



Files and directories



Contents

File system = file data + file attribute

- File attributes
 - Type, permission, size, time, user/group ID
- File attributes modification
 - chown, chmod, ...
- Symbolic link
- Directories

stat(), fstat(), lstat()

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

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

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

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

All three return: 0 if OK, -1 on error

stat

- Gets information about the named file.

fstat

- Gets information about the file that is already open.

lstat

- Gets information about the symbolic link itself.
- If pathname is not a symbolic link, equivalent to *stat*.

stat(), fstat(), lstat()

Status structure

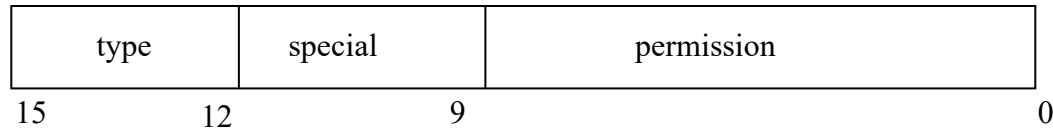
- can differ among implementations

```
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) */  
    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 */  
};
```

st_mode

Format of 'st_mode'

- Composed of three parts:
file type, special bit, and file permission bit.



st_mode

file type

● #define S_IFSOCK 0140000	/* socket */
● #define S_IFLNK 0120000	/* symbolic link */
● #define S_IFREG 0100000	/* regular */
● #define S_IFBLK 0060000	/* block special */
● #define S_IFDIR 0040000	/* directory */
● #define S_IFCHR 0020000	/* character special */
● #define S_IFIFO 0010000	/* FIFO */

st_mode

special bits

- `#define S_ISUID 0004000 /* set uid on execution */`
- `#define S_ISGID 0002000 /* set group id on execution */`
- `#define S_ISVTX 0001000 /* save text(sticky bit) */`

Usage of the sticky bit

- In early time, used to save a running program to swap area for speed up.
 - Needless because of the virtual memory
- Currently, it was extended for the following functions
 - In `/tmp` and `/var/spool/uucppublic`, all files are readable, writable, and executable by all users.
 - But, the files must not be deleted/renamed by non-owner.
 - Thus, the sticky bit is set in these directories.

st_mode

permission bits

- `#define S_IRUSR 00400` /* read permission: owner */
- `#define S_IWUSR 00200` /* write permission: owner */
- `#define S_IXUSR 00100` /* execute permission: owner */

- `#define S_IRGRP 00040` /* read permission: group */
- `#define S_IWGRP 00020` /* write permission: group */
- `#define S_IXGRP 00010` /* execute permission: group */

- `#define S_IROTH 00004` /* read permission: other */
- `#define S_IWOTH 00002` /* write permission: other */
- `#define S_IXOTH 00001` /* execute permission: other */

File types

About file type

Regular file

- Contains some form of data.
- No distinction to UNIX kernel whether the data is text or binary. (Applications interpret the file contents.)

Directory

- Contains the names of other files and pointers to information on these files.

Block special file

- Provides buffered I/O access in fixed-size units to devices
- E.g. disk

File types

About file type(cont.)

- Character special file
 - Provides unbuffered I/O access in variable-sized units to devices
 - Keyboard, mouse, ...
- FIFO
 - used for communication between processes.
 - Also called *named pipe*
- Socket
 - used for network communication between processes.
- Symbolic link
 - Points to another file.

File types

File type macros

- Argument of macros is the **st_mode** from the stat structure.

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

File types

example

```
#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;
        }
    }
}
```

File types

```
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);
}
```

File types

Running result

```
$ ./a.out /etc/passwd /etc /run/initctl /dev/log /dev/tty \  
> /dev/sda /dev/cdrom  
/etc/passwd: regular  
/etc: directory  
/run/initctl : fifo  
/dev/log: socket  
/dev/tty: character special  
/dev/sda : block special  
/dev/cdrom: symbolic link
```

File types

 different file types in a Linux systems

File type	Count	Percentage
regular file	226,856	88.22 %
directory	23,017	8.95
symbolic link	6,442	2.51
character special	447	0.17
block special	312	0.12
socket	69	0.03
FIFO	1	0.00

Set-user-ID and set-group-ID

❏ Every process has six or more than IDs

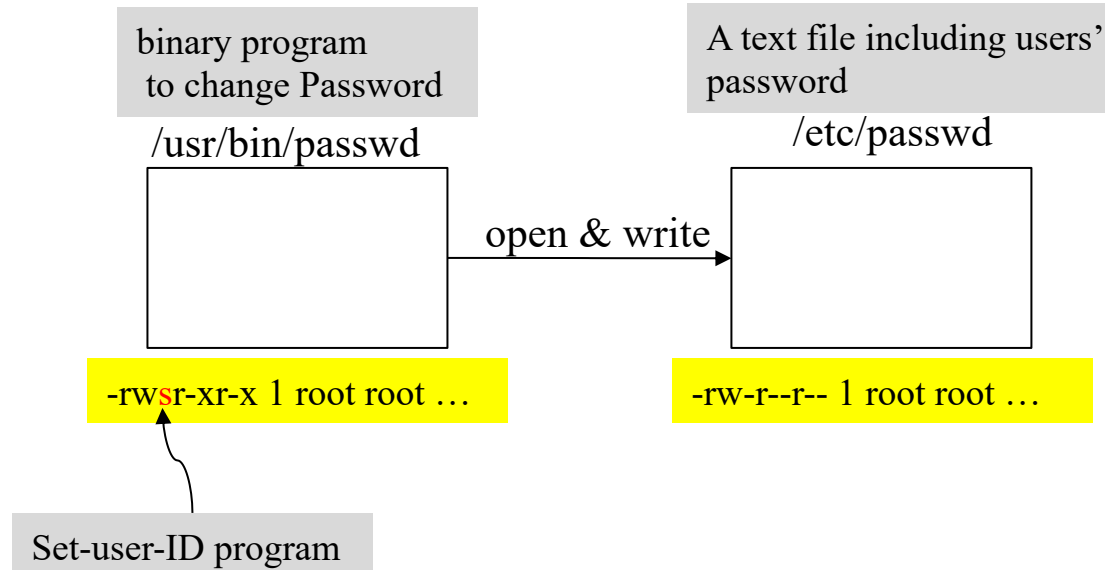
❏ real ID and effective ID

- real user(group) ID
 - Identifies who we really are.
 - Written in the password file(/etc/passwd,/etc/shadow).
- effective user(group) ID
 - Determines file access permission.
- Normally, effective user(group) ID = real user(group) ID.
- But, effective user(group) ID can be different from the real user(group) ID, in the case of programs with the **setuid**(setgid) bit set.

Set-user-ID and set-group-ID

❏ In case of changing password by using passwd program

- Has a root's privilege while passwd program is running
 - Real user ID of passwd program: user
 - Effective user ID of passwd program: root



File access permissions

R/W/X of user, group, others

Read

- file: can read the file data?(able to copy?)
- Directory: can read the file list?(i.e., able to run 'ls?')

write

- File: can modify the file data?
- Directory: can create or delete the file?

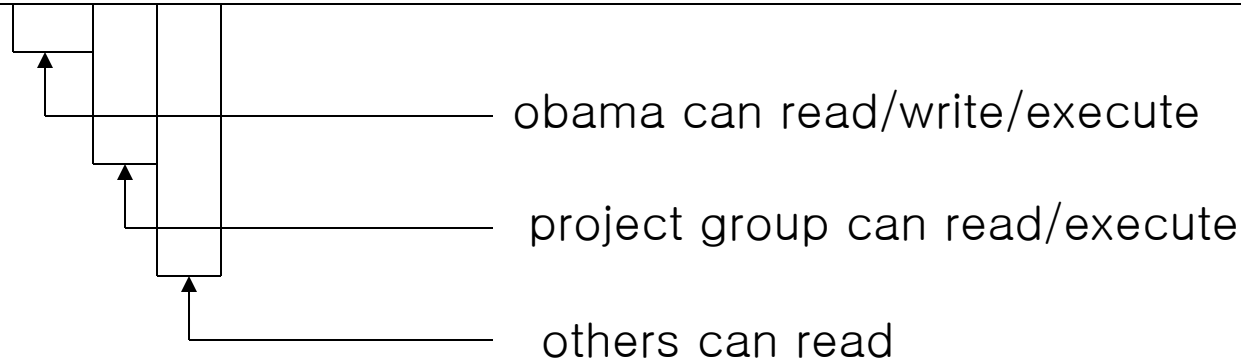
execute

- File: can execute the file?
- Directory: can move there?(i.e., able to run 'cd?')

File access permissions

File permission example

```
-rwxr-xr-- 1 obama project 14499 Jun 19 05:33 toast
```



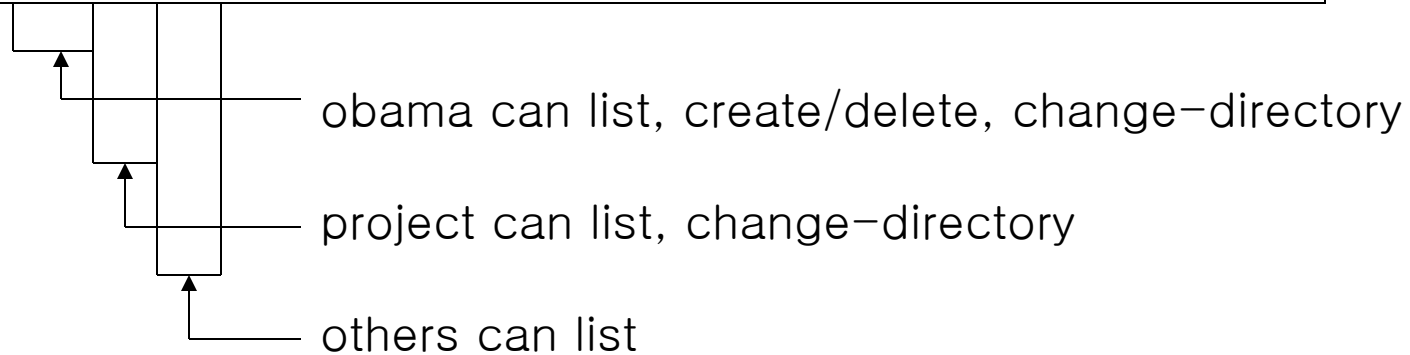
Character	Permission	Octal
R	Read	4
W	Write	2
X	Execute	1

Character	Permission	Value
---	None	0
r--	readable	4
rw-	readable/writable	6
rwX	readable/writable/executable	7
r-x	readable/executable	5
--X	executable	1

File access permissions

Directory permission example

```
drwxr-xr-- 2 obama project 1024 Oct 22 20:19 bin
```



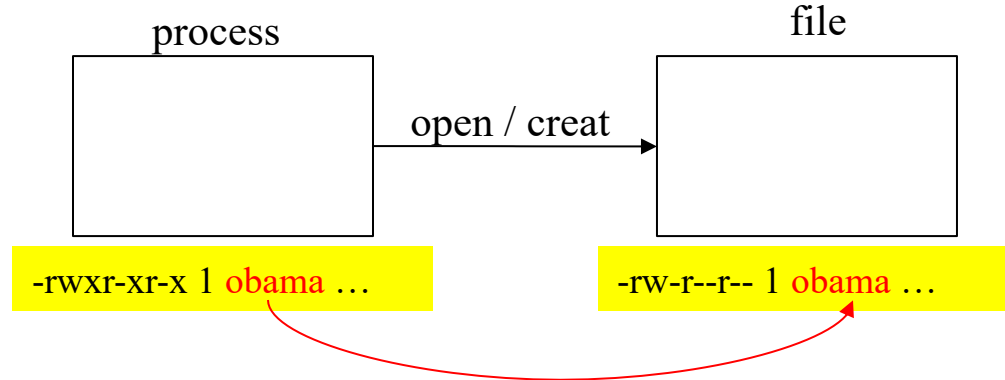
File access permissions

Frequently used file access permissions

Permission	Value	Description
<code>-rw-----</code>	600	?
<code>-rw-r--r--</code>	644	
<code>-rw-rw-rw-</code>	666	
<code>-rwx-----</code>	700	
<code>-rwxr-xr-x</code>	755	
<code>-rwxrwxrwx</code>	777	
<code>-rwx--x--x</code>	711	
<code>drwx-----</code>	700	
<code>drwxr-xr-x</code>	755	
<code>drwx--x--x</code>	711	

Ownerships of new files

- ❏ When a file is created using open or create
- Generally, the user ID of a new file is set to the effective user ID of the process.
 - What about group ID? There are two options.



access()

```
#include <unistd.h>
```

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

Returns: 0 if OK, -1 on error

 Check the access permission based on real ID.

 mode

mode	Description
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()

example

```
#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) < 0)
        err_ret("open error for %s", argv[1]);
    else
        printf("open for reading OK\n");
    exit(0);
}
```


access()

Execution

```
$ whoami
obama
$ ls -l a.out
-rwxrwxr-x 1 obama project 15945 Nov 30 12:10 a.out
$ ./a.out a.out
read access OK
open for reading OK
$ ls -l /etc/shadow
-r----- 1 root root 1315 Jul 17 2002 /etc/shadow
$ ./a.out /etc/shadow
access error for /etc/shadow
open error for /etc/shadow
$ sudo chown root a.out
$ sudo chmod u+s a.out
$ ls -l a.out
-rwsrwxr-x 1 root project 16216 9월 23 18:00 a.out
$ ./a.out /etc/shadow
access error for /etc/shadow
open for reading OK
```

umask()

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

```
mode_t umask(mode_t cmask);
```

Returns: previous file mode creation mask

- ❏ Set the file mode creation mask
- ❏ permissions in the umask are **turned off** from the mode argument to open()

```
umask(022);
```

```
fd = creat("tmp", 0666);
```

```
...
```

➔ permission of "tmp" is -rw-r--r-- (0644)

umask()

 The umask file access permission bits <sys/stat.h>

st_mode mask	Meaning	Mask bit
S_IRUSR	user-read	0400
S_IWUSR	user-write	0200
S_IXUSR	user-execute	0100
S_IRGRP	group-read	0040
S_IWGRP	group-write	0020
S_IXGRP	group-execute	0010
S_IROTH	other-read	0004
S_IWOTH	other-write	0002
S_IXOTH	other-execute	0001

umask()

Example

```
#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);
}
```

Diagram illustrating the effect of `umask` on file permissions:

- `umask(0)` results in permissions `666` (indicated by a red arrow).
- `umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)` results in permissions `066` (indicated by a red arrow).
- The final permissions for the file created in the second `creat` call are `600` (indicated by a red arrow).

umask()

Execution

```
$ ./a.out
$ ls -l foo bar
-rw----- 1 sar      0 Dec 7 21:20 bar
-rw-rw-rw- 1 sar      0 Dec 7 21:20 foo
$
```

참고) umask utility

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

chmod() and fchmod()

```
#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

Change the file access permission

- chmod: for the specified file
- fchmod: for an open file

To change the permission bits of a file

- the effective user ID of the process must be equal to the owner ID of the file, or
- the process must have super-user permissions.

chmod() and fchmod()

Mode (in <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() and fchmod()

Example

```
#include "apue.h"

int main(void)
{
    struct stat    statbuf;

    /* turn on group-execute and turn off group-write */
    if (stat("foo", &statbuf) < 0)
        err_sys("stat error for foo");
    if (chmod("foo", (statbuf.st_mode & ~S_IWGRP) | 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);
}
```


chmod() and fchmod()

Execution

```
$ ls -l foo bar
-rw----- 1 sar          0 Dec 7 21:20 bar
-rw-rw-rw- 1 sar          0 Dec 7 21:20 foo
$ ./a.out
$ ls -l foo bar
-rw-r--r-- 1 sar          0 Dec 7 21:20 bar
-rw-r-Srw- 1 sar          0 Dec 7 21:20 foo
$
```



s : set-group-ID bit is set w/ the group-execution bit being set

S : set-group-ID bit is set w.o. the group-execution bit being set

* Sticky bit is represented by 't' or 'T' in Other's execution bit location.

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

```
#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

Change the user ID and the group ID

- chown: of the specified file
- fchown: of an open file
- lchown: of the symbolic link itself, not the file pointed to by the symbolic link

 In BSD-based system, only **superuser may change** the owner of a file.

 If the arguments *owner* or *group* is specified as -1, then the corresponding ID is not changed.

File Size

st_size in stat structure

- The size of the file in bytes
- Valid only for regular, directory, and symbolic link files
- In a symbolic link file, it denotes the length of the pathname of the target file.

For block sizes,

- st_blksize : the block size for I/O
- st_blocks : the actual number of blocks allocated

truncate()

```
#include <unistd.h>
```

```
int truncate(const char *pathname, off_t length);
```

```
int ftruncate(int fildes, off_t length);
```

Returns: 0 if OK, -1 on error

 Truncate an existing file to *length* bytes

- Emptying a file is a special case
 - Open a file with O_TRUNC flag
- May make a hole

link()

```
#include <unistd.h>
```

```
int link(const char *existingpath, const char *newpath);
```

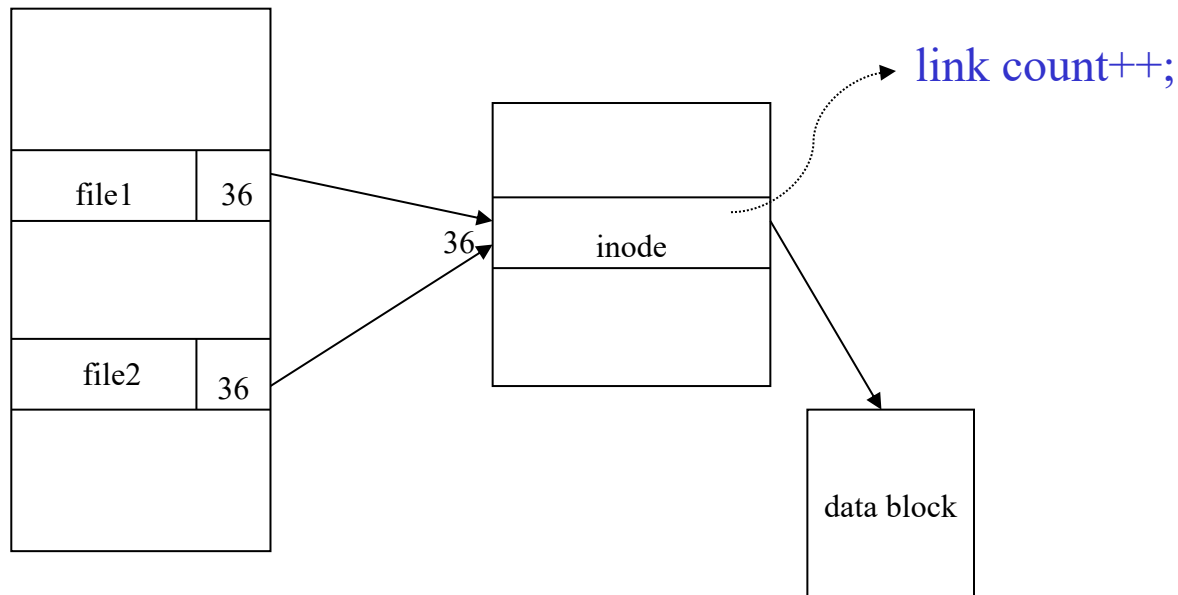
Returns: 0 if OK, -1 on error

Create a link(hard link) to an existing file

- creates a new directory entry, *newpath*, that references the existing file *existingpath*.
- If *newpath* exists, an error is returned.

link()

link("file1", "file2");







- It is impossible to tell which name was the original.
- **Hard links**, as created by `link`, cannot span file systems.

unlink()

```
#include <unistd.h>
```

```
int unlink(const char *pathname);
```

Returns: 0 if OK, -1 on error

-  Removes the directory entry and decrements the link count of the file referenced by pathname.
 -  If other process has opened the file, its contents will not be deleted.
 -  When the link count reaches 0, file content is deleted.
-  Useful to ensure that a temporary file won't be left in case the process crashes (see the following example.)

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

int
main(void)
{
    if (open("tempfile", O_RDWR) < 0)
        err_sys("open error");
    if (unlink("tempfile") < 0)
        err_sys("unlink error");
    printf("file unlinked\n");
    sleep(15);
    printf("done\n");
    exit(0);
}
```

Figure 4.16 Open a file and then unlink it

Symbolic links

- ❏ hard link points directly to the inode of the file.
 - The link and the file should reside in the same file system.
 - Only the superuser can create a hard link to a directory.

- ❏ symbolic link is an indirect pointer to a file.
 - There are no file system limitations on a symbolic link.
 - Anyone can create a symbolic link to a directory

- ❏ See Figure 4.17 for treatment of symbolic links by various functions

Function	Does not follow symbolic link	Follows symbolic link
access		•
chdir		•
chmod		•
chown		•
creat		•
exec		•
lchown	•	
link		•
lstat	•	
open		•
opendir		•
pathconf		•
readlink	•	
remove	•	
rename	•	
stat		•
truncate		•
unlink	•	

Figure 4.17 Treatment of symbolic links by various functions

symlink()

```
#include <unistd.h>
```

```
int symlink(const char *actualpath, const char *sympath);
```

Returns: 0 if OK, -1 on error

 Create a new directory entry, *sympath* that points to *actualpath*.

- Not require that *actualpath* exist when the symbolic link is created.
- *Actualpath* and *sympath* need not reside in the same file system.

symlink()

Dangling link

- may point to a non-existing file

<pre>\$ ln -s /no/such/file myfile \$ ls myfile myfile \$ cat myfile cat: myfile: No such file or directory \$ ls -l myfile lrwxrwxrwx 1 sar 13 Jan 22 00:26 myfile -> /no/such/file \$</pre>	<div>create a symbolic link</div> <div>ls says it's there</div> <div>so we try to look at it</div> <div>try -l option</div>	Example 1
---	---	-----------

<pre>\$ ln -s testfile newfile \$ ls -l newfile lrwxrwxrwx 1 yhshin users 8 Aug 27 20:02 newfile -> testfile \$ rm testfile \$ cat newfile cat: newfile: No such file or directory \$</pre>	<div></div> <div></div> <div></div> <div></div>	Example 2
--	---	-----------

readlink()

```
#include <unistd.h>
```

```
ssize_t readlink(const char *pathname, char *buf, size_t bufsiz);
```

Returns: number of bytes read if OK, -1 on error

read value of a symbolic link

- places the contents of the symbolic link path in the buffer *buf*, which has size *bufsiz*.

Return value

- the count of characters placed in the buffer if it succeeds
- -1 if an error occurs

remove()

```
#include <stdio.h>
```

```
int remove(const char *pathname);
```

Returns: 0 if OK, -1 on error

Unlink a file or a directory

- For a file, identical to [unlink](#).
- For a directory, identical to [rmdir](#).

rename()

```
#include <stdio.h>
```

```
int rename(const char *oldname, const char *newname);
```

Returns: 0 if OK, -1 on error

Rename

- **renames** a file or a directory.
- oldname and newname should be in the same file system.

File times

 The three time values associated with each file (see Figure 4.20)

Field	Description	Example	ls(1) option
st_atime	last-access time of file data	read	-u
st_mtime	last-modification time of file data	write	default
st_ctime	last-change time of i-node status	chmod, chown	-c

- st_mtime: time the **file contents** were last modified.
- st_ctime: time the **inode of the file** was last modified.

Function	Referenced file or directory			Parent directory of referenced file or directory			Section	Note
	a	m	c	a	m	c		
chmod, fchmod			•				4.9	
chown, fchown			•				4.11	
creat	•	•	•		•	•	3.4	O_CREAT new file
creat		•	•				3.4	O_TRUNC existing file
exec	•						8.10	
lchown			•				4.11	
link			•		•	•	4.15	parent of second argument
mkdir	•	•	•		•	•	4.21	
mkfifo	•	•	•		•	•	15.5	
open	•	•	•		•	•	3.3	O_CREAT new file
open		•	•				3.3	O_TRUNC existing file
pipe	•	•	•				15.2	
read	•						3.7	
remove			•		•	•	4.15	remove file = unlink
remove					•	•	4.15	remove directory = rmdir
rename			•		•	•	4.16	for both arguments
rmdir					•	•	4.21	
truncate, ftruncate		•	•				4.13	
unlink			•		•	•	4.15	
utimes, utimensat, futimens	•	•	•				4.20	
write		•	•				3.8	

Figure 4.20 Effect of various functions on the access, modification, and changed-status times

utime()

```
#include <utime.h>
```



```
int utime(const char *pathname, const struct utimbuf *times);
```

Returns: 0 if OK, -1 on error

Change the access time and modified time

utimbuf structure

```
struct utimbuf {  
    time_t actime; /* access time */  
    time_t modtime; /* modification time */  
}
```

-  If times is NULL, the access and modification times of the file are set to the current time.
-  st_ctime is automatically updated when the utime is called.

utime()

example

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

int
main(int argc, char *argv[])
{
    int      i, fd;
    struct stat  statbuf;
    struct utimbuf timebuf;

    for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) { /* fetch current times */
            err_ret("%s: stat error", argv[i]);
            continue;
        }
    }
}
```

utime()

example(cont.)

```
if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
    err_ret("%s: open error", argv[i]);
    continue;
}
close(fd);
timebuf.actime = statbuf.st_atime;
timebuf.modtime = statbuf.st_mtime;
if (utime(argv[i], &timebuf) < 0) { /* reset times */
    err_ret("%s: utime error", argv[i]);
    continue;
}
}
exit(0);
}
```

utime()

Execution

```
$ ls -l changemod times      look at sizes and last-modification times
-rwxrwxr-x 1 sar  15019  Nov 18 18:53 changemod
-rwxrwxr-x 1 sar  16172  Nov 19 20:05 times
$ ls -lu changemod times    look at last-access times
-rwxrwxr-x 1 sar  15019  Nov 18 18:53 changemod
-rwxrwxr-x 1 sar  16172  Nov 19 20:05 times
$ date                      print today's date
Thu Jan 22 06:55:17 EST 2004
$ ./a.out changemod times   run the program in the previous page
$ ls -l changemod times    and check the results
-rwxrwxr-x 1 sar    0 Nov 18 18:53 changemod
-rwxrwxr-x 1 sar    0 Nov 19 20:05 times
$ ls -lu changemod times   check the last-access times also
-rwxrwxr-x 1 sar    0 Nov 18 18:53 changemod
-rwxrwxr-x 1 sar    0 Nov 19 20:05 times
$ ls -lc changemod times   and the changed-status times
-rwxrwxr-x 1 sar    0 Jan 22 06:55 changemod
-rwxrwxr-x 1 sar    0 Jan 22 06:55 times
$
```

mkdir()

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

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

Returns: 0 if OK, -1 on error



Create a new empty directory.

- The entries for dot and dot-dot are automatically created.

rmkdir()

```
#include <unistd.h>
```

```
int rmkdir(const char *pathname);
```

Returns: 0 if OK, -1 on error

 Delete an empty directory.

- An empty directory is one that contains entries only for dot and dot-dot.

Reading directories

Write permission bits for a directory

- Means that we can create/remove files in the directory.
- Does not mean that we can write to the directory itself.
- ➔ We need some APIs that can deal with directory itself.

filename	i-number
filename	i-number
filename	i-number
⋮	
filename	i-number

Directory entry

A typical directory format
(Details of directory formats are system dependent.)

Reading directories

```
#include <dirent.h>
```

```
DIR *opendir(const char *pathname);
```

Returns: pointer if OK, NULL on error

```
int closedir(DIR *dp);
```

Returns: 0 if OK, -1 on error

Open a directory/close an open directory

- DIR: represents a directory stream

- Defined in <dirent.h>.
- Similar to FILE type in the standard I/O library.

Reading directories

```
#include <dirent.h>
```

```
struct dirent *readdir(DIR *dp);
```

Returns: pointer if OK, NULL at end of directory or error

 Read directory entry into a dirent structure.

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

- On the first call, the first directory entry is read into dirent.
- On completion, the directory pointer is moved onto the next entry in the directory.
- If the end of the directory is reached, NULL is returned.

Reading directories

```
#include <dirent.h>
```

```
void rewinddir(DIR *dp);
```

 If you want to reread from the beginning of directory, use `rewinddir`.

- Following the `rewinddir` call the next `readdir` will return the first entry of the directory.

Reading directories

example

```
#include <stdio.h>
#include <dirent.h>

main (int argc, char ** argv)
{
    char pathname[128];

    if (argc == 1) {
        strcpy(pathname, ".");
    }
    else if (argc > 2) {
        printf("Too many parameter...\n");
        exit(1);
    }
    else {
        strcpy(pathname, argv[1]);
    }
    if (my_double_ls(pathname) == -1 ) printf("Could not open the directory\n");
}
```

Reading directories

example (cont.)

```
int my_double_ls (const char *name) {
    struct dirent *d;
    DIR *dp;

    if ((dp = opendir(name)) == NULL)
        return (-1);
    while (d = readdir(dp)) {
        if (d->d_ino != 0)
            printf ("%s\n", d->d_name);
    }

    rewinddir(dp);
    while (d = readdir(dp)) {
        if (d->d_ino != 0)
            printf ("%s\n", d->d_name);
    }
    closedir(dp);
    return (0);
}
```

Reading directories

Execution

```
$ ls temp_dir/  
abc bookmark fred  
$ ./a.out temp_dir/  
fred  
abc  
bookmark  
..  
.  
fred  
abc  
bookmark  
..  
.  
$
```

chdir() and fchdir()

```
#include <unistd.h>
```

```
int chdir(const char *pathname);  
int fchdir(int filedes);
```

Both return: 0 if OK, -1 on error



Change the current working directory.

- Specify the new current working directory either as a pathname or file descriptor.

chdir() and fchdir()

Example

```
#include "apue.h"

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

● \$ gcc -o mycd chdir.c

chdir() and fchdir()

Execution

```
$ pwd
/usr/lib
$ mycd
chdir to /tmp succeeded
$ pwd
/usr/lib
$
```

- Current working directory of shell didn't change after mycd.
- Each program is run in a separate process
 - ➔ current working directory of shell is unaffected by *chdir* in mycd.
- Note that “cd” is a **built-in shell command**!

getcwd()

```
#include <unistd.h>
```

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

Returns: buf if OK, NULL on error

 Obtain the current working directory.

- Copies an absolute pathname of the current working directory to the array pointed to by *buf* of length, *size*.
- *size* must be large enough to accommodate
 - the absolute pathname + a terminating null byte.

getcwd()

example

```
#include <stdio.h>
#include <unistd.h>
#define SIZE 200
void my_pwd (void);

int main()
{
    my_pwd();
}

void my_pwd (void) {
    char dirname[SIZE];

    if ( getcwd(dirname, SIZE) == NULL)
        perror("getcwd error");
    else
        printf("%s\n", dirname);
}
```

getcwd()



Execution

```
$ ./a.out  
/home/obama/test  
$
```