15

FILE ATTRIBUTES

In this chapter, we investigate various attributes of files (file metadata). We begin with a description of the *stat()* system call, which returns a structure containing many of these attributes, including file timestamps, file ownership, and file permissions. We then go on to look at various system calls used to change these attributes. (The discussion of file permissions continues in Chapter 17, where we look at access control lists.) We conclude the chapter with a discussion of i-node flags (also known as *ext2* extended file attributes), which control various aspects of the treatment of files by the kernel.

15.1 Retrieving File Information: *stat()*

The *stat()*, *lstat()*, and *fstat()* system calls retrieve information about a file, mostly drawn from the file i-node.

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

int stat(const char *pathname, struct stat *statbuf);
int lstat(const char *pathname, struct stat *statbuf);
int fstat(int fd, struct stat *statbuf);
```

All return 0 on success, or –1 on error

These three system calls differ only in the way that the file is specified:

- *stat()* returns information about a named file;
- *lstat()* is similar to *stat()*, except that if the named file is a symbolic link, information about the link itself is returned, rather than the file to which the link points; and
- *fstat()* returns information about a file referred to by an open file descriptor.

The *stat()* and *lstat()* system calls don't require permissions on the file itself. However, execute (search) permission is required on all of the parent directories specified in *pathname*. The *fstat()* system call always succeeds, if provided with a valid file descriptor.

All of these system calls return a *stat* structure in the buffer pointed to by *statbuf*. This structure has the following form:

```
struct stat {
    dev_t     st_dev;         /* IDs of device on which file resides */
    ino_t     st_ino;         /* I-node number of file */
    mode_t    st_mode;        /* File type and permissions */
    nlink_t   st_nlink;       /* Number of (hard) links to file */
    uid_t     st_uid;         /* User ID of file owner */
    gid_t     st_gid;         /* Group ID of file owner */
    dev_t     st_rdev;        /* IDs for device special files */
    off_t     st_size;        /* Total file size (bytes) */
    blksize_t st_blksize;     /* Optimal block size for I/O (bytes) */
    blkcnt_t  st_blocks;      /* Number of (512B) blocks allocated */
    time_t    st_atime;       /* Time of last file access */
    time_t    st_mtime;       /* Time of last file modification */
    time_t    st_ctime;       /* Time of last status change */
};
```

The various data types used to type the fields in the *stat* structure are all specified in SUSv3. See Section 3.6.2 for further information about these types.

> According to SUSv3, when *lstat()* is applied to a symbolic link, it needs to return valid information only in the *st_size* field and in the file type component (described shortly) of the *st_mode* field. None of other fields (e.g., the time fields) need contain valid information. This gives an implementation the freedom to not maintain these fields, which may be done for efficiency reasons. In particular, the intent of earlier UNIX standards was to allow a symbolic link to be implemented either as an i-node or as an entry in a directory. Under the latter implementation, it is not possible to implement all of the fields required by the *stat* structure. (On all major contemporary UNIX implementations, symbolic links are implemented as i-nodes.) On Linux, *lstat()* returns information in all of the *stat* fields when applied to a symbolic link.

In the following pages, we look at some of the *stat* structure fields in more detail, and finish with an example program that displays the entire *stat* structure.

Device IDs and i-node number

The *st_dev* field identifies the device on which the file resides. The *st_ino* field contains the i-node number of the file. The combination of *st_dev* and *st_ino* uniquely identifies a file across all file systems. The *dev_t* type records the major and minor IDs of a device (Section 14.1).

If this is the i-node for a device, then the *st_rdev* field contains the major and minor IDs of the device.

The major and minor IDs of a *dev_t* value can be extracted using two macros: `major()` and `minor()`. The header file required to obtain the declarations of these two macros varies across UNIX implementations. On Linux, they are exposed by `<sys/types.h>` if the `_BSD_SOURCE` macro is defined.

The size of the integer values returned by `major()` and `minor()` varies across UNIX implementations. For portability, we always cast the returned values to *long* when printing them (see Section 3.6.2).

File ownership

The *st_uid* and *st_gid* fields identify, respectively, the owner (user ID) and group (group ID) to which the file belongs.

Link count

The *st_nlink* field is the number of (hard) links to the file. We describe links in detail in Chapter 18.

File type and permissions

The *st_mode* field is a bit mask serving the dual purpose of identifying the file type and specifying the file permissions. The bits of this field are laid out as shown in Figure 15-1.

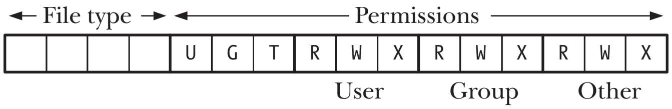

Figure 15-1: Layout of *st_mode* bit mask

The file type can be extracted from this field by ANDing (`&`) with the constant `S_IFMT`. (On Linux, 4 bits are used for the file-type component of the *st_mode* field. However, because SUSv3 makes no specification about how the file type is represented, this detail may vary across implementations.) The resulting value can then be compared with a range of constants to determine the file type, like so:

```
if ((statbuf.st_mode & S_IFMT) == S_IFREG)
    printf("regular file\n");
```

Because this is a common operation, standard macros are provided to simplify the above to the following:

```
if (S_ISREG(statbuf.st_mode))
    printf("regular file\n");
```

The full set of file-type macros (defined in `<sys/stat.h>`) is shown in Table 15-1. All of the file-type macros in Table 15-1 are specified in SUSv3 and appear on Linux. Some other UNIX implementations define additional file types (e.g., `S_IFDOOR`, for door files on Solaris). The type `S_IFLNK` is returned only by calls to *lstat()*, since calls to *stat()* always follow symbolic links.

The original POSIX.1 standard did not specify the constants shown in the first column of Table 15-1, although most of them appeared on most UNIX implementations. SUSv3 requires these constants.

> In order to obtain the definitions of `S_IFSOCK` and `S_ISSOCK()` from `<sys/stat.h>`, we must either define the `_BSD_SOURCE` feature test macro or define `_XOPEN_SOURCE` with a value greater than or equal to 500. (The rules have varied somewhat across *glibc* versions: in some cases, `_XOPEN_SOURCE` must be defined with a value of 600 or greater.)

Table 15-1: Macros for checking file types in the *st_mode* field of the *stat* structure

Constant	Test macro	File type
`S_IFREG`	`S_ISREG()`	Regular file
`S_IFDIR`	`S_ISDIR()`	Directory
`S_IFCHR`	`S_ISCHR()`	Character device
`S_IFBLK`	`S_ISBLK()`	Block device
`S_IFIFO`	`S_ISFIFO()`	FIFO or pipe
`S_IFSOCK`	`S_ISSOCK()`	Socket
`S_IFLNK`	`S_ISLNK()`	Symbolic link

The bottom 12 bits of the *st_mode* field define the permissions for the file. We describe the file permission bits in Section 15.4. For now, we simply note that the 9 least significant of the permission bits are the read, write, and execute permissions for each of the categories owner, group, and other.

File size, blocks allocated, and optimal I/O block size

For regular files, the *st_size* field is the total size of the file in bytes. For a symbolic link, this field contains the length (in bytes) of the pathname pointed to by the link. For a shared memory object (Chapter 54), this field contains the size of the object.

The *st_blocks* field indicates the total number of blocks allocated to the file, in 512-byte block units. This total includes space allocated for pointer blocks (see Figure 14-2, on page 258). The choice of the 512-byte unit of measurement is historical—this is the smallest block size on any of the file systems that have been implemented under UNIX. More modern file systems use larger logical block sizes. For example, under *ext2*, the value in *st_blocks* is always a multiple of 2, 4, or 8, depending on whether the *ext2* logical block size is 1024, 2048, or 4096 bytes.

> SUSv3 doesn't define the units in which *st_blocks* is measured, allowing the possibility that an implementation uses a unit other than 512 bytes. Most UNIX implementations do use 512-byte units, but HP-UX 11 uses file system–specific units (e.g., 1024 bytes in some cases).

The *st_blocks* field records the number of disk blocks actually allocated. If the file contains holes (Section 4.7), this will be smaller than might be expected from the corresponding number of bytes (*st_size*) in the file. (The disk usage command, *du –k file*, displays the actual space allocated for a file, in kilobytes; that is, a figure calculated from the *st_blocks* value for the file, rather than the *st_size* value.)

The *st_blksize* field is somewhat misleadingly named. It is not the block size of the underlying file system, but rather the optimal block size (in bytes) for I/O on files on this file system. I/O in blocks smaller than this size is less efficient (refer to Section 13.1). A typical value returned in *st_blksize* is 4096.

File timestamps

The *st_atime*, *st_mtime*, and *st_ctime* fields contain, respectively, the times of last file access, last file modification, and last status change. These fields are of type *time_t*, the standard UNIX time format of seconds since the Epoch. We say more about these fields in Section 15.2.

Example program

The program in Listing 15-1 uses *stat()* to retrieve information about the file named on its command line. If the *–l* command-line option is specified, then the program instead uses *lstat()* so that we can retrieve information about a symbolic link instead of the file to which it refers. The program prints all fields of the returned *stat* structure. (For an explanation of why we cast the *st_size* and *st_blocks* fields to *long long*, see Section 5.10.) The *filePermStr()* function used by this program is shown in Listing 15-4, on page 296.

Here is an example of the use of the program:

```
$ echo 'All operating systems provide services for programs they run' > apue
$ chmod g+s apue        Turn on set-group-ID bit; affects last status change time
$ cat apue              Affects last file access time
All operating systems provide services for programs they run
$ ./t_stat apue
File type:                regular file
Device containing i-node: major=3   minor=11
I-node number:            234363
Mode:                     102644 (rw-r--r--)
    special bits set:     set-GID
Number of (hard) links:   1
Ownership:                UID=1000   GID=100
File size:                61 bytes
Optimal I/O block size:   4096 bytes
512B blocks allocated:    8
Last file access:         Mon Jun  8 09:40:07 2011
Last file modification:   Mon Jun  8 09:39:25 2011
Last status change:       Mon Jun  8 09:39:51 2011
```

Listing 15-1: Retrieving and interpreting file *stat* information

files/t_stat.c

```
#define _BSD_SOURCE     /* Get major() and minor() from <sys/types.h> */
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include "file_perms.h"
#include "tlpi_hdr.h"

static void
displayStatInfo(const struct stat *sb)
{
    printf("File type:                ");

    switch (sb->st_mode & S_IFMT) {
    case S_IFREG:  printf("regular file\n");            break;
    case S_IFDIR:  printf("directory\n");               break;
    case S_IFCHR:  printf("character device\n");        break;
    case S_IFBLK:  printf("block device\n");            break;
    case S_IFLNK:  printf("symbolic (soft) link\n");    break;
    case S_IFIFO:  printf("FIFO or pipe\n");            break;
    case S_IFSOCK: printf("socket\n");                  break;
    default:       printf("unknown file type?\n");      break;
    }

    printf("Device containing i-node: major=%ld   minor=%ld\n",
            (long) major(sb->st_dev), (long) minor(sb->st_dev));

    printf("I-node number:            %ld\n", (long) sb->st_ino);

    printf("Mode:                     %lo (%s)\n",
        (unsigned long) sb->st_mode, filePermStr(sb->st_mode, 0));

    if (sb->st_mode & (S_ISUID | S_ISGID | S_ISVTX))
        printf("    special bits set:     %s%s%s\n",
            (sb->st_mode & S_ISUID) ? "set-UID " : "",
            (sb->st_mode & S_ISGID) ? "set-GID " : "",
            (sb->st_mode & S_ISVTX) ? "sticky " : "");

    printf("Number of (hard) links:   %ld\n", (long) sb->st_nlink);

    printf("Ownership:                UID=%ld   GID=%ld\n",
        (long) sb->st_uid, (long) sb->st_gid);

    if (S_ISCHR(sb->st_mode) || S_ISBLK(sb->st_mode))
        printf("Device number (st_rdev):  major=%ld; minor=%ld\n",
            (long) major(sb->st_rdev), (long) minor(sb->st_rdev));

    printf("File size:                %lld bytes\n", (long long) sb->st_size);
    printf("Optimal I/O block size:   %ld bytes\n", (long) sb->st_blksize);
    printf("512B blocks allocated:    %lld\n", (long long) sb->st_blocks);
    printf("Last file access:         %s", ctime(&sb->st_atime));
    printf("Last file modification:   %s", ctime(&sb->st_mtime));
    printf("Last status change:       %s", ctime(&sb->st_ctime));
}

int
main(int argc, char *argv[])
{
    struct stat sb;
    Boolean statLink;           /* True if "-l" specified (i.e., use lstat) */
    int fname;                  /* Location of filename argument in argv[] */

    statLink = (argc > 1) && strcmp(argv[1], "-l") == 0;
                                /* Simple parsing for "-l" */
    fname = statLink ? 2 : 1;

    if (fname >= argc || (argc > 1 && strcmp(argv[1], "--help") == 0))
        usageErr("%s [-l] file\n"
                "        -l = use lstat() instead of stat()\n", argv[0]);

    if (statLink) {
        if (lstat(argv[fname], &sb) == -1)
            errExit("lstat");
    } else {
        if (stat(argv[fname], &sb) == -1)
            errExit("stat");
    }

    displayStatInfo(&sb);

    exit(EXIT_SUCCESS);
}
```
files/t_stat.c

15.2 File Timestamps

The *st_atime, st_mtime*, and *st_ctime* fields of the *stat* structure contain file timestamps. These fields record, respectively, the times of last file access, last file modification, and last file status change (i.e., last change to the file's i-node information). Timestamps are recorded in seconds since the Epoch (1 January 1970; see Section 10.1).

Most native Linux and UNIX file systems support all of the timestamp fields, but some non-UNIX file systems may not.

Table 15-2 summarizes which of the timestamp fields (and in some cases, the analogous fields in the parent directory) are changed by various system calls and library functions described in this book. In the headings of this table, *a*, *m*, and *c* represent the *st_atime*, *st_mtime*, and *st_ctime* fields, respectively. In most cases, the relevant timestamp is set to the current time by the system call. The exceptions are *utime()* and similar calls (discussed in Sections 15.2.1 and 15.2.2), which can be used to explicitly set the last file access and modification times to arbitrary values.