /* null-terminated filename */
}
```

Moving Around Directories

- Get the kernel's idea of our process's current working directory.

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);


                Returns: buf if OK, NULL on error
```

Moving Around Directories

- Allows a process to change its current working directory.
- Note that `chdir` and `fchdir` affect only the current process.

```
#include <unistd.h>

int chdir(const char *pathname);

int fchdir(int filedes);
```

Both return: 0 if OK, 1 on error

Figure 4.23. Example of `chdir` function

```
#include "apue.h"

int
main(void)
{

      if (chdir("/tmp") < 0)
          err_sys("chdir failed");
      printf("chdir to /tmp succeeded\n");
      exit(0);
}
```

```
jere@VB(MBA) [~/SystemProgramming/apue.2e/test] make fig4-23
gcc -DLINUX -ansi -I~/SystemProgramming/apue.2e/include -Wall -D_GNU_SOURCE    -L
jere@VB(MBA) [~/SystemProgramming/apue.2e/test] pwd
/home/jere/SystemProgramming/apue.2e/test
jere@VB(MBA) [~/SystemProgramming/apue.2e/test] ./fig4-23
chdir to /tmp succeeded
jere@VB(MBA) [~/SystemProgramming/apue.2e/test] pwd
/home/jere/SystemProgramming/apue.2e/test
```

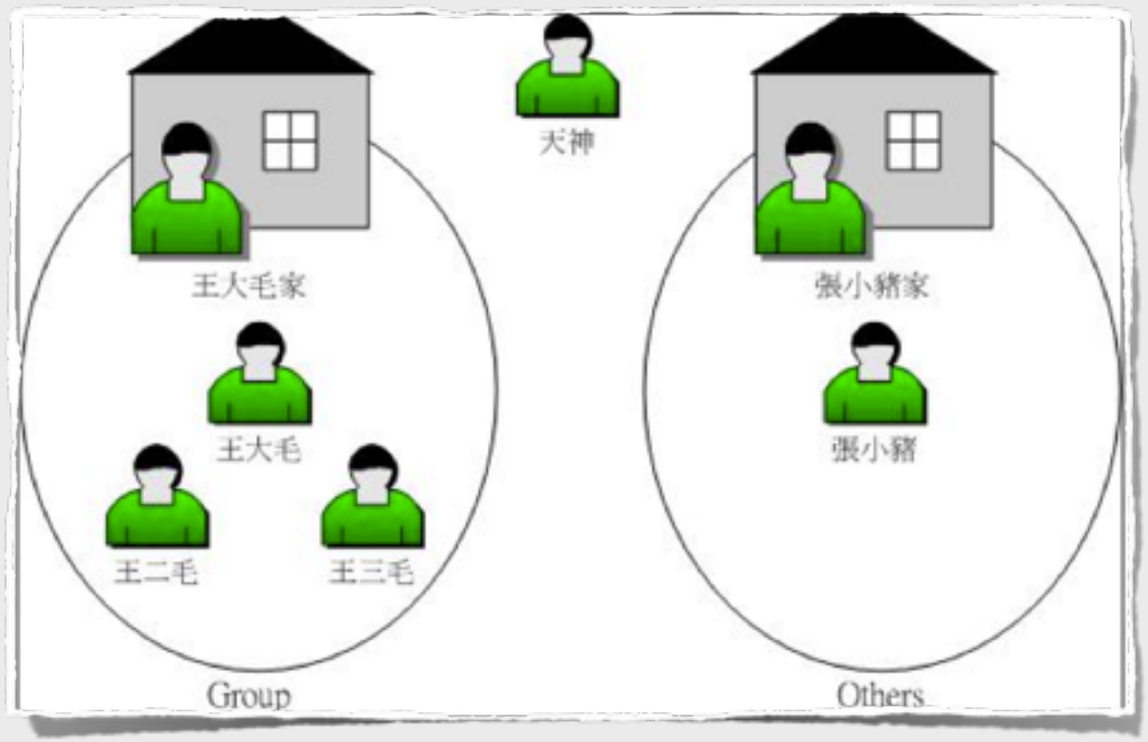
Figure 4.24. Example of `getcwd` function

```
#include "apue.h"

int
main(void)
{

    char    *ptr;
    int     size;

    if (chdir("/usr/spool/uucppublic") < 0)
        err_sys("chdir failed");

    ptr = path_alloc(&size); /* our own function */
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd failed");

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

```
$ ./a.out
cwd = /var/spool/uucppublic
$ ls -l /usr/spool
lrwxrwxrwx 1 root 12 Jan 31 07:57 /usr/spool -> ../var/spool
```

User/Group/Others

- Every file has an owner and a group owner.
- The owner is specified by the `st_uid` member of the stat structure; the group owner, by the `st_gid` member.*




```
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -al
total 108
drwxrwxr-x  2 jere jere  4096 2012-01-30 22:25 .
drwxr-xr-x 33 jere jere  4096 2012-01-26 17:24 ..
-rwxrwxr-x  1 jere jere  7359 2012-01-25 19:43 fig1-8
-rw-r--r--  1 jere jere   172 2012-01-25 19:43 fig1-8.c
-rwxrwxr-x  1 jere jere  7233 2012-01-26 18:06 fig3-1
-rwxrwxr-x  1 jere jere  7904 2012-01-29 16:33 fig3-10
-rw-r--r--  1 jere jere   783 2012-01-29 16:32 fig3-10.c
-rw-r--r--  1 jere jere   149 2012-01-26 17:10 fig3-1.c
-rwxrwxr-x  1 jere jere  7930 2012-01-26 18:09 fig3-2
-rw-r--r--  1 jere jere   478 2012-01-26 18:09 fig3-2.c
-rwxrwxr-x  1 jere jere  7826 2012-01-28 00:05 fig3-4
-rw-r--r--  1 jere jere   262 2012-01-30 22:23 fig3-4.c
-rw-rw-r--  1 jere jere   399 2012-01-28 00:04 fig3-4-test.sh
-rwxrwxr-x  1 jere jere  7913 2012-01-30 22:23 fig4-3
-rw-r--r--  1 jere jere   695 2012-01-30 22:23 fig4-3.c
-rw-r--r--  1 jere jere   676 2012-01-09 22:20 Makefile
-rw-rw-r--  1 jere jere    11 2012-01-29 16:33 temp.foo
-rw-rw-r--  1 jere jere   220 2012-01-28 00:03 test.sh
```

See `/etc/passwd`, `/etc/shadow`, `/etc/group`

* http://linux.vbird.org/linux_basic/0210filepermission.php

Set-User-ID and Set-Group-ID

- Every process has six or more IDs associated with it.

Figure 4.5. User IDs and group IDs associated with each process

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

Real user/group ID: taken from our entry in the password file when we log in

Effective user/group ID: determine our file access permissions

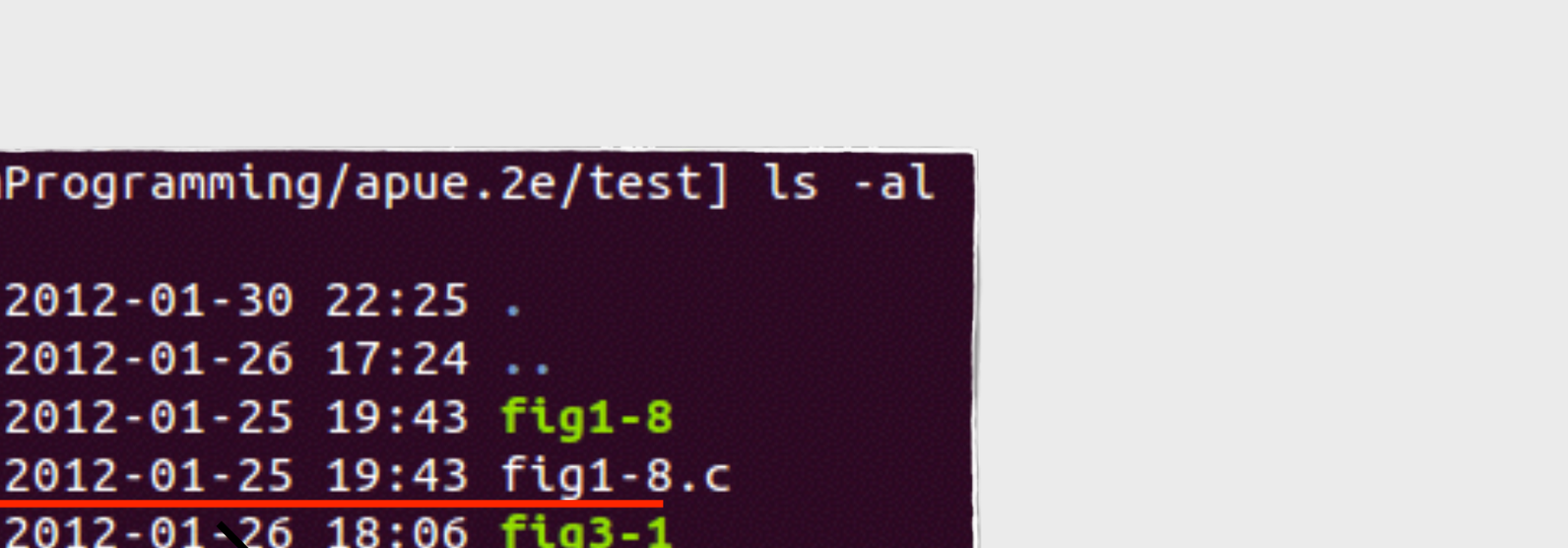
Saved set-user-ID and saved set-group-ID: copies of the effective user ID and the effective group ID when a program is executed (described in Steven Section 8.11).

Set-User-ID and Set-Group-ID

- Normally, the effective user ID equals the real user ID, and the effective group ID equals the real group ID.
- Set special flag in the file's mode word (`st_mode`)
  - When this file is executed, set the effective user/group ID of the process to be the owner of the file (`st_uid`/`st_gid`).
  - These two bits in the file's mode word are called the *set-user-ID* bit and the *set-group-ID* bit.
  - Example:
    - `passwd(1)`, is a set-user-ID program.

File Access Permissions

```
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -al
total 108
drwxrwxr-x  2 jere jere 4096 2012-01-30 22:25 .
drwxr-xr-x 33 jere jere 4096 2012-01-26 17:24 ..
-rwxrwxr-x  1 jere jere 7359 2012-01-25 19:43 fig1-8
-rw-r--r--  1 jere jere  172 2012-01-25 19:43 fig1-8.c
-rwxrwxr-x  1 jere jere 7233 2012-01-26 18:06 fig3-1
-rwxrwxr-x  1 jere jere 7904 2012-01-29 16:33 fig3-10
-rw-r--r--  1 jere jere  783 2012-01-29 16:32 fig3-10.c
-rw-r--r--  1 jere jere  149 2012-01-26 17:10 fig3-1.c
-rwxrwxr-x  1 jere jere 7930 2012-01-26 18:09 fig3-2
-rw-r--r--  1 jere jere  478 2012-01-26 18:09 fig3-2.c
-rwxrwxr-x  1 jere jere 7826 2012-01-28 00:05 fig3-4
-rw-r--r--  1 jere jere  262
-rw-rw-r--  1 jere jere  399
-rwxrwxr-x  1 jere jere 7913
-rw-r--r--  1 jere jere  695
-rw-r--r--  1 jere jere  676
-rw-rw-r--  1 jere jere   11
-rw-rw-r--  1 jere jere  220
```



File Access Permissions

- There are nine permission bits for each file.

Figure 4.6. The nine file access permission bits, from <sys/stat.h>

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

Uses of the permissions are summarized as follows:

- To open a file, need execute permission on each directory component of the path
  - Example: Open the file `/usr/include/stdio.h`
- 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

File Access Permissions

- FileAccessTest: Tests performed by OS when process opens, creates or deletes file. Depend on the owners of the file (`st_uid`, `st_gid`), the effective IDs of the process (`EUID`, `EGID`) and the supplementary groups.

  1. If the `EUID` == 0 superuser!--> access allowed
  2. If the `EUID` == owner `UID` of the file
     - If appropriate access permissions, access allowed, otherwise denied
  3. If the `EGID` == owner `GID` of the file
     - If appropriate access permissions, access allowed, otherwise denied
  4. Check appropriate access permissions for others same way.

- These 4 steps are tried in sequence.

Ownership of New Files

- `UID` of a new file = the effective `UID` of the creating process
- `GID` of a new file – 2 options defined by POSIX.1:
  1. `GID` of the new file = the effective `GID` of the process
  2. `GID` of the new file = the `GID` of the directory in which the file is created.

access(2) Function

- Accessibility tests are performed by OS on open operation, based on effective UID and GID.
- Sometimes, need to test accessibility for a file based on real UID and GID.

```
#include <unistd.h>

int access(const char *pathname, int mode);

                                    Returns: 0 if OK, 1 on error
```

The mode parameter 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

Figure 4.8. Example of access function

```
#include "apue.h"
#include <fcntl.h>

int
main(int argc, char *argv[])
{
    if (argc != 2)
        err_quit("usage: a.out <pathname>");
    if (access(argv[1], R_OK) < 0)
        err_ret("access error for %s", argv[1]);
    else
        printf("read access OK\n");
    if (open(argv[1], O_RDONLY) <
        err_ret("open error for %
    else
        printf("open for reading
    exit(0);
}
```

```
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e] cp fig4.8 ./test/fig4-8.c
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e] cd test
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls
fig1-8    fig3-1   fig3-10.c  fig3-2    fig3-4    fig3-4-test.sh  fig4-3.c  Mak
fig1-8.c  fig3-10  fig3-1.c   fig3-2.c  fig3-4.c  fig4-3          fig4-8.c  tem
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] make fig4-8
gcc -DLINUX -ansi -I/home/jere/SystemProgramming/apue.2e/include -Wall -D_GNU_S
-lapue  -o fig4-8
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l fig4-8
-rwxrwxr-x 1 jere jere 7863 2012-02-02 17:44 fig4-8
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ./fig4-8 fig4-8
read access OK
open for reading OK
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l /etc/shadow
-rw-r----- 1 root shadow 1059 2012-01-02 23:16 /etc/shadow
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ./fig4-8 /etc/shadow
access error for /etc/shadow: Permission denied
open error for /etc/shadow: Permission denied
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] sudo chown root fig4-8
[sudo] password for jere:
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] sudo chmod u+s fig4-8
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l fig4-8
-rwsrwxr-x 1 root jere 7863 2012-02-02 17:44 fig4-8
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ./fig4-8 /etc/shadow
access error for /etc/shadow: Permission denied
open for reading OK
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test]
```

`umask(2)` Function

- Set the file creation mode mask.
- Any bits that are on in the file creation mask are turned off in the file's mode.

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

mode_t umask(mode_t cmask);


    Returns: previous file mode creation mask
```

- *cmask* = bitwise OR of any of file permissions `S_IRUSR`, `S_IWUSR`, `S_IXUSR`, `S_IRGRP`, `S_IWGRP`, `S_IXGRP`, `S_IROTH`, `S_IWOTH`, `S_IXOTH` (see Fig. 4.6)
- 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.

Figure 4.9. Example of `umask` function

```
#include "apue.h"
#include <fcntl.h>

#define RWRWRW (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

int
main(void)
{
    umask(0);
    if (creat("foo", RWRWRW) < 0)
        err_sys("creat error for foo");
    umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
    if (creat("bar", RWRWRW) < 0)
        err_sys("creat error for bar");
    exit(0);
}
```

Figure 4.10. The `umask` file access permission bits

Mask bit	Meaning
0400	user-read
0200	user-write
0100	user-execute
0040	group-read
0020	group-write
0010	group-execute
0004	other-read
0002	other-write
0001	other-execute

```
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e] cp fig4.9 ./test/fig4-9.c
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e] cd test
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls
fig1-8    fig3-1   fig3-10.c  fig3-2    fig3-4    fig3-4-test.sh  fig4-3.c  fig4-9
fig1-8.c  fig3-10  fig3-1.c   fig3-2.c  fig3-4.c  fig4-3          fig4-8.c  fig4-9
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] make fig4-9
gcc -DLINUX -ansi -I/home/jere/SystemProgramming/apue.2e/include -Wall -D_GNU_SOUR
-lapue  -o fig4-9
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] umask
0002
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ./fig4-9
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l foo bar
-rw------- 1 jere jere 0 2012-02-02 18:04 bar
-rw-rw-rw- 1 jere jere 0 2012-02-02 18:04 foo
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] umask
0002
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test]
```

```
$ umask          first print the current file mode creation mask
002
$ umask -S       print the symbolic form
u=rwx,g=rwx,o=rx
$ umask 027      change the file mode creation mask
$ umask -S       print the symbolic form
u=rwx,g=rx,o=
```

chmod

- chmod(1)
  - r:4, w:2, x:1

```
[root@www ~]# chmod [-R] xyz 檔案或目錄
選項與參數：
xyz : 就是剛剛提到的數字類型的權限屬性，為 rwx 屬性數值的相加。
-R : 進行遞迴(recursive)的持續變更，亦即連同次目錄下的所有檔案都會變更
```

```
$ chmod 664 myfile
$ ls -l myfile
-rw-rw-r--  1   57 Jul  3 10:13  myfile
```

- `chmod(1)`
  - Symbolic mode

Add the **r**ead and **w**rite permissions to the **u**ser and **g**roup classes of a directory:

```
$ chmod ug+rw mydir
$ ls -ld mydir
drw-rw----   2 unixguy  uguys  96 Dec 8 12:53 mydir
```

For a file, remove **w**rite permissions for **a**ll classes:

```
$ chmod a-w myfile
$ ls -l myfile
-r-xr-xr-x   2 unixguy  uguys 96 Dec 8 12:53 myfile
```

Set the permissions for the **u**ser and the **g**roup to **r**ead and e**x**ecute only (no write permission) on `mydir`.

```
$ chmod ug=rx mydir
$ ls -ld mydir
dr-xr-x---   2 unixguy  uguys 96 Dec 8 12:53 mydir
```

chmod(2)

- chmod(2) and fchmod(2)
  - Change file access permissions for an existing file
    - chmod function operates on the specified file
    - fchmod function operates on a file that has already been opened.
    - Callers must be a superuser or effective UID = file UID (file owner).

- *mode* = bitwise OR of constants defined in the next table

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

int chmod(const char *pathname, mode_t mode);

int fchmod(int filedes, mode_t mode);
```

Both return: 0 if OK, 1 on error

Figure 4.11. The *mode* constants for `chmod` functions, from `<sys/stat.h>`

mode	Description
S_ISUID	set-user-ID on execution
S_ISGID	set-group-ID on execution
S_ISVTX	saved-text (sticky bit)
S_IRWXU	read, write, and execute by user (owner)
S_IRUSR	read by user (owner)
S_IWUSR	write by user (owner)
S_IXUSR	execute by user (owner)
S_IRWXG	read, write, and execute by group
S_IRGRP	read by group
S_IWGRP	write by group
S_IXGRP	execute by group
S_IRWXO	read, write, and execute by other (world)
S_IROTH	read by other (world)
S_IWOTH	write by other (world)
S_IXOTH	execute by other (world)

chmod(2)

- S_ISUID
  - When a process runs a regular file that has the S_ISUID and S_IXUSR bit set, the effective user ID of the process is set to the owner ID of the file.
    - SUID option tells Linux to run the process with the permissions of the owner of the program.
  - It is indicated by s in the owner's execute bit position: rwsr-xr-x
    - E.g., if a file is owned by root and has its SUID bit set, the program runs with root privileges and can therefore read any file on the computer.

chmod(2)

- S_ISGID
  - When a process runs a regular file that has both the S_ISGID bit and the S_IXGRP permission bit set, the effective user ID of the process is set to the group ID of the file.
  - It is indicated by **s** in the group execute bit position: rwxr-**s**r-x

Figure 4.12. Example of `chmod` function

```
#include "apue.h"

int
main(void)
{
     struct stat      statbuf;

     /* turn on set-group-ID and turn off group-execute */

     if (stat("foo", &statbuf) < 0)
          err_sys("stat error for foo");
     if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
          err_sys("chmod error for foo");

     /* set absolute mode to "rw-r--r--" */

     if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
          err_sys("chmod error for bar");

     exit(0);
}
```

```
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l foo bar
-rw------- 1 jere jere 0 2012-02-02 18:04 bar
-rw-rw-rw- 1 jere jere 0 2012-02-02 18:04 foo
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] cp ../fig4.12 ./fig4-12.c
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] make fig4-12
gcc -DLINUX -ansi -I/home/jere/SystemProgramming/apue.2e/include -Wall -D_GNU_SOUR
 -lapue  -o fig4-12
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ./fig4-12
jere@VirtualBox-MBP [~/SystemProgramming/apue.2e/test] ls -l foo bar
-rw-r--r-- 1 jere jere 0 2012-02-02 18:04 bar
-rw-rwSrw- 1 jere jere 0 2012-02-02 18:04 foo
```

Why is uppercase "**S**"? What does it mean?
Your homework.

Sticky bit (`S_ISVTX`)

- In modern Linux implementations, the sticky bit is used to protect files from being deleted by those who don't own the file.
- When this bit is present on a directory, the directory's files can only be deleted or renamed by a user who has write permission for the directory and is:
  - The owner of the file
  - The owner of the directory
  - Superuser
- The sticky bit is indicated by a **t** in the others execute bit, rwxr-xr-**t**

```
drwxr-xr-x  12 root root     0 2011-12-13 18:26 sys
drwxrwxrwt  12 root root  4096 2012-02-05 13:17 tmp
drwxr-xr-x  10 root root  4096 2011-10-12 22:27 usr
```

chown

- chown(1)

```
[root@www ~]# chown [-R] 帳號名稱 檔案或目錄
[root@www ~]# chown [-R] 帳號名稱:群組名稱 檔案或目錄
選項與參數：
-R : 進行遞迴(recursive)的持續變更，亦即連同次目錄下的所有檔案都變更

範例：將install.log的擁有者改為bin這個帳號：
[root@www ~]# chown bin install.log
[root@www ~]# ls -l
-rw-r--r--  1 bin  users 68495 Jun 25 08:53 install.log

範例：將install.log的擁有者與群組改回為root：
[root@www ~]# chown root:root install.log
[root@www ~]# ls -l
-rw-r--r--  1 root root 68495 Jun 25 08:53 install.log
```

chown

- chgrp(1)

```
[root@www ~]# chgrp [-R] dirname/filename ...
選項與參數：
-R：進行遞迴(recursive)的持續變更，亦即連同次目錄下的所有檔案、目錄
   都更新成為這個群組之意。常常用在變更某一目錄內所有的檔案之情況。
範例：
[root@www ~]# chgrp users install.log
[root@www ~]# ls -l
-rw-r--r--  1 root users 68495 Jun 25 08:53 install.log
[root@www ~]# chgrp testing install.log
chgrp: invalid group name `testing' <== 發生錯誤訊息囉～找不到這個群組名～
```

chown

- `chown(2)`, `fchown(2)`, and `lchown(2)` allow us to change the user ID of a file and the group ID of a file.

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t
➥  group);

int fchown(int filedes, uid_t owner, gid_t group);

int lchown(const char *pathname, uid_t owner,
➥  gid_t group);
```

All three return: 0 if OK, 1 on error

  - If either of the arguments *`owner`* or *`group`* is -1, the corresponding ID is left unchanged.
  - 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).

File Size

- `st_size` in the `stat` struct is the size of the file in bytes.
  - Regular file – size of 0 is allowed (`EOF` on first read)
  - Directory – multiple of directory entry size, such as 16 or 512 (talk about this later)
  - Symbolic link – number of bytes in referenced file name
    - E.g., `lib -> usr/lib` has `st_size` 7.

File Size

```
drwxr-xr-x  2 jere jere 4096 2012-01-09 16:27 lock
-rw-r--r--  1 jere jere  552 2005-05-29 15:10 Make.defines.freebsd
-rw-r--r--  1 jere jere  568 2005-05-29 15:10 Make.defines.linux
-rw-r--r--  1 jere jere  542 2005-05-29 15:10 Make.defines.macos
-rw-r--r--  1 jere jere  550 2005-05-29 15:10 Make.defines.solaris
-rw-r--r--  1 jere jere  678 2005-05-29 03:54 Makefile
drwxr-xr-x  2 jere jere 4096 2012-01-09 16:27 mycat
```

```
drwxr-xr-x   3 root root  4096 2011-12-13 00:33 home
lrwxrwxrwx   1 root root    32 2011-12-13 00:41 initrd.img -> boot/initrd.img-3.0.0-12-generic
drwxr-xr-x  20 root root  4096 2011-12-13 18:51 lib
drwx------   2 root root 16384 2011-12-13 00:26 lost+found
drwxr-xr-x   4 root root  4096 2011-12-13 18:27 media
drwxr-xr-x   2 root root  4096 2011-10-09 15:29 mnt
drwxr-xr-x   3 root root  4096 2011-12-13 01:07 opt
dr-xr-xr-x 143 root root     0 2011-12-13 18:26 proc
drwx------   3 root root  4096 2011-12-13 00:55 root
drwxr-xr-x  20 root root   800 2012-02-03 01:27 run
drwxr-xr-x   2 root root  4096 2011-12-13 01:09 sbin
drwxr-xr-x   2 root root  4096 2011-06-22 02:43 selinux
drwxr-xr-x   2 root root  4096 2011-10-12 22:27 srv
drwxr-xr-x  12 root root     0 2011-12-13 18:26 sys
drwxrwxrwt  12 root root  4096 2012-02-05 14:17 tmp
drwxr-xr-x  10 root root  4096 2011-10-12 22:27 usr
drwxr-xr-x  13 root root  4096 2011-12-13 18:26 var
lrwxrwxrwx   1 root root    29 2011-12-13 00:41 vmlinuz -> boot/vmlinuz-3.0.0-12-generic
```