【Linux Programming】Day9

Class	Understanding Linux/Unix Programming
Date	@May 11, 2022

【Ch3】Directories and File Properties

3.6.3 The stat Call Gets File Information

The following figure shows how `stat` works:

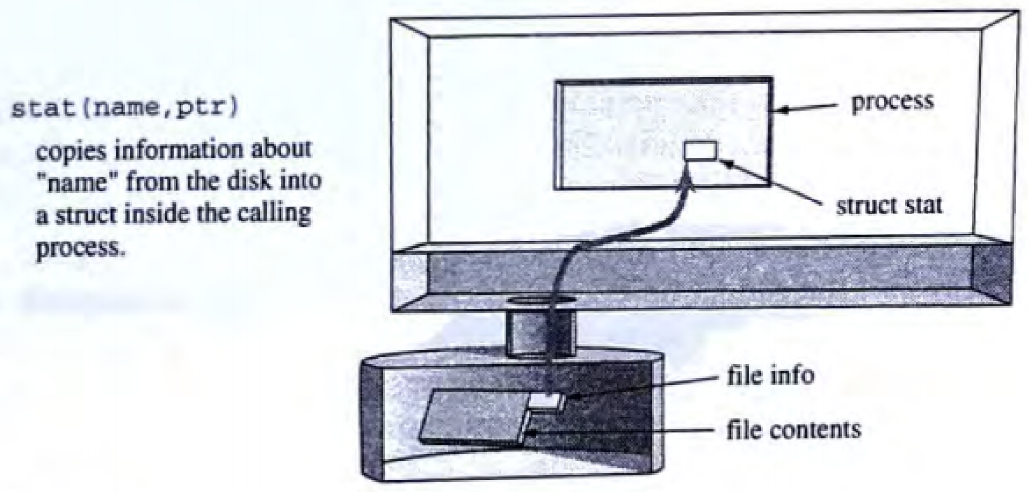
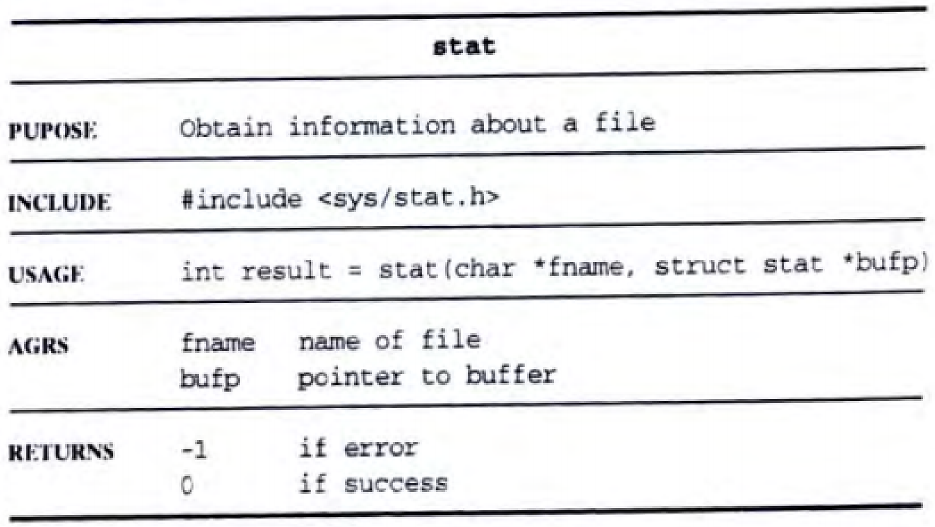

stat(name,ptr)
copies information about "name" from the disk into a struct inside the calling process.
process
struct stat
file info
file contents

FIGURE 3.3

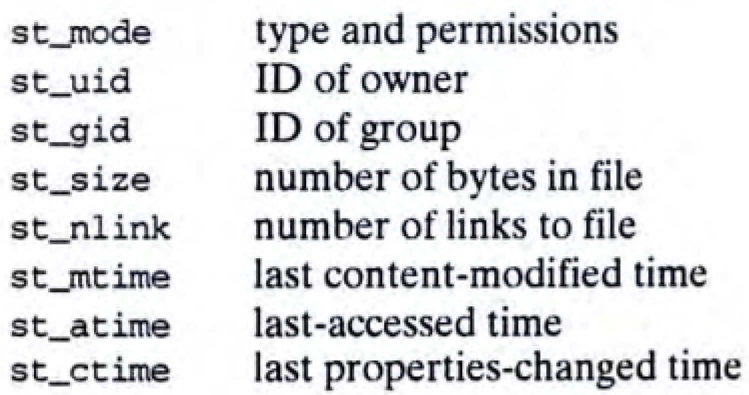
Reading file properties using stat.

A file is stored on the disk. A file has contents, and a file has a set of attributes: size, owner ID, etc.



stat		
PUPOSE	Obtain information about a file	
INCLUDE	#include <sys/stat.h>	
USAGE	int result = stat(char *fname, struct stat *bufp)	
AGRS	fname	name of file
	bufp	pointer to buffer
RETURNS	-1	if error
	0	if success

The man page for stat describe the members of `struct stat`:



st_mode	type and permissions
st_uid	ID of owner
st_gid	ID of group
st_size	number of bytes in file
st_nlink	number of links to file
st_mtime	last content-modified time
st_atime	last-accessed time
st_ctime	last properties-changed time

3.6.6 Converting File Mode to a String

What is the connection between the octal number 100664 and the string `-rw-rw-r--`?

`st_mode` is a 16-bit quantity. Separate attribuates are encoded in substrings of these 16 bits. The following figure shows five coding substrings:

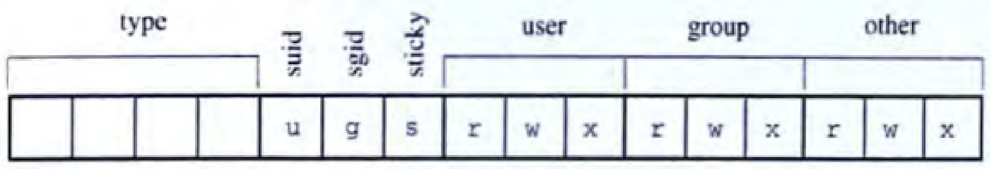


FIGURE 3.4

File type and access coding.

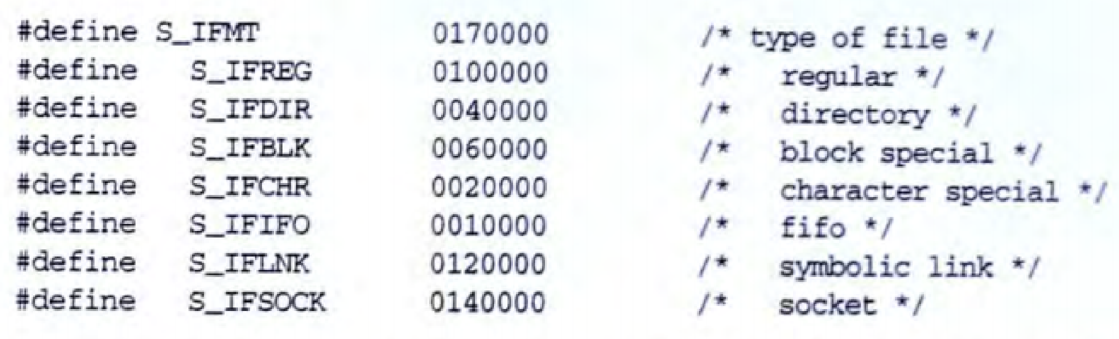
The first four bits represent the type of the file. The next three bits are used for special attributes of a file. Each bit corresponds to a special attribute; a '1' turns the attribute on, a '0' turns it off. These set-user-ID, set-group-ID, and sticky bits will be discussed later. Finally, there are three sets of permission bits.

Thing One: The Conecept of Masking

Masking a value is zeroing out bits in the number so only a subfield is unaffected.

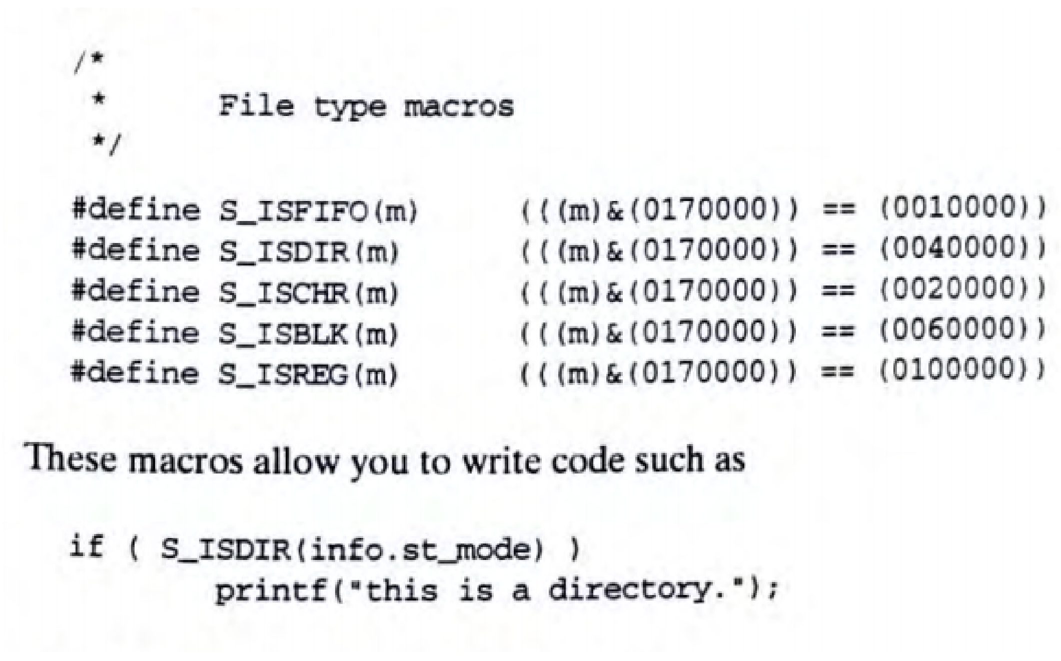
Using Masking to Decode File Types

These definitions are in `<sys/stat.h>`:



```
#define S_IFMT           0170000        /* type of file */
#define   S_IFREG        0100000        /*   regular */
#define   S_IFDIR        0040000        /*   directory */
#define   S_IFBLK        0060000        /*   block special */
#define   S_IFCHR        0020000        /*   character special */
#define   S_IFIFO        0010000        /*   fifo */
#define   S_IFLNK        0120000        /*   symbolic link */
#define   S_IFSOCK       0140000        /*   socket */
```

We can use macros to check the property of the file:



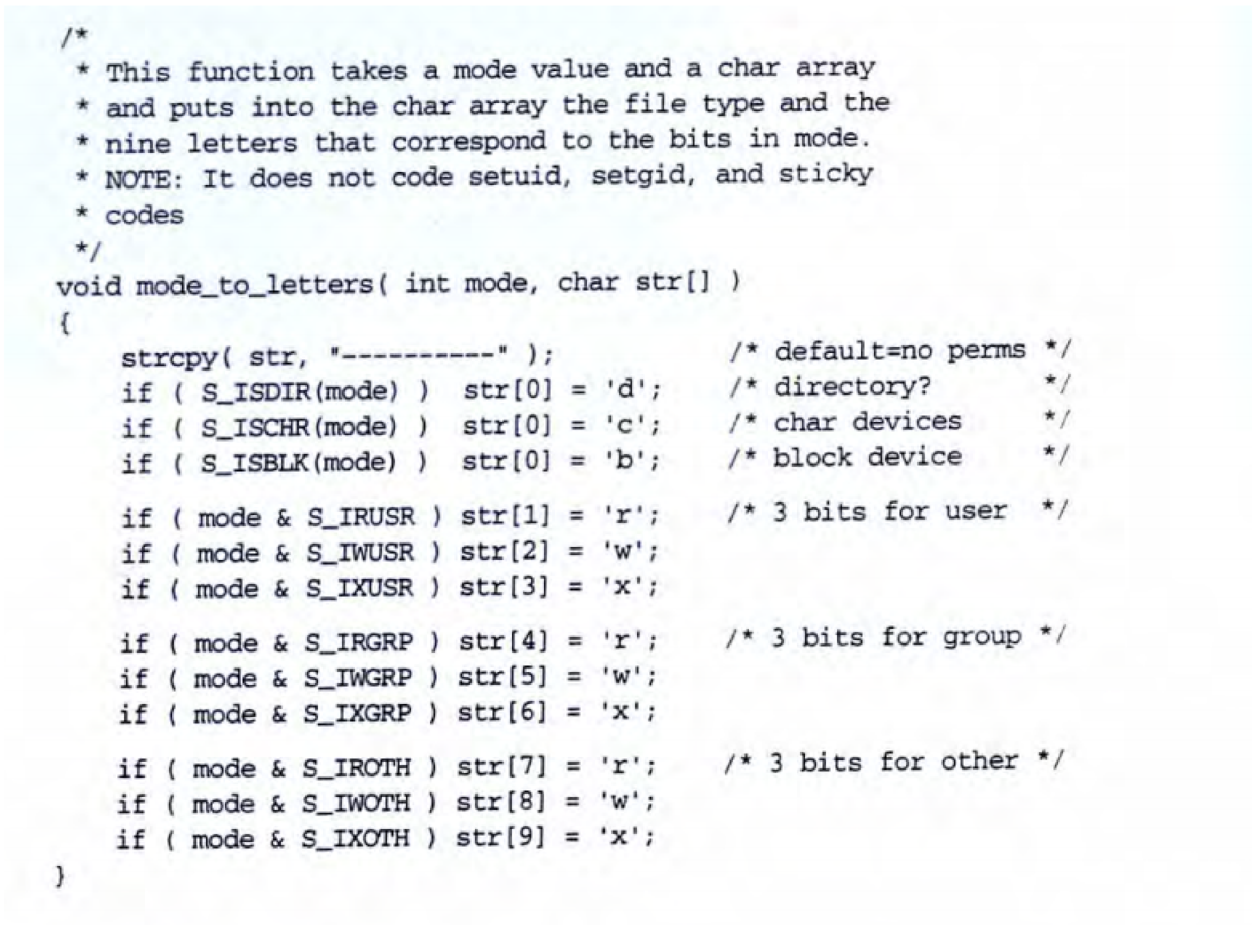
```
/*
 *      File type macros
 */

#define S_ISFIFO(m)      (((m)&(0170000)) == (0010000))
#define S_ISDIR(m)       (((m)&(0170000)) == (0040000))
#define S_ISCHR(m)       (((m)&(0170000)) == (0020000))
#define S_ISBLK(m)       (((m)&(0170000)) == (0060000))
#define S_ISREG(m)       (((m)&(0170000)) == (0100000))
```

These macros allow you to write code such as

```
if ( S_ISDIR(info.st_mode) )
        printf("this is a directory.");
```

Write `mode_to_letters`:



```
/*
 * This function takes a mode value and a char array
 * and puts into the char array the file type and the
 * nine letters that correspond to the bits in mode.
 * NOTE: It does not code setuid, setgid, and sticky
 * codes
 */
void mode_to_letters( int mode, char str[] )
{
    strcpy( str, "----------" );          /* default=no perms */
    if ( S_ISDIR(mode) )  str[0] = 'd';    /* directory?       */
    if ( S_ISCHR(mode) )  str[0] = 'c';    /* char devices     */
    if ( S_ISBLK(mode) )  str[0] = 'b';    /* block device     */

    if ( mode & S_IRUSR ) str[1] = 'r';    /* 3 bits for user  */
    if ( mode & S_IWUSR ) str[2] = 'w';
    if ( mode & S_IXUSR ) str[3] = 'x';

    if ( mode & S_IRGRP ) str[4] = 'r';    /* 3 bits for group */
    if ( mode & S_IWGRP ) str[5] = 'w';
    if ( mode & S_IXGRP ) str[6] = 'x';

    if ( mode & S_IROTH ) str[7] = 'r';    /* 3 bits for other */
    if ( mode & S_IWOTH ) str[8] = 'w';
    if ( mode & S_IXOTH ) str[9] = 'x';
}
```