System Programming
(ELEC462)

Directories and File Properties

Dukyun Nam

HPC Lab@KNU

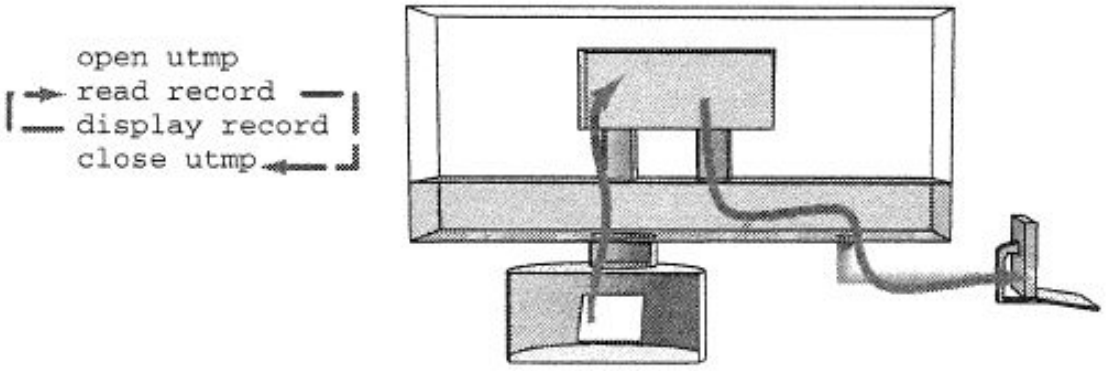
Contents

Introduction

- Recall
  - Last time, we looked at ways to operate on the CONTENTS of a file:

    `open, read, write, lseek, close`

- But, there is more to a file than just contents:
  - 1) Properties
    - Size, owner, permission, type, etc.
  - 2) Location in a directory tree
- To learn about these:
  - We shall write a version of `ls`



< Data flow in the `who` command >

What Does `ls` Do?

- Type the command `ls` to see what it does:

```
$ ls
Makefile    docs        ls2.c       s.tar       statdemo.c  tail1.c
chap03      ls1.c       old_src     stat1.c     tail1
$
```

- The default action of '`ls`':
  - To list the names of the files in the "current" directory

What Does `ls` Do? (cont.)

- When given the '`-l`' command-line option
  - `ls` present information about each file and its properties using the *long format*

```
$ ls -l
total 108
-rw-rw-r--   2 bruce    users         345 Jul 29 11:05 Makefile
-rw-rw-r--   1 bruce    users       27521 Aug  1 12:14 chap03
drwxrwxr-x   2 bruce    users        1024 Aug  1 12:15 docs
-rw-r--r--   1 bruce    users         723 Feb  9  1998 ls1.c
-rw-r--r--   1 bruce    users        3045 Feb 15 03:51 ls2.c
drwxrwxr-x   2 bruce    users        1024 Aug  1 12:14 old_src
-rw-rw-r--   1 bruce    users       30720 Aug  1 12:05 s.tar
-rw-r--r--   1 bruce    support       946 Feb 18 17:15 stat1.c
-rw-r--r--   1 bruce    support       191 Feb  9  1998 statdemo.c
-rwxrwxr-x   1 bruce    users       37351 Aug  1 12:13 tail1
-rw-r--r--   1 bruce    users        1416 Aug  1 12:05 tail1.c
$
```

What Does `ls` Do? (cont.)

- Listing other directories

Asking `ls` about Other Directories and Their Files

Example	Action
`ls /tmp`	list names of files in `/tmp` directory
`ls -l docs`	show attributes of files in `docs` directory
`ls -l ../Makefile`	show attributes of `../Makefile`
`ls *.c`	list names of files matching pattern `*.c`

  - If an argument is a "directory," `ls` lists its contents
  - If an argument is a "file," `ls` lists its name or attributes
    - What command line option is specified affects what `ls` does and what it shows

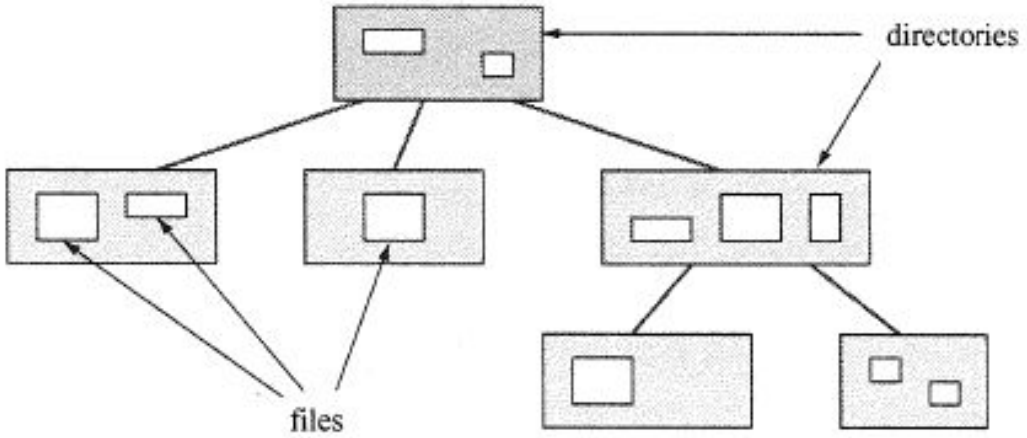
What Does `ls` Do? (cont.)

- Popular command-line options

Command	Action
`ls -a`	shows "."-files
`ls -lu`	shows last-read time
`ls -s`	shows size in blocks
`ls -t`	sorts in time order
`ls -F`	shows file types

  - Use `man` to see more options for `ls`
  - Why don't we run `ls` with different options? What can you see?

Brief Review of the File System Tree

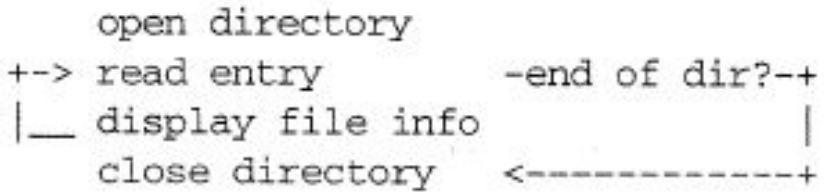


< A tree of directories >

- Listing other directories
  - A disk is organized as a tree of directories, each of which contains files or directories
  - In Unix/Linux, every file on the system is located somewhere in a single tree of directories
  - This simplicity simplifies writing `ls`: only thinking about directories and files

How Does `ls` Work?

- What is a directory?



  - A directory is *a kind of file* that contains a list of names of files and directories
  - Unlike a regular file, a directory is never empty
  - Every directory contains two specific items
    - *dot* (.) is the name of the current directory
    - *dotdot* (..) is the name of the directory one level up

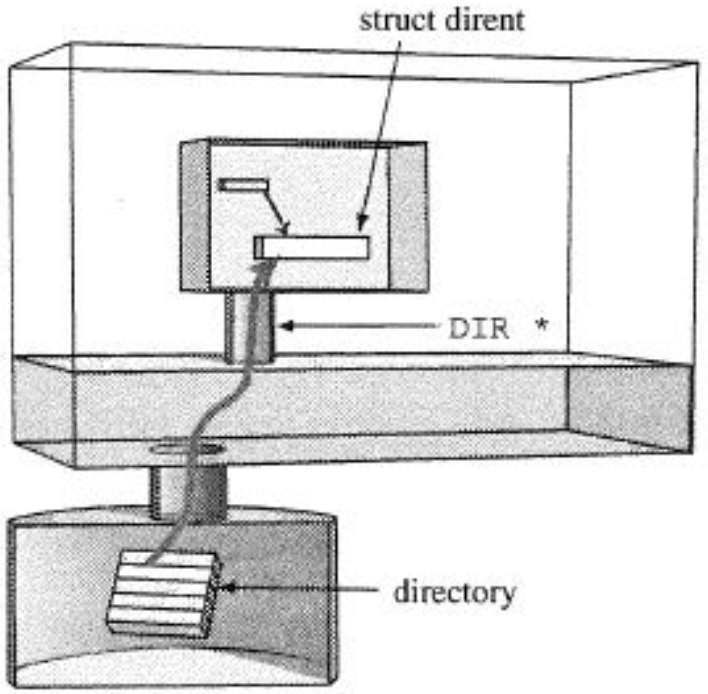
How Does `ls` Work? (cont.)

- Reading the contents of a directory

```
opendir(char *)
  creates a connection,
  returns a DIR *

readdir(DIR *)
  reads next record,
  returns a pointer
  to a struct dirent

closedir(DIR *)
  closes a connection
```



< Reading entries from a directory >

How Does `ls` Work? (cont.)

- The `dirent` structure
  - Typically defined `/usr/include/dirent.h`

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

    - An inode (index node) serves as a unique identifier for a specific piece of **metadata** on a given filesystem
    - **d_name** contains the null-terminated filename

Can I Write `ls`? (`ls1.c`)

- Logic for listing a directory:

```
main()
        opendir
        while ( readdir )
             print d_name
        closedir
```

```c
/** ls1.c
 **   purpose  list contents of directory or directories
 **   action   if no args, use .  else list files in args
 **/
#include        <stdio.h>
#include        <sys/types.h>
#include        <dirent.h>

void do_ls(char []);

int main(int ac, char *av[])
{
        if ( ac == 1 )
                do_ls( "." );
        else
                while ( --ac ){
                        printf("%s:\n", *++av );
                        do_ls( *av );
                }

        return 0;
}
```

```c
void do_ls( char dirname[] )
/*
 *      list files in directory called dirname
 */
{
        DIR             *dir_ptr;               /* the directory */
        struct dirent   *direntp;               /* each entry    */

        if ( ( dir_ptr = opendir( dirname ) ) == NULL )
                fprintf(stderr,"ls1: cannot open %s\n", dirname);
        else
        {
                while ( ( direntp = readdir( dir_ptr ) ) != NULL )
                        printf("%s\n", direntp->d_name );
                closedir(dir_ptr);
        }
}
```

Writing `ls -l`

- `ls -l` operation
  - Mode: type of file (-, d) + permissions (user, group, everyone (other))
  - Links: reference to a file (often used in Linux)
  - Owner
  - Group
  - Size
  - Last-modified
  - Name

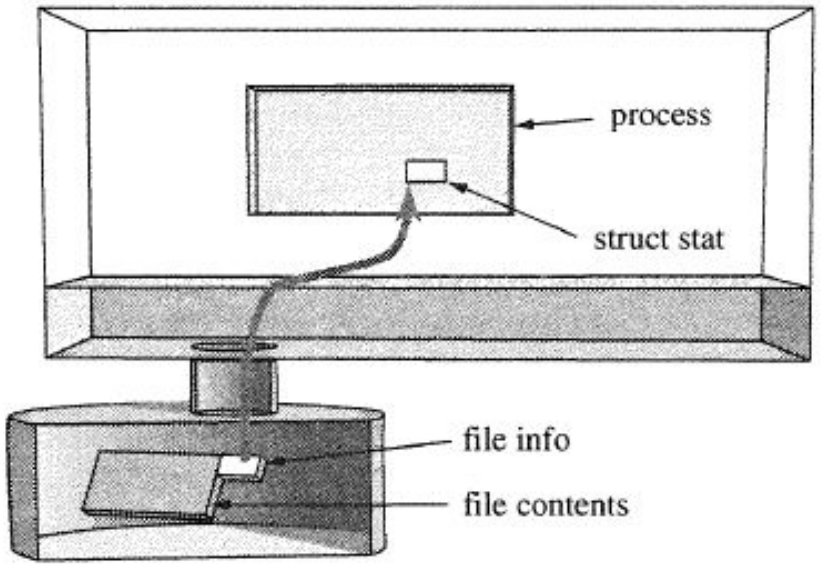
```
$ ls -l
total 108
-rw-rw-r--   2 bruce    users        345 Jul 29 11:05 Makefile
-rw-rw-r--   1 bruce    users      27521 Aug  1 12:14 chap03
drwxrwxr-x   2 bruce    users       1024 Aug  1 12:15 docs
-rw-r--r--   1 bruce    users        723 Feb  9  1998 ls1.c
-rw-r--r--   1 bruce    users       3045 Feb 15 03:51 ls2.c
drwxrwxr-x   2 bruce    users       1024 Aug  1 12:14 old_src
-rw-rw-r--   1 bruce    users      30720 Aug  1 12:05 s.tar
-rw-r--r--   1 bruce    support      946 Feb 18 17:15 stat1.c
-rw-r--r--   1 bruce    support      191 Feb  9  1998 statdemo.c
```

Writing `ls -l` (cont.)

- Needs a system call, called "`stat`":

`stat(name,ptr)`

copies information about "name" from the disk into a struct inside the calling process.



< Reading file properties using `stat` >

  - Obtains the information about a given file
  - Interface: `stat(name, ptr)`

Writing `ls -l` (cont.)

- System call "`stat`":

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

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

Writing `ls -l` (cont.)

- '`fileinfo.c`' : to show file properties via `stat()`

```c
/* statinfo.c - demonstrates using stat() to obtain
 *              file information.
 *            - some members are just numbers...
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>

void show_stat_info(char *, struct stat *);

int main(int ac, char *av[])
{
    struct stat info;        /* buffer for file info */

    if (ac>1)
        if( stat(av[1], &info) != -1 ){
            show_stat_info( av[1], &info );
            return 0;
        }
        else
            perror(av[1]);  /* report stat() errors  */
    return 0;
}
```

```c
void show_stat_info(char *fname, struct stat *buf)
/*
 * displays some info from stat in a name=value format
 */
{
    printf("   mode: %o\n", buf->st_mode);         /* type + mode */
    printf("  links: %ld\n", buf->st_nlink);        /* # links     */
    printf("   user: %d\n", buf->st_uid);          /* user id     */
    printf("  group: %d\n", buf->st_gid);          /* group id    */
    printf("   size: %ld\n", buf->st_size);        /* file size   */
    printf("modtime: %ld\n", buf->st_mtime);       /* modified    */
    printf("   name: %s\n", fname );               /* filename    */
}
```

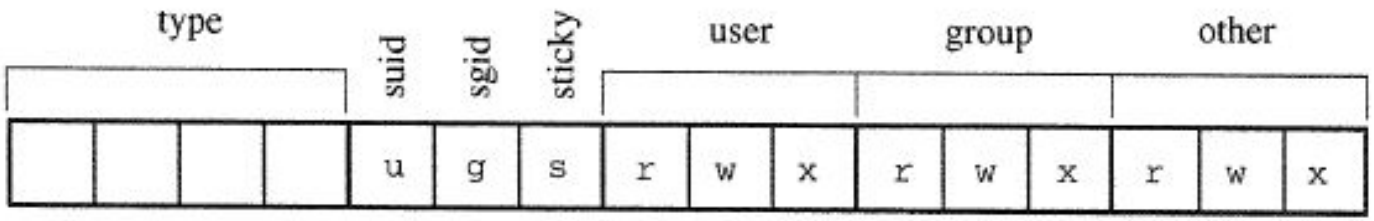
Writing `ls -l` (cont.)

- Results of `fileinfo`

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ ./fileinfo fileinfo.c
   mode: 100644
  links: 1
   user: 1000
  group: 1000
   size: 1155
modtime: 1663595693
   name: fileinfo.c
dynam@DESKTOP-Q4IJBP7:~/lab4$ ls -l fileinfo.c
-rw-r--r-- 1 dynam dynam 1155 Sep 19 22:54 fileinfo.c
```

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ cat /etc/passwd | grep dynam
dynam:x:1000:1000:,,,:/home/dynam:/bin/bash
```

Converting File Mode to a String



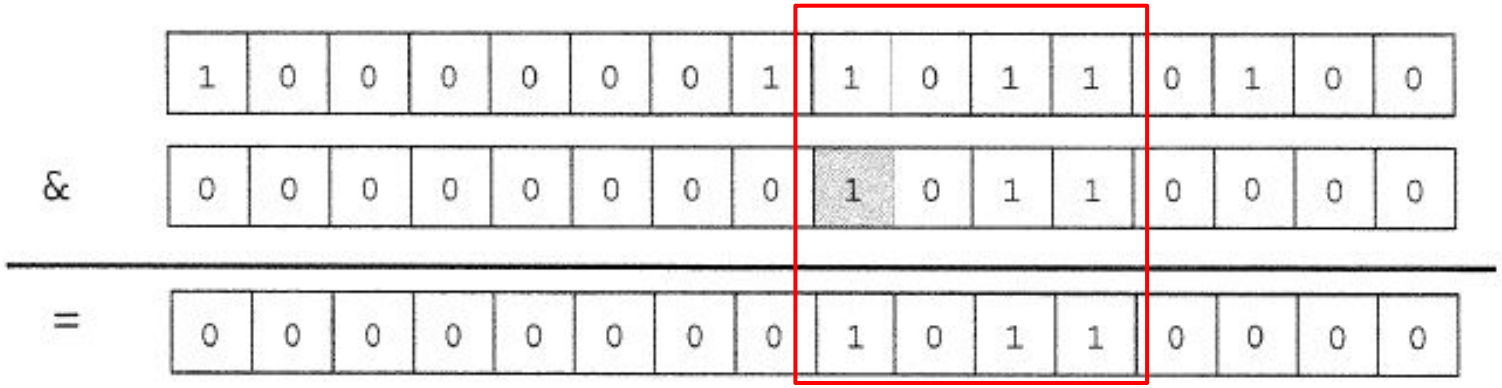
- `st_mode` includes field type and permission bits
- We convert file mode into a 16-bit integer, called *subfield coding*
- To read subfields, we need to use a technique, called *masking*
  - Zeroing out digits (making zeros), only subfield show through
  - Set marks to translate st_mode into string displayed by `ls -l`

Converting File Mode to a String (cont.)

- Subfield coding

Examples of Subfield Coding	
617-495-4204	area, exchange, line
027-93-1111	social security number
128.103.33.100	IP numbers

- Masking



  - By masking a bit, focus will be reduced; unnecessary information is hidden.

Converting File Mode to a String (cont.)

- Using ***masking*** to decode file type and permission bits

```
#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 */
```

```
/*
 *      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))
```

```
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';
}
```

Converting User/Group ID to Strings

- Users
  - `/etc/passwd` is the list of users
    - But that does not always present a complete list of users
  - A library function `getpwuid` provides access to the complete list of users.
- Groups
  - `/etc/group` is the list of group
  - A user can belong to more than one group
  - Function `getgrgid` provides access to list of group

Putting It All Together: `ls2.c`

```c
/* ls2.c
 *      purpose  list contents of directory or directories
 *      action   if no args, use .  else list files in args
 *      note     uses stat and pwd.h and grp.h
 *      BUG: try ls2 /tmp
 */
#include        <stdio.h>
#include        <sys/types.h>
#include        <dirent.h>
#include        <sys/stat.h>
#include        <string.h>

void do_ls(char[]);
void dostat(char *);
void show_file_info( char *, struct stat *);
void mode_to_letters( int , char [] );
char *uid_to_name( uid_t );
char *gid_to_name( gid_t );

int main(int ac, char *av[])
{
        if ( ac == 1 )
                do_ls( "." );
        else
                while ( --ac ){
                        printf("%s:\n", *++av );
                        do_ls( *av );
                }

        return 0;
}
```

```c
void do_ls( char dirname[] )
/*
 *      list files in directory called dirname
 */
{
        DIR             *dir_ptr;               /* the directory */
        struct dirent   *direntp;               /* each entry    */

        if ( ( dir_ptr = opendir( dirname ) ) == NULL )
                fprintf(stderr,"ls1: cannot open %s\n", dirname);
        else
        {
                while ( ( direntp = readdir( dir_ptr ) ) != NULL )
                        dostat( direntp->d_name );
                closedir(dir_ptr);
        }
}

void dostat( char *filename )
{
        struct stat info;

        if ( stat(filename, &info) == -1 )              /* cannot stat    */
                perror( filename );                     /* say why        */
        else                                    /* else show info         */
                show_file_info( filename, &info );
}
```

Putting It All Together: `ls2.c` (cont.)

```
void show_file_info( char *filename, struct stat *info_p )
/*
 * display the info about 'filename'.  The info is stored in struct at *info_p
 */
{
        char    *uid_to_name(), *ctime(), *gid_to_name(), *filemode();
        void    mode_to_letters();
        char    modestr[11];

        mode_to_letters( info_p->st_mode, modestr );

        printf( "%s"    , modestr );
        printf( "%4d "  , (int) info_p->st_nlink);
        printf( "%-8s " , uid_to_name(info_p->st_uid) );
        printf( "%-8s " , gid_to_name(info_p->st_gid) );
        printf( "%8ld " , (long)info_p->st_size);
        printf( "%.12s ", 4+ctime(&info_p->st_mtime));
        printf( "%s\n"  , filename );

}
```

Putting It All Together: `ls2.c` (cont.)

```c
/*
 * 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';
}
```

Putting It All Together: `ls2.c` (cont.)

```c
#include        <pwd.h>

char *uid_to_name( uid_t uid )
/*
 *      returns pointer to username associated with uid, uses getpw()
 */
{
        struct  passwd *getpwuid(), *pw_ptr;
        static  char numstr[10];

        if ( ( pw_ptr = getpwuid( uid ) ) == NULL ){
                sprintf(numstr,"%d", uid);
                return numstr;
        }
        else
                return pw_ptr->pw_name ;
}
```

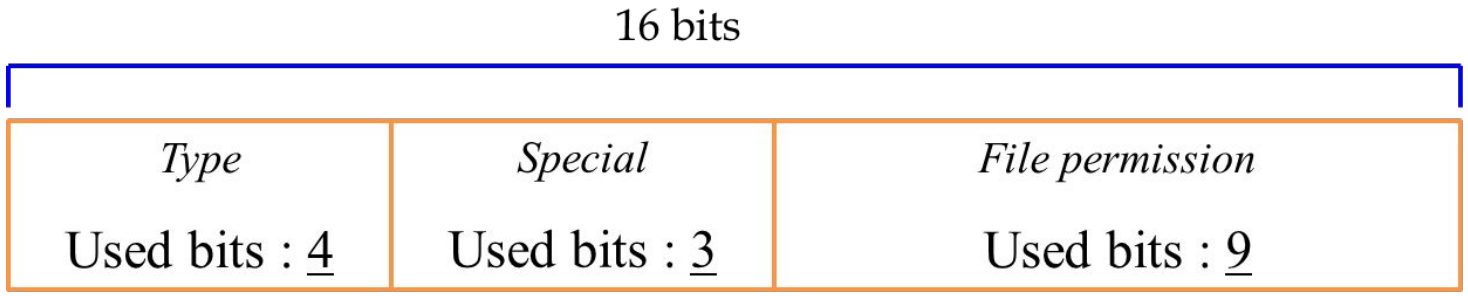
```c
#include        <grp.h>

char *gid_to_name( gid_t gid )
/*
 *      returns pointer to group number gid. used getgrgid(3)
 */
{
        struct group *getgrgid(), *grp_ptr;
        static  char numstr[10];

        if ( ( grp_ptr = getgrgid(gid) ) == NULL ){
                sprintf(numstr,"%d", gid);
                return numstr;
        }
        else
                return grp_ptr->gr_name;
}
```

The Three Special Bits

- The `st_mode` member of the `stat` structure contains 16 bits



- The three ***special*** bits
  - The Set-User-ID bit
  - The Set-Group-ID bit
  - The Sticky bit

1. The Set-User-ID Bit

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ ls -asl /usr/bin/passwd
68 -rwsr-xr-x 1 root root 68208 Mar 14  2022 /usr/bin/passwd
```

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ ls -asl /etc/passwd
4 -rw-r--r-- 1 root root 2939 Sep 13 01:33 /etc/passwd
```

- That SUID bit tells the kernel to run the program as though it were being run by the owner of the program: let's type 'passwd.'
  - Note that this program belongs to "root" as owner and "root" as group.
- The name 'set user ID' comes from the fact that the bit causes the kernel to set the *effective user ID* to the user ID of the owner of the program
  - User `root` owns `/etc/passwd`, so a program running as root can modify the file

2. The Set-Group-ID Bit

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ ls -asl /usr/bin/wall
36 -rwxr-sr-x 1 root tty 35048 Feb  7  2022 /usr/bin/wall
```

- The second special bit sets the *effective group ID* of a program
  - Effective group ID: a number assigned by the kernel to a group
  - If a program belongs to group *g* and the set group ID bit is set, then the program runs as though it were being run by a member of *g*
  - This bit grants the program the access rights of members of that group

3. The Sticky Bit

- For files
  - The sticky bit tells the kernel to keep the program on the “swap device” (somewhere on disk) even if nobody was using it right now
    - The program gets never fragmented on the swap device on the disk
  - If the bit is “unset,” then the program might be split into many small sections scattered across the disk
    - So, the loading time of the program may get longer than when the bit is unset
  - Typically, loading a program from the swap device is faster than that from the regular section of the disk
    - By having the bit set to the file associated with the program, we can load the program quicker than otherwise

3. The Sticky Bit (cont.)

```
dynam@DESKTOP-Q4IJBP7:~/lab4$ chmod +t /tmp/sticky_bit_test
dynam@DESKTOP-Q4IJBP7:~/lab4$ ls -l /tmp
total 32
drwx------ 3 root  root  4096 Sep 20 08:39 snap.lxd
drwxrwxr-t 2 dynam dynam 4096 Sep 22 03:27 sticky_bit_test
```

- For directories
  - Some directories are designed to hold “temporary” files
  - These scratch (temp) directories, notably `/tmp`, are publicly writable, allowing any user to create and delete any files there
  - Setting the sticky bit overrides the publicly writable attribute for a directory
    - To avoid deletion of a folder and its content by other users
  - Files in the directory with the sticky bit set may only be deleted by their owners
    - “No one can delete things in my directory!”

Setting and Modifying the Properties of a File

- 1) Type of a file
  - A file has a type
    - It can be a regular file(-), a directory(d), a device file(b), a socket(s), a symbolic link(l), or a named pipe(p) (connected processes by a pipe)
  - Establishing the “type” of a file
    - The type of the file is determined when the file is created
      - For example, the `creat` system call creates a regular file
      - Different system calls are used to create directories, devices, and the like
  - Changing the type of a file is impossible unless the file is recreated

Setting and Modifying the Properties of a File (cont.)

- 2) Permission bits and special bits
  - Establishing the mode of a file

```
fd = creat( "newfile", 0744 );
```

  - Changing the mode of a file

```
  chmod( "/tmp/myfile", 04764 );
and
  chmod( "/tmp/myfile", S_ISUID | S_IRWXU | S_IRGRP|S_IWGRP | S_IROTH );
```

chmod		
PURPOSE	Change permission and special bits for a file	
INCLUDE	#include <sys/types.h> #include <sys/stat.h>	
USAGE	int result = chmod(char *path, mode_t mode);	
ARGS	path	path to file
	mode	new value for mode
RETURNS	-1	if error
	0	if success

Setting and Modifying the Properties of a File (cont.)

- 3) Number of links to a file
  - Represents the number of times the file is referenced in directories (or somewhere in the filesystem)

Setting and Modifying the Properties of a File (cont.)

- 4) Changing the owner and group of a file
  - A program can modify the owner and group of a file by making the chown system call

chown		
PURPOSE	Change owner and or group ID of a file	
INCLUDE	#include <unistd.h>	
USAGE	int chown(char *path, uid_t owner, gid_t group)	
ARGS	path	path to file
	owner	user ID for file
RETURNS	group	group ID for file
	-1	if error
	0	if success

Setting and Modifying the Properties of a File (cont.)

- 5) Size of a file
  - The size of a file, directory, and named pipe represents the number of bytes stored
- 6) Modification and access time

utime		
PURPOSE	Change access and modification time for files	
INCLUDE	#include <sys/time.h> #include <utime.h>	
USAGE	#include <sys/types.h> int utime(char *path, struct utimbuf *newtimes)	
ARGS	path newtimes	path to file pointer to a struct utimbuf see utime.h for details
RETURNS	-1 0	if error if success

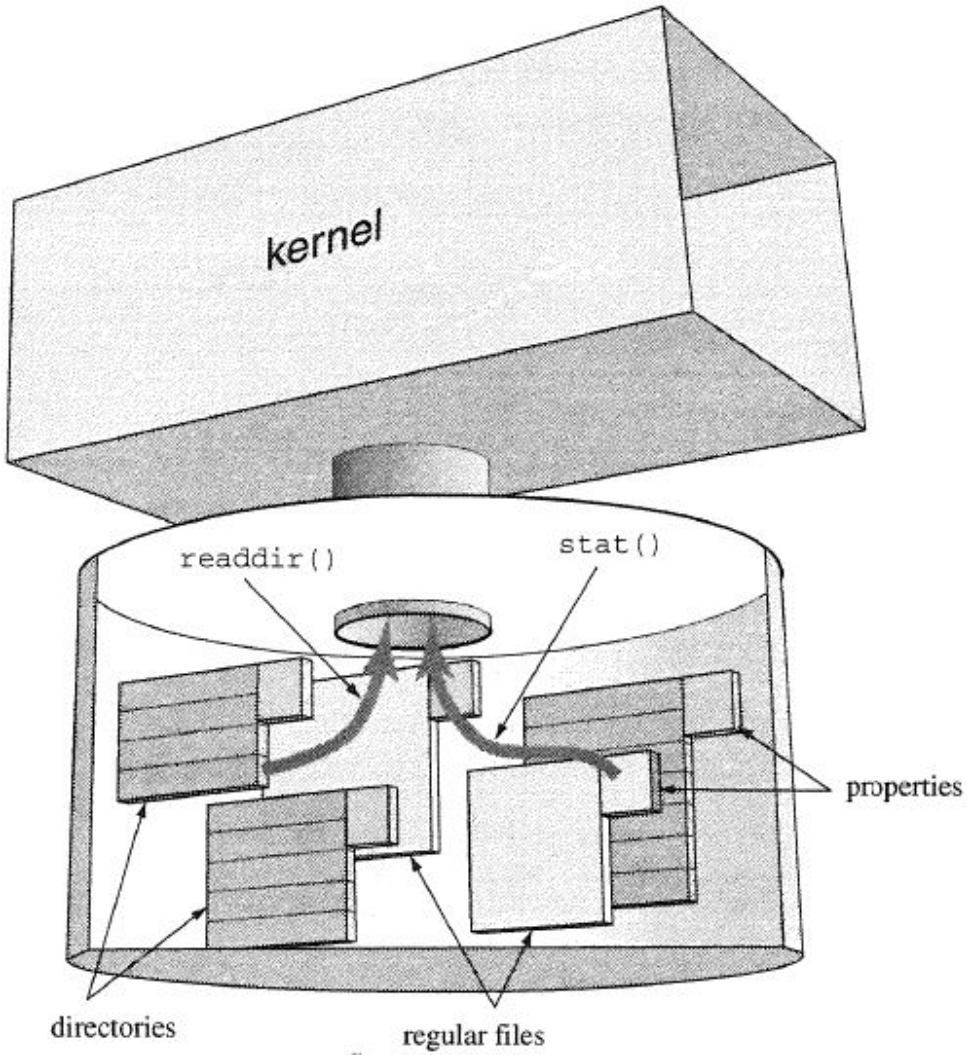
Setting and Modifying the Properties of a File (cont.)

- 7) Name of a file
  - Changing the name of a file is the ‘rename’ system call

rename	
PURPOSE	Change name and/or move a file
INCLUDE	#include <stdio.h>
USAGE	int result = rename(char *old, char *new)
ARGS	old old name of file or directory new new pathname for file or directory
RETURNS	-1 if error 0 if success

Summary

- Chapter 3 continues the discussion about files, showing there is more to files than their contents.
- Files are stored in directories, and files have attributes.
- Chapter 3 shows how to read a directory, and explains how to read, modify, and understand file attributes.
- The project for the chapter is to understand and write a version of the ls command.



Appendix

`gdb` Debugger

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution
- Errors like segmentation faults may be easier to find with the help of gdb
- Online manual
  - https://sourceware.org/gdb/current/onlinedocs/gdb/

gdb Debugger

- GDB: interactive debugger
  - Allows the user to run a program and interactively examine its execution.
  - Features include:
    - breakpoints (“run until control reaches here, then prompt user”)
    - stack backtrace (chain of calls leading to some point in the code)
    - examination of program variables
- Compile option for debug
  - You have to use “–g” option when you compile a code
  - The (g)cc -g flag tells (g)cc to generate and embed debug info.
- Running gdb
  - `$ gdb program`
  - Caution: argument(program) is a program NOT a source code

gdb Commands

- `run [arglist]`
  - Run a program with a new argument (abbreviated command: r)
- `break {[file:] function | line number }`
  - Stop at a line that a user specifies for break. (abbreviated command: b)
- `continue`
  - Continue to run a program (abbreviated command: c)
- `quit`
  - Quit gdb (abbreviated command: q)

`gdb` Commands (cont.)

- `step`
  - Execute a line; if the current line has a function call, then run into the function and execute each line of the function at a time (abbreviated command: s)
- `next`
  - Continue to the next source line; if that line has a function call, then execute the function without stopping (abbreviated command: n)
- `print` *`expr`*
  - Perform the ‘expr’ expression and then print its value (outcome) (abbreviated: p).
- `display`
  - Enable automatic displaying of certain expressions each time GDB stops at a breakpoint or after a step
- `list [first, last]`
  - Print lines from first to last: 10 lines by default

gdb Hands-on with ls1

- Compile a program with –g option

```
$ gcc -g -o ls1 ls1.c
```

- Starting up gdb

```
$ gdb ls1
```

- Running the program

```
(gdb) run
```

- Setting breakpoints

```
(gdb) break ls1.c:37
(gdb) b do_ls
```