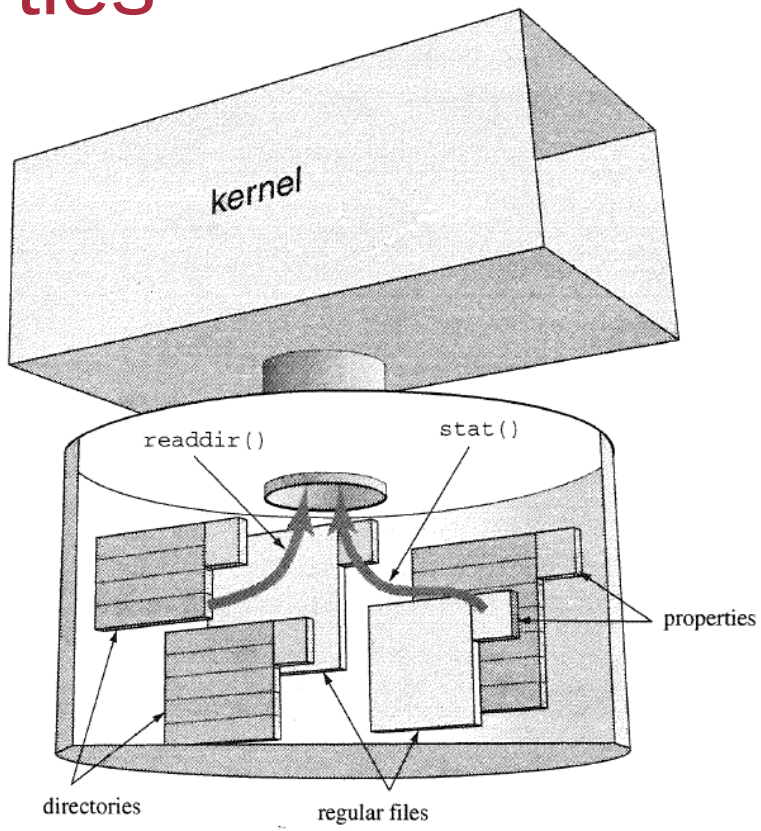
Directories and File Properties

Looking through `ls`

류은경

ekryu@knu.ac.kr



Objectives

- **Ideas and Skills**
  - A directory is a 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`
  - `chmod, chown, utime`
  - `rename`
- **Commands**
  - **`ls`**

3.2 What Does `ls` Do?

`ls` Lists Names of Files and Reports File Attributes

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

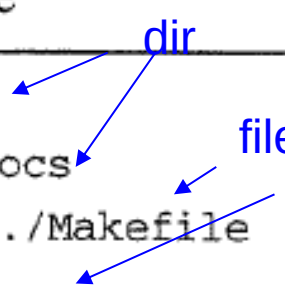
```
$ ls -l
total 108
-rw-rw-r--   2 bruce    users        345 Jul 29 11:05 Makefile
-rw-rw-r--   1 bruce    users      27521 Aug  1 12:14 chap03
drwxrwxr-x   2 bruce    users       1024 Aug  1 12:15 docs
```

permission *owner* *group* *size* *modtime* *name*

Listing Other Directories, Reporting on Other Files

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`



Popular Command-Line Options

- **Other `ls` opations:**

Command	Action
`ls -al`	
`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

옵션	설명	예제
`-a`	숨김 파일(`.`으로 시작하는 파일 포함) 표시	`ls -a`
`-l`	자세한 정보(파일 권한, 크기, 수정 날짜 등) 표시	`ls -l`
`-h`	파일 크기를 사람이 읽기 쉬운 형식(KB, MB 등)으로 표시 (`-l`과 함께 사용)	`ls -lh`
`-t`	수정 시간 기준 정렬(최근 수정된 파일 먼저)	`ls -lt`
`-r`	정렬 순서를 반대로 변경(내림차순)	`ls -lr`
`-R`	하위 디렉터리까지 재귀적으로 출력	`ls -R`
`-d`	디렉터리 자체만 출력 (디렉터리 내부 파일은 출력하지 않음)	`ls -d */`
`-i`	파일의 inode 번호 출력	`ls -i`

What Does `ls` Do

- **`ls` does two things :**
  - Lists the contents of directories
  - Displays information about files

File System Tree

- **The disk is organized as a tree of directories:**
  - Each of which contains files or directories.
  - Even floppy disks, CD-ROMs, and other removable drives appear as subdirectories somewhere on the tree

- **Commands: `cd,pwd,ls`**
  - Allow us to explore a FILE SYSTEM

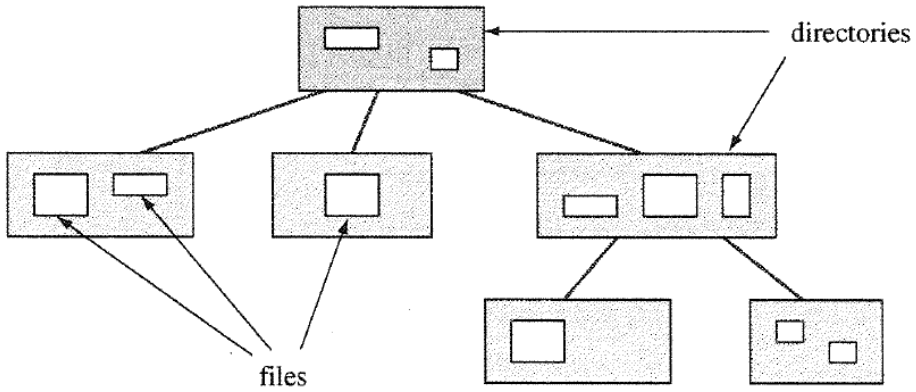
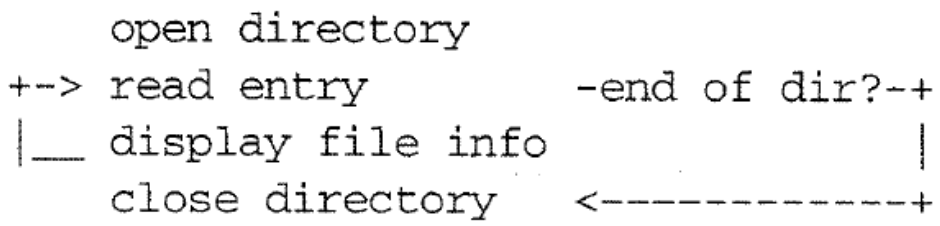


FIGURE 3.1

A tree of directories.

How Dos `ls` Work?

- **Outline :**



What Is a Directory?

- **A directory is a kind of file**
  - It contains a list of names of files and directories.

- Unlike a regular file, **a directory never empty**
  - Every directory contains two specific items: **. and ..**
  - ***dot(.)*** : the name of the current directory
  - ***dotdot(..) :*** the name of the directory one level up

How Do I Read a Directory?

```
$ man -k direct
```

```
$ man -k direct | grep read
DXmHelpSystemDisplay (3X)    - Displays a topic or directory of the
   help file in Bookreader.
opendir, readdir, readdir_r, telldir, seekdir, rewinddir, closedir (3)  -
   Performs operations on directories
```

```
$ man 3 readdir
opendir(3)                                                    opendir(3)

NAME

opendir, readdir, readdir_r, telldir, seekdir, rewinddir, closedir -
Performs operations on directories
```

LIBRARY

Standard C Library (libc.a)

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir (
  const char *dir_name );

struct dirent *readdir (
  DIR *dir_pointer );

int readdir_r (
  DIR *dir_pointer,
  struct dirent *entry,
  struct dirent **result);

long telldir (
  DIR *dir_pointer );

void seekdir (
  DIR *dir_pointer,
  long location );

void rewinddir (
  DIR *dir_pointer );

int closedir (
  DIR *dir_pointer );
```

[more] (11%)

File Formats dirent(4)

NAME

dirent - file system independent directory entry

SYNOPSIS

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

DESCRIPTION

Different file system types may have different directory entries. The dirent structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the getdents(2) system call.

The dirent structure is defined:

```
struct  dirent {
        ino_t           d_ino;
        off_t           d_off;
        unsigned short  d_reclen;
        char            d_name[1];
};
```

In the glibc implementation, the dirent structure is defined as follows:

```
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 */
};
```

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

```
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
```

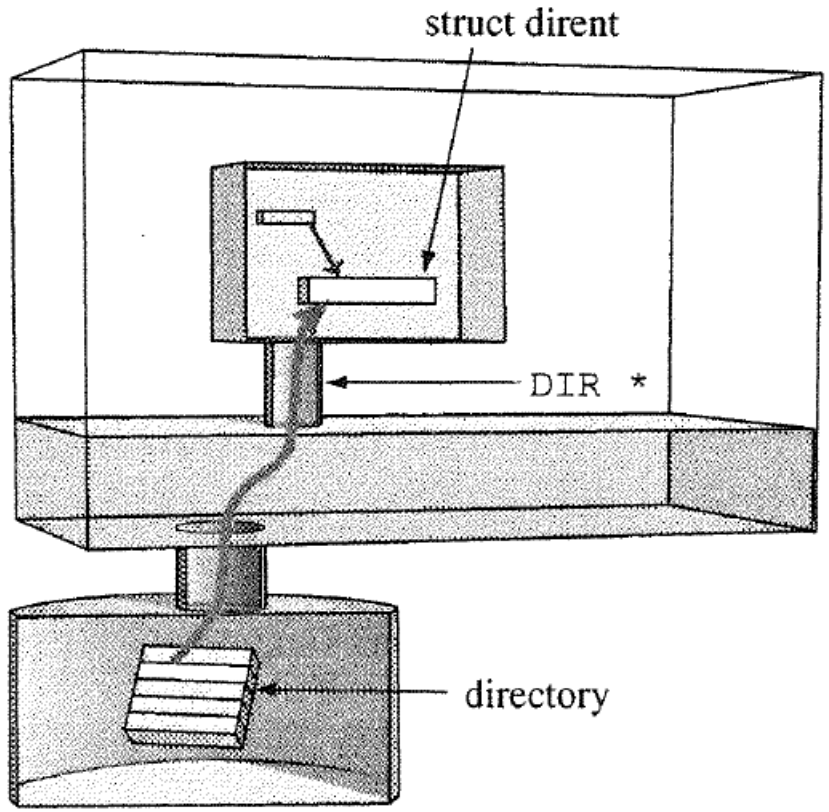


FIGURE 3.2

Reading entries from a directory.

Writing ls1.c

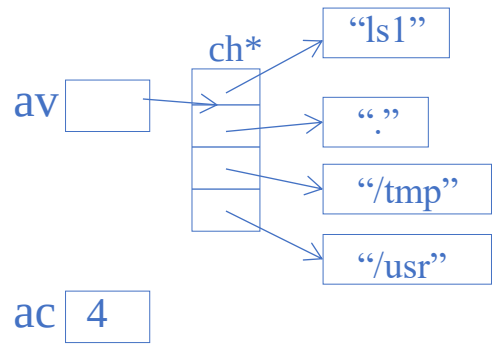
- **Logic for listing a directory:**

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

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

※

```
$ ls1
$ ls1 .
$ ls1 . /tmp /usr
```




```
$ cc -o ls1 ls1.c
$ ls1
.
..
s.tar
tail1
Makefile
ls1.c
ls2.c
chap03
old_src
docs
ls1
stat1.c
statdemo.c
tail1.c
```

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

How Did We Do?

- **`ls1.c` needs work in the following area:**

(a) Not Sorted

- *Fix* → Use `qsort` to sort the array

(b) No Columns

- *Fix* → Read the list of names into an array and then figure out column widths and heights.

(c) List '.' files

- *Fix* → It should be easy to suppress these names and add the –a option.

(d) No –`l` info

- *Fix* → not easy….

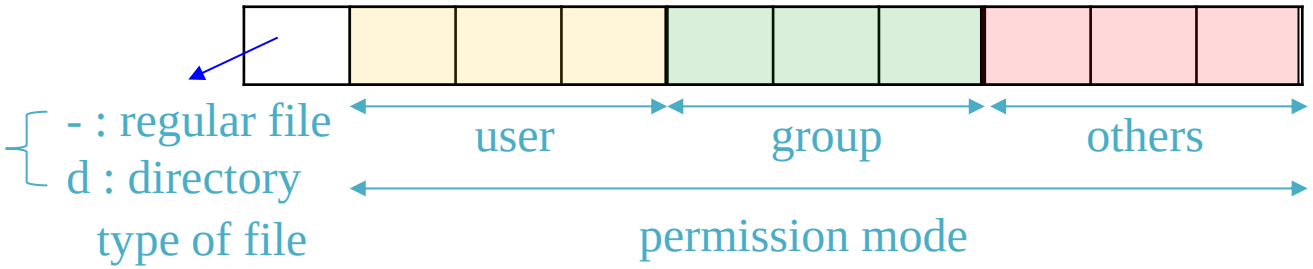
```
$ ls
Makefile    docs        ls1.c       old_src     stat1.c     tail1
chap03      ls1         ls2.c       s.tar       statdemo.c  tail1.c
$
```

```
$ cc -o ls1 ls1.c
$ ls1
.
..
s.tar
tail1
Makefile
ls1.c
ls2.c
chap03
old_src
docs
ls1
stat1.c
statdemo.c
tail1.c
```

Q1: What Does `ls -l` Do?

```
$ 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
```

permssion mode | owner | group | size | last-modified time | name



Q2: How Does `ls -l` work?

- **How can we get information (status/properties) about a file?**
  - The system call that retrieves ***file status*** is called *stat*.
  - Search the manual:

```
$ man -k file | grep -i status
```

STAT(2) Linux Programmer's Manual

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

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

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

The stat structure

All of these system calls return a stat structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    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 ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
       precision for the following timestamp fields.
       For the details before Linux 2.6, see NOTES. */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

The `stat` Call Gets File Information

- **How Does `stat` Work:**

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

`stat(name,ptr)`

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

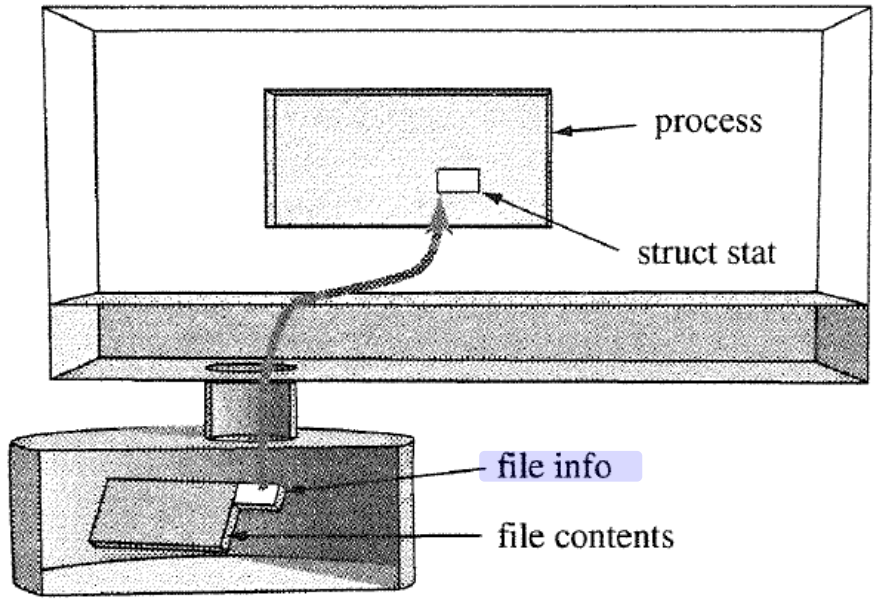


FIGURE 3.3

Reading file properties using `stat`.

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    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 ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */
```

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

What Other Information Does `stat` Provide?

- **Members of `struct stat` :**

```
struct stat {
    dev_t     st_dev;     // 파일이 위치한 장치 ID
    ino_t     st_ino;     // 파일의 inode 번호
    mode_t    st_mode;    // 파일 유형 및 권한 (S_ISREG, S_ISDIR 등)
    nlink_t   st_nlink;   // 하드 링크 개수
    uid_t     st_uid;     // 파일 소유자의 사용자 ID
    gid_t     st_gid;     // 파일 소유자의 그룹 ID
    dev_t     st_rdev;    // 디바이스 ID (장치 파일인 경우)
    off_t     st_size;    // 파일 크기 (바이트 단위)
    blksize_t st_blksize; // I/O 블록 크기
    blkcnt_t  st_blocks;  // 할당된 블록 수
    time_t    st_atime;   // 마지막 접근 시간 (Access)
    time_t    st_mtime;   // 마지막 수정 시간 (Modify)
    time_t    st_ctime;   // 마지막 상태 변경 시간 (Change)
};
```

```
/* fileinfo.c - use stat() to obtain and print file properties
 *            - 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;
}
```

```
$ cc -o fileinfo fileinfo.c
$ ./fileinfo fileinfo.c
   mode: 100664
  links: 1
   user: 500
  group: 120
   size: 1106
modtime: 965158604
   name: fileinfo.c
```

```
$ cc -o fileinfo fileinfo.c
$ ./fileinfo fileinfo.c
   mode: 100664
  links: 1
   user: 500
  group: 120
   size: 1106
modtime: 965158604
   name: fileinfo.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: %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'd We Do?

```
$ cc -o fileinfo fileinfo.c
$ ./fileinfo fileinfo.c
    mode: 100664
   links: 1
    user: 500
   group: 120
    size: 1106
 modtime: 965158604
    name: fileinfo.c
$ ls -l fileinfo.c
-rw-rw-r--    1 bruce    users        1106 Aug  1 15:36 fileinfo.c
```

- Mod time → Use `ctime()` to convert
- Owner, group → …
- Permssion → …

Converting File Mode to a String

- **`st_mode` is a 16-bit value :**

```
$ ./fileinfo fileinfo.c
    mode: 100664
```

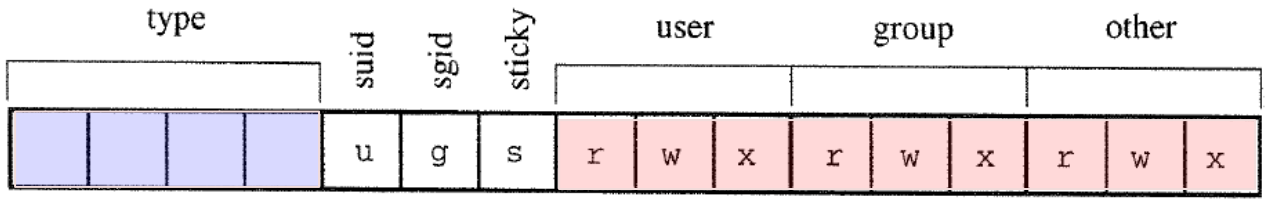


FIGURE 3.4

File type and access coding.

- Type: **file types**
  - 4 bits means 16 possible patterns
  - Each pattern can correspond to a file type
- **Permission** bits : ‘1’ or ‘0’

How to Read Subfields: Masking

- **How do we examine a bit or subfield?**
  - ex) 100664 (base 8) → -rw-rw-r--

- **Use "bitwise AND (&)" to MASK :**
  - 1's in the mask allow a value to 'show through'

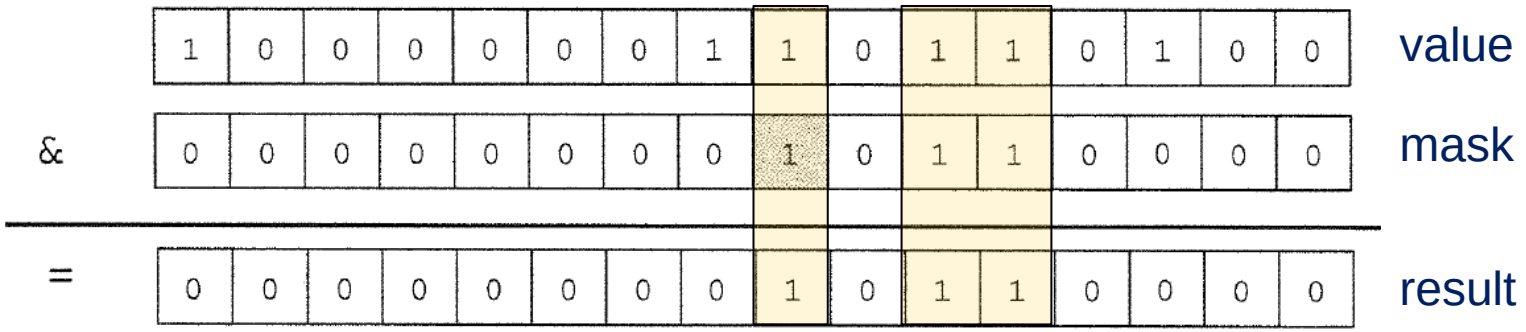
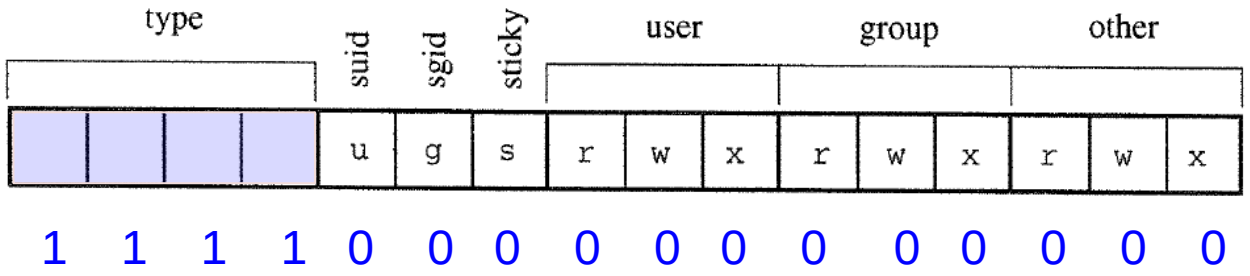


FIGURE 3.6

Applying a bitmask.

Using Masking to Decode: File Types

- **Mask and file types defined in `<sys/stat.h>`**



mask

```
#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 types

※ octal

```
if ( (info.st_mode & 0170000) == 0040000 )
        printf("this is a directory.");
```

```
/*
 *      File type macros defined in <sys/stat.h>`
 */

#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))
```

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

Using Masking to Decode : Permission Bits

- **100664 (base 8) → -rw-rw-r--**

```
$ ./fileinfo fileinfo.c
    mode: 100664
```

```
$ ./fileinfo fileinfo.c
    mode: 100664
```

```
S_IRWXU      00700   owner has read, write, and execute permission
S_IRUSR      00400   owner has read permission
S_IWUSR      00200   owner has write permission
S_IXUSR      00100   owner has execute permission

S_IRWXG      00070   group has read, write, and execute permission
S_IRGRP      00040   group has read permission
S_IWGRP      00020   group has write permission
S_IXGRP      00010   group has execute permission
```

```
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';
}
```

masks defined in <sys/stat.h>

Converting User ID to Strings

```
$ ./fileinfo fileinfo.c
   mode: 100664
  links: 1
   user: 500
  group: 120
   size: 1106
modtime: 965158604
   name: fileinfo.c
$ ls -l fileinfo.c
-rw-rw-r--    1 bruce    users        1106 Aug  1 15:36 fileinfo.c
```

`/etc/passwd` Is the List of Users

```
$ more /etc/passwd
root:WPA4d1OwUxypE:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
smith:x1mEPcp4TNokc:9768:3073:James Q Smith:/home/s/smith:/shells/tcsh
fred:mSuVNOF4CRTmE:20359:550:Fred:/home/f/fred:/shells/tcsh
```

username UID GID information about one user

- `/etc/passwd` Is Not Always the Complete List of Users
- it does NOT work on many networked systems.

- **Library function `getpwuid()`**
  - Provides access to the complete list of users

```
GETPWNAM(3)                                                Linux Programmer's Manual

NAME
       getpwnam, getpwnam_r, getpwuid, getpwuid_r - get password file entry

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

       struct passwd *getpwnam(const char *name);

       struct passwd *getpwuid(uid_t uid);

       int getpwnam_r(const char *name, struct passwd *pwd,
                      char *buf, size_t buflen, struct passwd **result);

       int getpwuid_r(uid_t uid, struct passwd *pwd,
                      char *buf, size_t buflen, struct passwd **result);
```

```
/* The passwd structure.  */
struct passwd
{
    char *pw_name;                /* Username.  */
    char *pw_passwd;              /* 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.  */
};
```

```
/*
 * returns a username associated with the specified uid
 * NOTE: does not work if there is no username
 */
char *uid_to_name( uid_t uid )
{
    return getpwuid(uid)->pw_name ;
}
```

Converting Group ID to Strings

```
$ ./fileinfo fileinfo.c
    mode: 100664
   links: 1
    user: 500
   group: 120
    size: 1106
modtime: 965158604
    name: fileinfo.c
$ ls -l fileinfo.c
-rw-rw-r--    1 bruce    users        1106 Aug  1 15:36 fileinfo.c
```

`/etc/group` Is the List of Groups

- A user can belong to more than one group

```
$ more /etc/group
root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
uucp::5:root,uucp
mail::6:root
tty::7:root,tty,adm
lp::8:root,lp,adm
```

※ `$ man 5 group`
`$ ls -l /etc/group`

```
    group_name:password:GID:user_list

The fields are as follows:

group_name   the name of the group.

password     the (encrypted) group password.  If this field is empty, no password is needed.

GID          the numeric group ID.

user_list    a list of the usernames that are members of this group, separated by commas.
```

`getgrgid()` Provides Access to the List of Groups

- Defined in `/usr/include/grp.h`

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

※ `$ man getgrgid`

```
struct group {
    char   *gr_name;        /* group name */
    char   *gr_passwd;      /* group password */
    gid_t   gr_gid;         /* group ID */
    char  **gr_mem;         /* group members */
};
```

```
/*
 * returns a groupname associated with the specified gid
 * NOTE: does not work if there is no groupname
 */
char *gid_to_name( gid_t gid )
{
    return getgrgid(gid)->gr_name ;
}
```

Putting It All Together: `ls2.c`

```
$ cc -o ls1 ls1.c
$ ls1
.
..
s.tar
tail1
Makefile
ls1.c
ls2.c
chap03
old_src
docs
ls1
stat1.c
statdemo.c
tail1.c
```

```
$ cc -o fileinfo fileinfo.c
$ ./fileinfo fileinfo.c
   mode: 100664
  links: 1
   user: 500
  group: 120
   size: 1106
modtime: 965158604
   name: fileinfo.c
```

```
$ ls2
drwxrwxr-x   4 bruce    bruce        1024 Aug  2 18:18 .
drwxrwxr-x   5 bruce    bruce        1024 Aug  2 18:14 ..
-rw-rw-r--   1 bruce    users       30720 Aug  1 12:05 s.tar
-rwxrwxr-x   1 bruce    users       37351 Aug  1 12:13 tail1
-rw-rw-r--   2 bruce    users         345 Jul 29 11:05 Makefile
-rw-r--r--   1 bruce    users         723 Aug  1 14:26 ls1.c
-rw-r--r--   1 bruce    users        3045 Feb 15 03:51 ls2.c
-rw-rw-r--   1 bruce    users       27521 Aug  1 12:14 chap03
drwxrwxr-x   2 bruce    users        1024 Aug  1 12:14 old_src
drwxrwxr-x   2 bruce    users        1024 Aug  1 12:15 docs
-rwxrwxr-x   1 bruce    bruce       37048 Aug  1 14:26 ls1
```

```
/* 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 );

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 );
}
```

```
$ ./ls2
drwxrwxr-x   4 bruce    bruce        1024 Aug  2 18:18 .
drwxrwxr-x   5 bruce    bruce        1024 Aug  2 18:14 ..
-rw-rw-r--   1 bruce    users       30720 Aug  1 12:05 s.tar
-rwxrwxr-x   1 bruce    users       37351 Aug  1 12:13 tail1
-rw-rw-r--   2 bruce    users         345 Jul 29 11:05 Makefile
-rw-r--r--   1 bruce    users         723 Aug  1 14:26 ls1.c
-rw-r--r--   1 bruce    users        3045 Feb 15 03:51 ls2.c
```

```
/*
 * 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;
}
```

- **Result : *Needs more work!***

```
$ ./ls2
drwxrwxr-x   4 bruce    bruce        1024 Aug  2 18:18 .
drwxrwxr-x   5 bruce    bruce        1024 Aug  2 18:14 ..
-rw-rw-r--   1 bruce    users       30720 Aug  1 12:05 s.tar
-rwxrwxr-x   1 bruce    users       37351 Aug  1 12:13 tail1
-rw-rw-r--   2 bruce    users         345 Jul 29 11:05 Makefile
-rw-r--r--   1 bruce    users         723 Aug  1 14:26 ls1.c
-rw-r--r--   1 bruce    users        3045 Feb 15 03:51 ls2.c
```

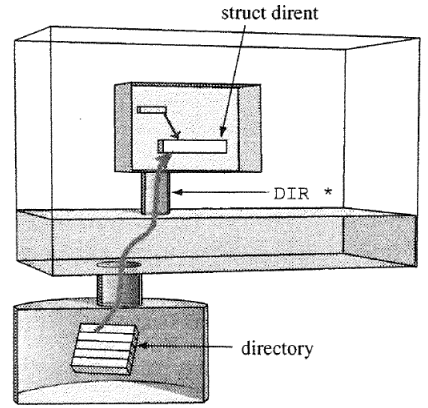
```
$ ls -l
total 189
-rw-rw-r--   2 bruce    users         345 Jul 29 11:05 Makefile
-rw-rw-r--   1 bruce    users       27521 Aug  1 12:14 chap03
drwxrwxr-x   2 bruce    users        1024 Aug  1 12:15 docs
-rwxrwxr-x   1 bruce    bruce       37048 Aug  1 14:26 ls1
-rw-r--r--   1 bruce    users         723 Aug  1 14:26 ls1.c
-rwxrwxr-x   2 bruce    bruce       42295 Aug  2 18:18 ls2
-rw-r--r--   1 bruce    users        3045 Feb 15 03:51 ls2.c
```

sorting

Summary : $ ./ls2 or $ ls -l

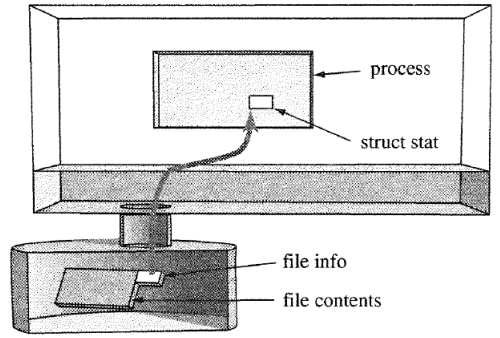
readdir()

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* not an offset; see NOTES */
    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]; /* filename */
};
```



stat()

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    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 ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;  /* number of 512B 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 */
};
```

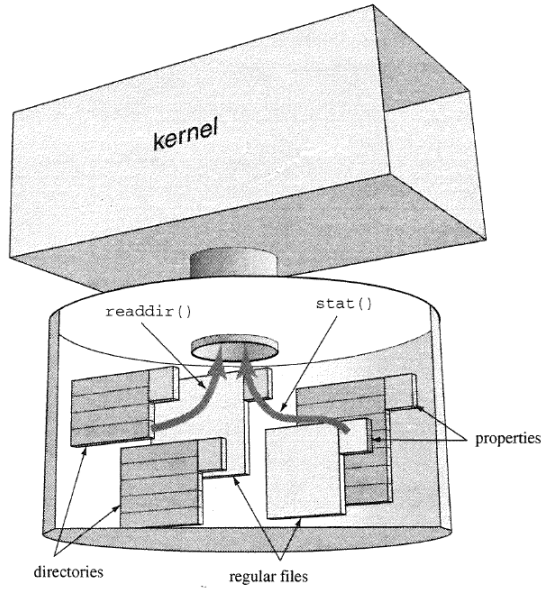


getpwuid()

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

getgrgid()

```
struct group {
    char   *gr_name;       /* group name */
    char   *gr_passwd;     /* group password */
    gid_t   gr_gid;        /* group ID */
    char  **gr_mem;        /* group members */
};
```



Directories and File Properties

3.2 What Does ls Do?

3.3 Brief Review of the File System Tree

3.4 How Dos ls Work?

3.5 Can I Write ls?

3.6 Writing ls -l

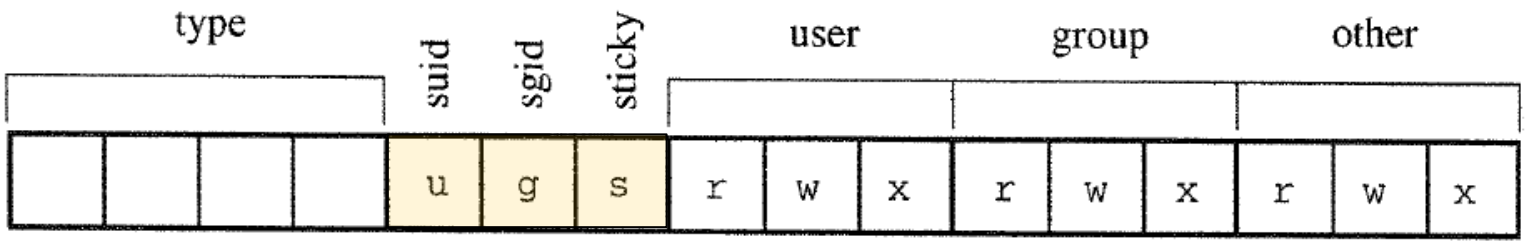
3.7 Three Special Bits

3.8 Summay of ls

3.9 Setting and Modifying the Properties of a File

Three Special Bits

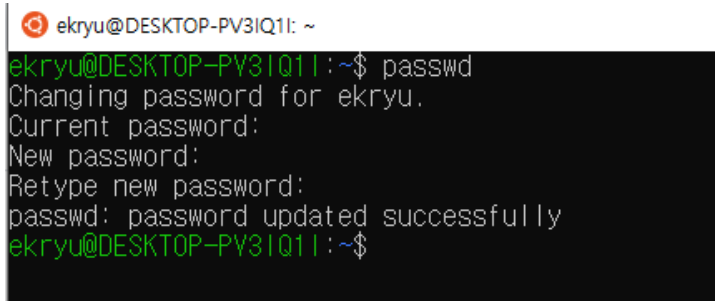
- **The `st_mode` member of the `stat` structure:**



- **Three special bits are used *to activate special properties of a file:***
  - suid(set-user-ID) bit
  - sgid(set-group-ID) bit
  - sticky bit

The Set-User-ID Bit

- How can a regular user **change his/her password?**
  - Use the **`passwd`** command!



- **Problem:**
  - Changing your record in the file `/etc/passwd`
    → you do **NOT** have **permission** to write to that file
  - Only the user named **`root`** has write permission.

```
$ ls -l /etc/passwd
-rw-r--r--   1 root     root          894 Jun 20 19:17 /etc/passwd
```

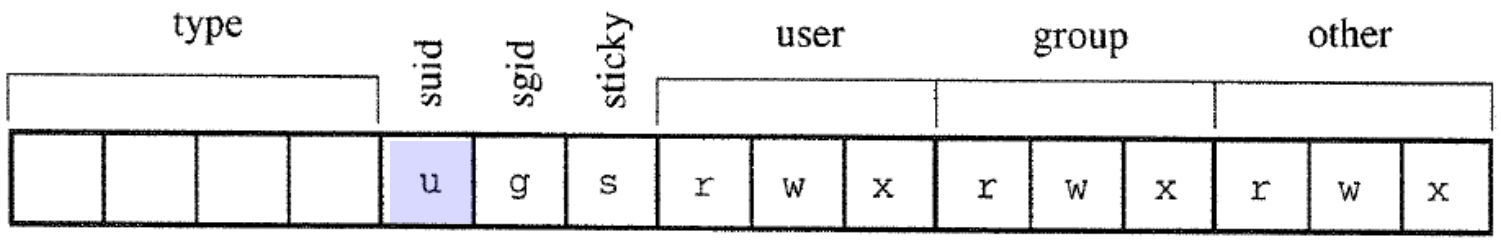
The Set-User-ID Bit

Solution: Give Permission to the Program, not to you.

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x   1 root     bin          15725 Oct 31  1997 /usr/bin/passwd
```

suid

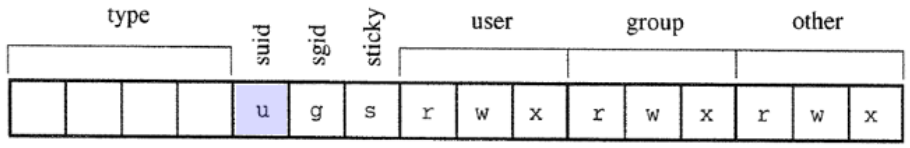
- That **SUID bit** tells the kernel to run the program as though it were being run by the **owner** of the program



The Set-User-ID Bit

- **A mask to test for the `SUID` bit**
  - defined via `<sys/stat.h>`

```
#define S_ISUID         0004000         /* set user id on execution */
```




```c
if (stat(argv[1], &fileStat) < 0) {
    perror("stat");
    return 1;
}

if (fileStat.st_mode & S_ISUID) {
    printf("SUID bit is set.\n");
} else {
    printf("SUID bit is not set.\n");
}
```

The Set-Group-ID Bit

- **The SGID bit sets the *effective group ID* of a program**
  - This bit grants the program **the access rights of members** of that group

- A mask to test for the **`SGID`** bit:

```
#define S_ISGID         0002000         /* set group id on execution */
```

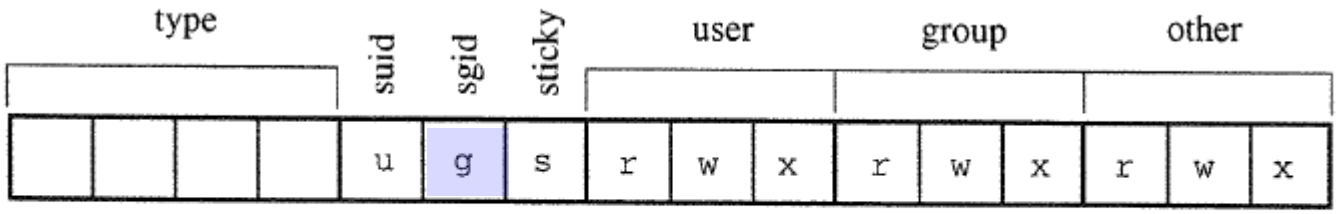


FIGURE 3.4

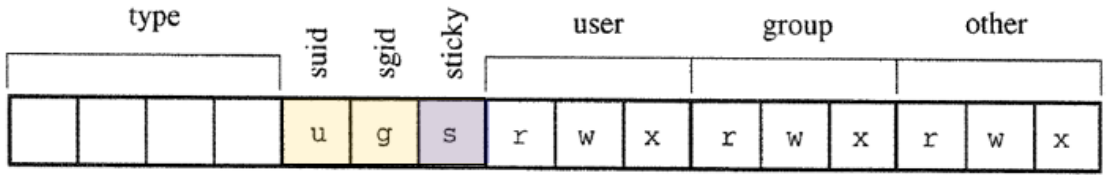
File type and access coding.

The Sticky Bit

- **Use for directories**
  - `/tmp` are publicly writable, allowing ***any user*** *to create and delete any files there*

```
dr-xr-xr-x  13 root root     0  9월 12 13:25 sys
drwxrwxrwt  25 root root  4096  9월 19 13:54 tmp
drwxr-xr-x  14 root root  4096  8월 19  2021 usr
```

- **When a directory's sticky bit is set,**
  - Files in the directory may **ONLY** be deleted **by** their **owners**



Sticky Bit
- Linux 및 Unix 시스템에서 디렉터리에 적용되는 특별한 권한
- Sticky Bit 설정 시, 파일 소유자만 삭제 및 수정 가능
- 디렉터리에 쓰기 권한이 있어도 다른 사용자는 삭제 불가

The Special Bits and `ls -l`

- Each file has *a type* and *12 attribute bits*
  - but `ls` uses only 9 spots to display these 12 attributes

```
-rwsr-sr-t   1 root     root         2345 Jun 12 14:02 sample
```

  - Letter s → augmented by the **set-user** and **set-group ID** bits
  - Letter t → the **sticky bit** is on

```
$ echo "Hello, World! "  > sample
$ sudo chmod 7755 sample   # SUID + SGID + Sticky Bit 설정
```

```
-rw-r--r-- 1 2007200_40634 domain users    14  9월 19 14:42 sample
2007200_40634@Ubuntu-039:~/다운로드$ chmod 7755 sample
2007200_40634@Ubuntu-039:~/다운로드$ ls -l sample
-rwsr-sr-t 1 2007200_40634 domain users 14  9월 19 14:42 sample
```

Type of a File

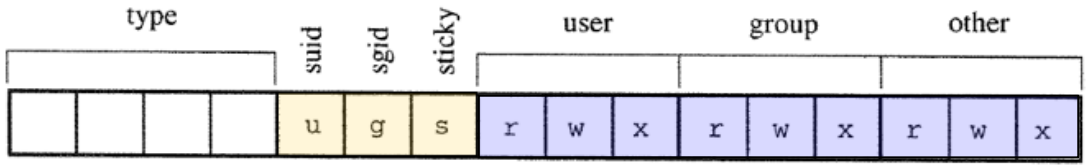
```
root@DESKTOP-K4MA2V5:~# ls -l /etc/passwd
-rw-r--r-- 1 root root 1615  2월 27 16:59 /etc/passwd
root@DESKTOP-K4MA2V5:~#
```

$ls –l /dev | more

- **A file has a type**
  - Regular file(-), Directory(d), Device file(b, c), Socket(s), Symbolic link(l), Named pipe(p)
- The type of the file is established ***when the file is created***
  - The **`creat()`** system call creates a *regular file*.
  - Different system call are used to create *directories*, *devices*, and the like.
- It is **not possible to change** the type of a file

Permission Bits and Special Bits

- Every file has 9 permission bits and 3 special bits.



- These bits are **established when file is created** and can be **modified by** making the **`chmod`** system call:

```
fd = creat( "newfile", 0744 );
$ chmod 744 newfile
```

- If you want **to prevent** programs from creating files that can be **modified by group or others :**

```
umask(022);
```

- Changing the **Mode of a File:** **`Chmod()`** system call

```
$ man -k chmod
$ man 2 chmod
```

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

or

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

- A **Shell Command** to Change Permission and Special Bits :

```
$ chmod 04764 test
        or
$ chmod u=rws test
$ chmod g=rw test
$ chmod o=r test
```

Owner and Group of a File

- System call : **`chown()`**
  - Normally, users do not change the owner of a file
  - Typically used to set up and manage **user accounts**

- **Shell Commands** : **`chown, chgrp`**

```
# myfile의 소유자(owner)를 newuser로 변경
sudo chown newuser myfile

# myfile의 그룹(group)을 newgroup으로 변경
sudo chgrp newgroup myfile

# mydir 디렉토리의 소유자(owner)를 newuser로, 그룹(group)을 newgroup으로 변경
sudo chown newuser:newgroup mydir

# mydir 디렉토리와 하위 모든 파일 및 디렉토리의 소유자와 그룹을 변경 (재귀적 적용)
sudo chown -R newuser:newgroup /path/to/mydir

# mydir 디렉토리와 하위 모든 파일 및 디렉토리의 그룹을 newgroup으로 변경 (재귀적 적용)
sudo chgrp -R newgroup /path/to/mydir
```

Modification and Access Time

- Each file has **three timestamps of**
  - *last modified, last read*
  - *file properties were last changed*
    - owner ID or permission bits
- **Kernel** automatically updates these times as programs read and write the file

- **Changing Modification and Access Times:**
  `utime(), touch`

```
$ touch -t 202309201230 file.txt
```

```
# 1. 파일 생성
touch newfile.txt

# 2. 타임스탬프 갱신
touch existingfile.txt

# 3. 특정 시간으로 타임스탬프 변경 (2023년 9월 29일 12:00)
touch -t 202309291200 file.txt

# 4. 접근 시간만 변경
touch -a file.txt

# 5. 수정 시간만 변경
touch -m file.txt

# 6. 다른 파일의 타임스탬프 복사
touch -r sourcefile.txt targetfile.txt
```

Name of a File

- **Establishing** the Name of a File
  - `creat()` system call sets the name and the initial mode of a file.

- **Changing** the Name of a File: `rename()`, `mv`

Summary

- **Ideas and Skills**
  - A directory is a 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`
  - `chmod, chown, utime`
  - `rename`
- **Commands**
  - **`ls`**

VISUAL SUMMARY

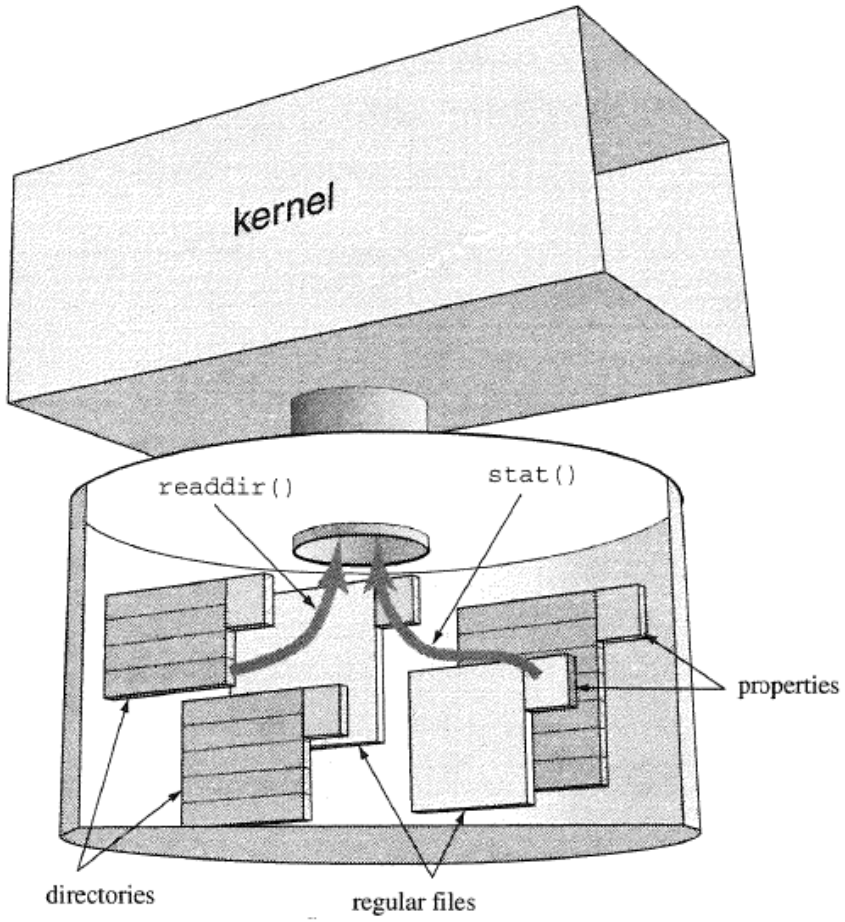


FIGURE 3.7

A disk contains files, directories, and their properties.