Objectives

- Ideas and skills
  - A directory is list of files
  - How to read a directory
  - Types of files and how to determine the type of a file
  - Properties of files and how to determine properties of a file
  - Bit sets and bit masks
  - User and group ID numbers and the passwd database
- System calls and functions
  - opendir, readdir, closedir, seekdir
  - stat, rename
  - chmod, chown, utime
- Commands
  - ls

Question 1: What does ls do?

ls lists the names of files and reports file attributes

```
[wch@localhost chapter3]$ ls
chapter3.ppt          slides_chapter3.dvi  slides_chapter3.tex
slides_chapter3.aux   slides_chapter3.log  slides_chapter3.tex~
slides_chapter3.bbl   slides_chapter3.pdf  slides_chapter3.tex.backup
slides_chapter3.blg   slides_chapter3.ps

[wch@localhost chapter3]$ ls -l
total 204
-rw-r--r--  1 wch users 27136 Feb 27 20:13 chapter3.ppt
-rw-r--r--  1 wch users   128 Feb 27 20:36 slides_chapter3.aux
-rw-r--r--  1 wch users     0 Feb 27 20:36 slides_chapter3.bbl
-rw-r--r--  1 wch users   970 Feb 27 20:36 slides_chapter3.blg
-rw-r--r--  1 wch users  3136 Feb 27 20:36 slides_chapter3.dvi
-rw-r--r--  1 wch users 10677 Feb 27 20:36 slides_chapter3.log
-rw-r--r--  1 wch users 14719 Feb 27 20:36 slides_chapter3.pdf
-rw-r--r--  1 wch users 74053 Feb 27 20:36 slides_chapter3.ps
-rw-r--r--  1 wch users  2518 Feb 27 20:36 slides_chapter3.tex
-rw-r--r--  1 wch users  2517 Feb 27 20:36 slides_chapter3.tex~
-rw-r--r--  1 wch users  2181 Feb 27 20:28 slides_chapter3.tex.backup
```

Question 1: What does ls do?

ls lists other directories, report on other files

```
-------------------------------------------------------------
Asking ls about other directories and files
-------------------------------------------------------------
Example                 Action
-------------------------------------------------------------
ls /tmp                 list name 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
-------------------------------------------------------------
```

Question 2: How does ls work?

A initial guess can be (written in pseudocode):

```
Open directory
while the end of directory is not reached
    display file information
close directory
```

But how to open and read a directory? Any difference from reading a file?

Facts A directory is a kind of file that contains a list of names of files and directories. It consists of a sequence of records, each record represents one item, a single file or a single directory.

Every diretory will contain at least the following two items

- . the current directory
- .. the directory one level up

How to read directory files?

Search in the manual to get some information relevant to directories.

```
$ man -k direct | grep read
pax             (1)  - read and write file archives and copy director
readdir         (2)  - read directory entry
readdir         (3)  - read a directory
seekdir         (3)  - set the position of the next readdir() call in
```

man 3 readdir

```
READDIR(3)                 Linux Programmers Manual                 READDIR(3)

NAME
       readdir - read a directory

SYNOPSIS
       #include <sys/types.h>

       #include <dirent.h>

       struct dirent *readdir(DIR *dir);

DESCRIPTION
       The  readdir()  function  returns  a  pointer to a dirent structure
       representing the next directory entry in the directory stream pointed to
       by dir.  It returns NULL on reaching the end-of-file or if an error
       occurred.

       According  to  POSIX, the dirent structure contains a field char d_name[]
       of unspecified size, with at most NAME_MAX characters preceding the
       terminating null character.  Use  of  other  fields  will harm  the
       portability of your programs.  POSIX 1003.1-2001 also documents the field
       ino_t d_ino as an XSI extension.
```

```
       The data returned by readdir() may be overwritten by subsequent calls to
       readdir()  for  the  same directory stream.

RETURN VALUE
       The  readdir() function returns a pointer to a dirent structure, or NULL
       if an error occurs or end-of-file is reached.

ERRORS
       EBADF  Invalid directory stream descriptor dir.

CONFORMING TO
       SVID 3, BSD 4.3, POSIX 1003.1-2001

SEE ALSO
       read(2), closedir(3), dirfd(3), opendir(3), rewinddir(3), scandir(3),
       seekdir(3), telldir(3)
```

Procedure of reading 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

man dirent.h

```
<dirent.h>(P)                                                    <dirent.h>(P)

NAME
       dirent.h - format of directory entries

SYNOPSIS
       #include <dirent.h>

DESCRIPTION
       The internal format of directories is unspecified.

       The <dirent.h> header shall define the following type:

       DIR    A type representing a directory stream.

       It  shall also define the structure dirent which shall include the fol-
       lowing members:

              ino_t  d_ino       File serial number.

              char   d_name[]    Name of entry.

       The type ino_t shall be defined as described in <sys/types.h> .
```

The character array d_name is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed {NAME_MAX}.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int             closedir(DIR *);
DIR            *opendir(const char *);
struct dirent  *readdir(DIR *);

int             readdir_r(DIR *restrict, struct dirent *restrict,
                    struct dirent **restrict);

void            rewinddir(DIR *);

void            seekdir(DIR *, long);
long            telldir(DIR *);
```

dirent structure defined in /usr/include/dirent.h

```
struct dirent
  {
#ifndef __USE_FILE_OFFSET64
    __ino_t d_ino;
    __off_t d_off;
#else
    __ino64_t d_ino;
    __off64_t d_off;
#endif
    unsigned short int d_reclen;
    unsigned char d_type;
    char d_name[256];
  };
```

Logic for listing a directory

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

First version of our ls–ls1

```
/** 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 []);

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

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);
        }
}
```

Compile and run our first version of ls–ls1

```
$ cc -o ls1 ls1.c
$ ls1
Makefile
ls1.c
.
ls2.c
fileinfo.c
filesize.c
a.out
..
$ ls
a.out  fileinfo.c  filesize.c  ls1.c  ls2.c  Makefile
```

Comparison between our ls1 and ls

- Not sorted
  we can read all the file names into an array and then use qsort to sort the array
- No column
  read the names into an array and print them out in rows and columns.
- Lists . files
  To fix it, just suppress . and .. names
- No -l info
  To be discussed.

Information shown by ls -l

Each line consists of the following seven fields:

- **mode**: The first character in each line represents the types of the file. The remaining nine characters represent the access permissions (read write execute for user, group, everyone else).
- **owner** Each file belongs to a user. The username of the owner is shown.
- **group**: Each file belongs to a group of users too.
- **size**: The number of bytes in the file
- **last-modified** This field consists of three strings representing the last-modified time. For newer files, the output includes the month, day, and time For older files, **ls** shows the month, day, and year.
- **name**: The name of the file

How does ls -l work

```
$ man -k file | grep -i status
fileno [ferror]    (3)- check and reset stream status
fstat [stat]       (2)- get file status
ifcfg-ppp0 [pppoe] (5)- Configuration file used by adsl-start(8),
                         adsl-stop(8), adsl-status(8) and
                         adsl-connect(8)
lam_rfstate        (2)- Report status of remote LAM file
                         descriptors
lstat [stat]       (2)- get file status
stat               (1)- display file or filesystem status
stat               (2)- get file status
```

How does ls -l work

```
NAME
       stat, fstat, lstat - get file status

SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <unistd.h>

       int stat(const char *file_name, struct stat *buf);
       int fstat(int filedes, struct stat *buf);
       int lstat(const char *file_name, struct stat *buf);

DESCRIPTION
       These  functions  return  information about the specified file.  You do
       not need any access rights to the file to get this information but  you
       need  search rights to all directories named in the path leading to the
       file.

       stat stats the file pointed to by file_name and fills in buf.

       lstat is identical to stat, except in the  case  of  a  symbolic  link,
       where the link itself is stat-ed, not the file that it refers to.
```

fstat is identical to stat, only the open file pointed to by filedes (as returned by open(2)) is stat-ed in place of file_name.

They all return a stat structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;     /* device */
    ino_t     st_ino;     /* inode */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device type (if inode device) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;  /* number of blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

The value st_size gives the size of the file (if it is a regular file or a symlink) in bytes. The size of a symlink is the length of the pathname it contains, without trailing NUL.

The value st_blocks gives the size of the file in 512-byte blocks. (This may be smaller than st_size/512 e.g. when the file has holes.) The value st_blksize gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux filesystems implement all of the time fields. Some file system types allow mounting in such a way that file accesses do not cause an update of the st_atime field. (See noatime in mount(8).)

The field st_atime is changed by file accesses, e.g. by execve(2), mknod(2), pipe(2), utime(2) and read(2) (of more than zero bytes). Other routines, like mmap(2), may or may not update st_atime.

The field st_mtime is changed by file modifications, e.g. by mknod(2), truncate(2), utime(2) and write(2) (of more than zero bytes). Moreover, st_mtime of a directory is changed by the creation or deletion of files in that directory. The st_mtime field is not changed for changes in owner, group, hard link count, or mode.

The field st_ctime is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).
...

How does ls -l work

```
/* filesize.c - prints size of passwd file */

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

int main()
{
        struct stat infobuf;              /* place to store info */

        if ( stat( "/etc/passwd", &infobuf) == -1 )  /* get info */
                perror("/etc/passwd");
        else
                printf(" The size of /etc/passwd is %d\n", infobuf.st_size );
}
```

How does ls -l work

```
/* 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 1;
}
```

```
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: %d\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: %d\n", buf->st_size);          /* file size   */
    printf("modtime: %d\n", buf->st_mtime);         /* modified    */
    printf("   name: %s\n", fname );                /* filename    */
}
```

How does ls -l work

```
$ ./fileinfo fileinfo.c
   mode: 100755
  links: 1
   user: 500
  group: 500
   size: 1152
modtime: 1124320515
   name: fileinfo.c
$ ls -l fileinfo.c
-rwxr-xr-x  1 wch wch 1152 Aug 17  2005 fileinfo.c
```

There are several differences between the output of our fileinfo and ls -l: mode, user, group, modtime. For modtime, we can use ctime to convert it to string.

Converting file mode to a string

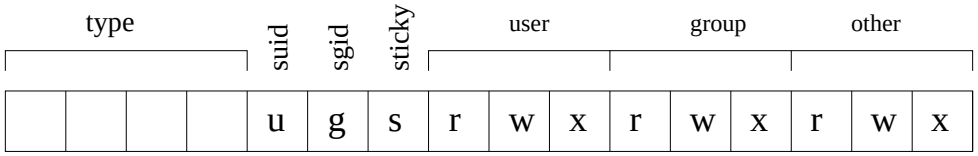

Figure 1: File type and access coding

- The first four bits represent the type of the file. Four bits can contain 16 possible patterns of 1s and 0s. Each pattern can represent one file type.
- The next three bits are for special attributes of a file. Each bit corresponds to a special attribute.

Converting file mode to a string

We use mask to zero out all the bits except the bits we are interested in.

For example 011001010101 && 11100000000 = 01100000000

Here 11100000000 is the marsk.

Using masking to decode file types

The definition of bits pattern for file types are in /usr/include/bits/stat.h

```
/* Encoding of the file mode.  */

#define __S_IFMT 0170000 /* These bits determine file type.  */

/* File types.  */
#define __S_IFDIR 0040000 /* Directory.  */
#define __S_IFCHR 0020000 /* Character device.  */
#define __S_IFBLK 0060000 /* Block device.  */
#define __S_IFREG 0100000 /* Regular file.  */
#define __S_IFIFO 0010000 /* FIFO.  */
#define __S_IFLNK 0120000 /* Symbolic link.  */
#define __S_IFSOCK 0140000 /* Socket.  */
```

Using masking to decode file types

The micros to check the file types is in /usr/include/sys/stat.h

```
/* Test macros for file types. */

#define __S_ISTYPE(mode, mask) (((mode) & __S_IFMT) == (mask))

#define S_ISDIR(mode)  __S_ISTYPE((mode), __S_IFDIR)
#define S_ISCHR(mode)  __S_ISTYPE((mode), __S_IFCHR)
#define S_ISBLK(mode)  __S_ISTYPE((mode), __S_IFBLK)
#define S_ISREG(mode)  __S_ISTYPE((mode), __S_IFREG)
#ifdef __S_IFIFO
# define S_ISFIFO(mode)  __S_ISTYPE((mode), __S_IFIFO)
#endif
#ifdef __S_IFLNK
# define S_ISLNK(mode)  __S_ISTYPE((mode), __S_IFLNK)
#endif
```

Using masking to decode file types

The bit pattern for user access permission is defined in /usr/include/bits/stat.h

```
/* Protection bits.  */
#define __S_ISUID 04000 /* Set user ID on execution.  */
#define __S_ISGID 02000 /* Set group ID on execution.  */
#define __S_ISVTX 01000 /* Save swapped text after use (sticky).  */
#define __S_IREAD 0400 /* Read by owner.  */
#define __S_IWRITE 0200 /* Write by owner.  */
#define __S_IEXEC 0100 /* Execute by owner.  */
```

The bit pattern for group and other permission is defined in /usr/include/sys/stat.h, which includes /usr/include/bits/stat.h

```
#define S_IRGRP (S_IRUSR >> 3) /* Read by group.  */
#define S_IWGRP (S_IWUSR >> 3) /* Write by group.  */
#define S_IXGRP (S_IXUSR >> 3) /* Execute by group.  */
/* Read, write, and execute by group.  */
#define S_IRWXG (S_IRWXU >> 3)
#define S_IROTH (S_IRGRP >> 3) /* Read by others.  */
#define S_IWOTH (S_IWGRP >> 3) /* Write by others.  */
#define S_IXOTH (S_IXGRP >> 3) /* Execute by others.  */
/* Read, write, and execute by others.  */
#define S_IRWXO (S_IRWXG >> 3)
```

Using masking to decode file types

```
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 ID to Strings

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
wch:x:500:100:Changhua Wu:/home/wch:/bin/bash
tomcat:x:91:91:Tomcat:/usr/share/tomcat5:/bin/sh
```

The first field is the user name, the second field is the encripted password, the third field is the user id,the fourth field is the group id. The other fields show the real name of the user, the home directory, and the default shell.

Converting User ID to Strings

However, on some Unix systems, /etc/passwd does not contain all the users. To ensure portability, we use getpwuid.

```
struct passwd *getpwuid(uid_t uid);
```

The getpwuid() function returns a pointer to a structure containing the broken out fields of a line from /etc/passwd for the entry that matches the user uid uid. The passwd structure is defined in pwd.h as follows:

```
struct passwd {
   char    *pw_name;      /* user name */
   char    *pw_passwd;    /* user password */
   uid_t   pw_uid;        /* user id */
   gid_t   pw_gid;        /* group id */
   char    *pw_gecos;     /* real name */
   char    *pw_dir;       /* home directory */
   char    *pw_shell;     /* shell program */
};
```

So we can use this function to get user name from user id

```
char * uid_to_name ( uid_t uid) {
   return getpwuid (uid)->pw_name;
}
```

Converting Group ID to Strings

/etc/group is the list of groups. Like /etc/passwd, it is a plain text file that looks like the following:

```
root:x:0:root
apache:x:48:
iiimd:x:101:
mysql:x:27:
screen:x:84:
users:x:100:wch
wch:x:500:
tomcat:x:91:
```

The name of the group is the first field, the group password is the second filed, the group ID is the third field. The fourth field is a comma-seperated list of usernames.

Converting Group ID to Strings

```
struct group *getgrgid(gid_t gid);
```

The getgrgid() function returns a pointer to a structure containing the group information from /etc/group for the entry that matches the group id.

```
char *gid_to_name ( gid_t uid){
   return getgrgid(gid)->gr_name;
}
```

Our second version of ls–ls2

```
/* 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>

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 );

main(int ac, char *av[])
{
        if ( ac == 1 )
                do_ls( "." );
        else
```

```
                while ( --ac ){
                        printf("%s:\n", *++av );
                        do_ls( *av );
                }
}

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 );
}

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 );

}
/*
 * utility functions
 */

/*
 * 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';
}

#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 ;
}

#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;
}
```

Three special bits

This three bits is defined in /usr/include/bits/stat.h

- Set-User-ID bit: set the effective user ID as the uses ID of the owner

  ```
  #define __S_ISUID 04000 /* Set user ID on execution.*/
  ```

- Set-Group-ID: set the effective group id as the group id of the file

  ```
  #define __S_ISGID 02000 /* Set group ID on execution.*/
  ```

- The Sticky Bit: It has two different uses, one for files and one for directories.

  ```
  #define __S_ISVTX 01000 /* Save swapped text after use (sticky).*/
  ```

A quite detailed description of the sticky bit for files and directories.

`http://www.faqs.org/faqs/hp/hpux-faq/section-70.html`

Setting and modifying the properties of a file

- Type of a file
  The type of file is determined at the time of creation. It is not possible to change the type of a file.
- Permission bits and special bits
  - Establishing the mode of a file
    fd = creat("newfile", 0744); This will create newfile and requests an initial set of permission bits rwxr–r–
  - changing the Mode of a file

```
chmod("/tmp/myfile", 04764);
```

    or

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

man 2 chmod

```
NAME
       chmod, fchmod - change permissions of a file

SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>

       int chmod(const char *path, mode_t mode);
       int fchmod(int fildes, mode_t mode);

DESCRIPTION
       The  mode of the file given by path or referenced by fildes is changed.

       Modes are specified by oring the following:

              S_ISUID   04000 set user ID on execution
              S_ISGID   02000 set group ID on execution
              S_ISVTX   01000 sticky bit
              S_IRUSR (S_IREAD) 00400 read by owner
              S_IWUSR (S_IWRITE) 00200 write by owner
              S_IXUSR (S_IEXEC) 00100 execute/search by owner
              S_IRGRP   00040 read by group
              S_IWGRP   00020 write by group
```

```
        S_IXGRP   00010 execute/search by group
        S_IROTH   00004 read by others
        S_IWOTH   00002 write by others
        S_IXOTH   00001 execute/search by others

    The effective UID of the process must be zero or must match  the  owner
    of the file.
```

Changing the owner and group a file

```
chown ("file1", 200, 40 );
```

```
$man 2 chown

NAME
       chown, fchown, lchown - change ownership of a file

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       int chown(const char *path, uid_t owner, gid_t group);
       int fchown(int fd, uid_t owner, gid_t group);
       int lchown(const char *path, uid_t owner, gid_t group);

DESCRIPTION
       The  owner of the file specified by path or by fd is changed.  Only the
       super-user may change the owner of a file.  The owner  of  a  file  may
       change the group of the file to any group of which that owner is a mem-
       ber.  The super-user may change the group arbitrarily.

       If the owner or group is specified as -1, then that ID is not  changed.
```

```
       When  the  owner  or  group of an executable file are changed by a non-
       super-user, the S_ISUID and S_ISGID mode bits are cleared.  POSIX  does
       not  specify  whether this also should happen when root does the chown;
       the Linux behaviour depends on the kernel version.  In case of  a  non-
       group-executable  file  (with  clear S_IXGRP bit) the S_ISGID bit indi-
       cates mandatory locking, and is not cleared by a chown.

RETURN VALUE
       On success, zero is returned.  On error, -1 is returned, and  errno  is
       set appropriately.
```

Changing the time of a file–utime

The utime system call stes the modificaton and access times for a file.

```
UTIME(2)                    Linux Programmers Manual                    UTIME(2)

NAME
       utime, utimes - change access and/or modification times of an inode

SYNOPSIS
       #include <sys/types.h>
       #include <utime.h>

       int utime(const char *filename, const struct utimbuf *buf);

       #include <sys/time.h>

       int utimes(char *filename, struct timeval *tvp);

DESCRIPTION
       utime  changes the access and modification times of the inode specified
       by filename to the actime and modtime fields of buf  respectively.   If
       buf is NULL, then the access and modification times of the file are set
       to the current time.  The utimbuf structure is:
```

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

    In the Linux DLL 4.4.1 libraries, utimes is just a wrapper  for  utime:
    tvp[0].tv_sec  is  actime,  and  tvp[1].tv_sec is modtime.  The timeval
    structure is:

        struct timeval {
                long    tv_sec;         /* seconds */
                long    tv_usec;        /* microseconds */
        };

RETURN VALUE
    On success, zero is returned.  On error, -1 is returned, and  errno  is
    set appropriately.
```

Changing the name of a file–rename

```
RENAME(2)                  Linux Programmers Manual                  RENAME(2)

NAME
       rename - change the name or location of a file

SYNOPSIS
       #include <stdio.h>

       int rename(const char *oldpath, const char *newpath);

DESCRIPTION
       rename renames a file, moving it between directories if required.

       Any  other  hard links to the file (as created using link(2)) are unaf-
       fected.

       If newpath already exists it will be atomically replaced (subject to  a
       few  conditions - see ERRORS below), so that there is no point at which
       another process attempting to access newpath will find it missing.

       If newpath exists but the operation fails for some reason rename  guar-
       antees to leave an instance of newpath in place.
```

```
       However, when overwriting there will probably be a window in which both
       oldpath and newpath refer to the file being renamed.

       If oldpath refers to a symbolic link the link is  renamed;  if  newpath
       refers to a symbolic link the link will be overwritten.

RETURN VALUE
       On  success,  zero is returned.  On error, -1 is returned, and errno is
       set appropriately.
```