CH 3: Directories and File Properties

Ideas and Skills

- A directory is a list of files
- How to read a directory
- Types of files and how to determine their type
- 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

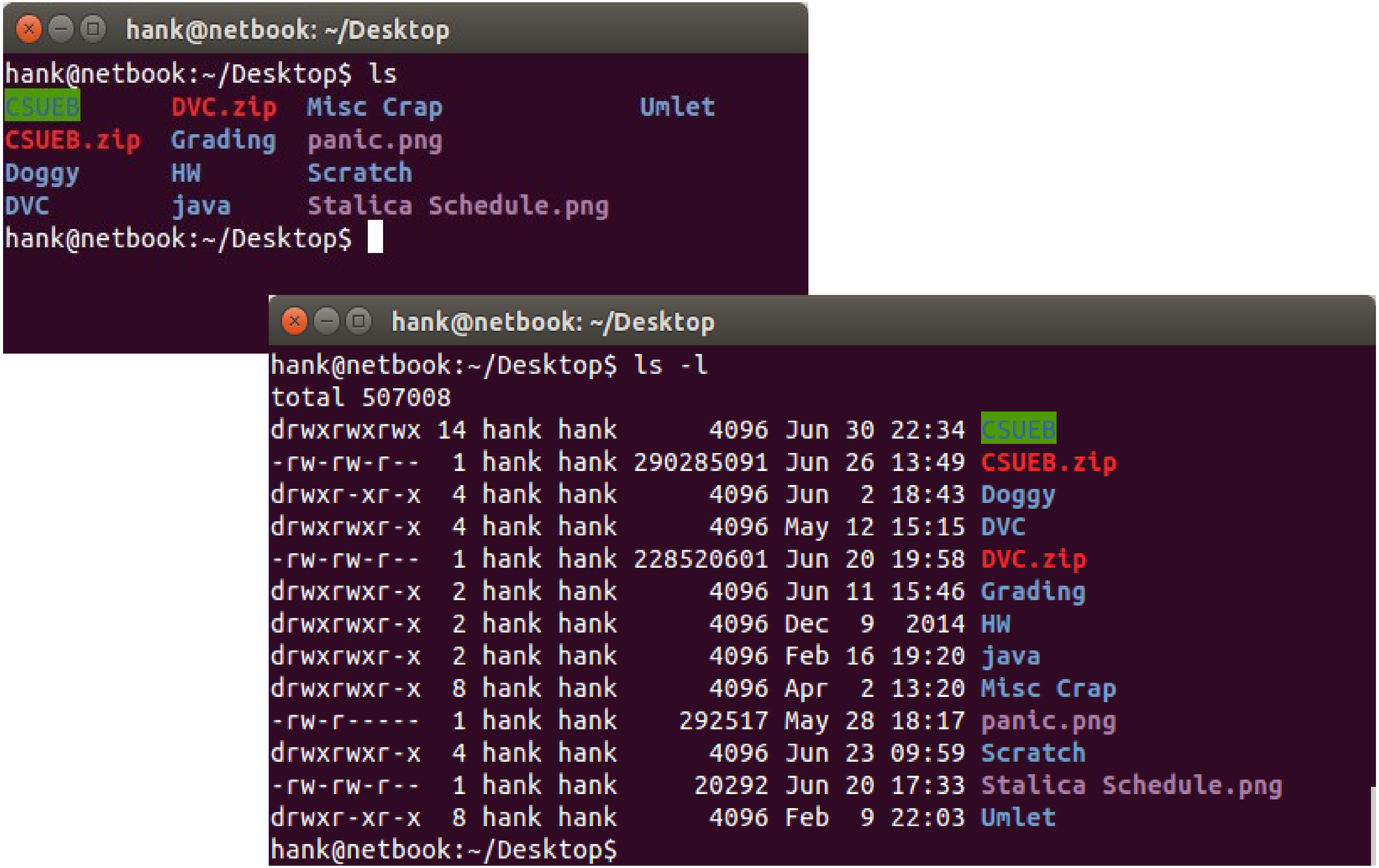
Commands

- ls

Introduction

- Files have more than content: owner, size, type, etc. How can we access this info?
- The ls command lists this information.
- Let's use the three question approach to study ls and learn more about files.

Q1: What does ls Do?



```
hank@netbook:~/Desktop$ ls
CSUEB      DVC.zip  Misc Crap               Umlet
CSUEB.zip  Grading  panic.png
Doggy      HW       Scratch
DVC        java     Stalica Schedule.png
hank@netbook:~/Desktop$ 
```

```
hank@netbook:~/Desktop$ ls -l
total 507008
drwxrwxrwx 14 hank hank      4096 Jun 30 22:34 CSUEB
-rw-rw-r--  1 hank hank 290285091 Jun 26 13:49 CSUEB.zip
drwxr-xr-x  4 hank hank      4096 Jun  2 18:43 Doggy
drwxrwxr-x  4 hank hank      4096 May 12 15:15 DVC
-rw-rw-r--  1 hank hank 228520601 Jun 20 19:58 DVC.zip
drwxrwxr-x  2 hank hank      4096 Jun 11 15:46 Grading
drwxrwxr-x  2 hank hank      4096 Dec  9  2014 HW
drwxrwxr-x  2 hank hank      4096 Feb 16 19:20 java
drwxrwxr-x  8 hank hank      4096 Apr  2 13:20 Misc Crap
-rw-r-----  1 hank hank    292517 May 28 18:17 panic.png
drwxrwxr-x  4 hank hank      4096 Jun 23 09:59 Scratch
-rw-rw-r--  1 hank hank     20292 Jun 20 17:33 Stalica Schedule.png
drwxr-xr-x  8 hank hank      4096 Feb  9 22:03 Umlet
hank@netbook:~/Desktop$
```

Listing Other Directories, Reporting on Other Files

- We can get information about files and directories in other directories.

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

Popular command-line options

- Popular command-line options:

Example	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

Answer 1: A Summary

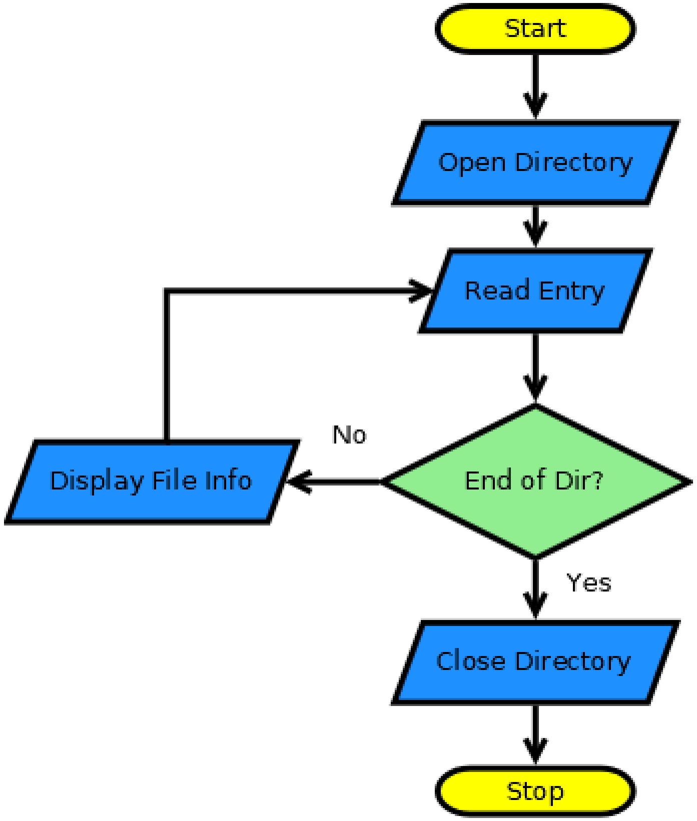
- Playing with ls and reading man pages, we find ls:

  Lists the contents of directories
  Displays the information about files

Q2: How does ls work?

Seems to work like this:

Sort of like who, but who reads from a file, ls reads from a directory

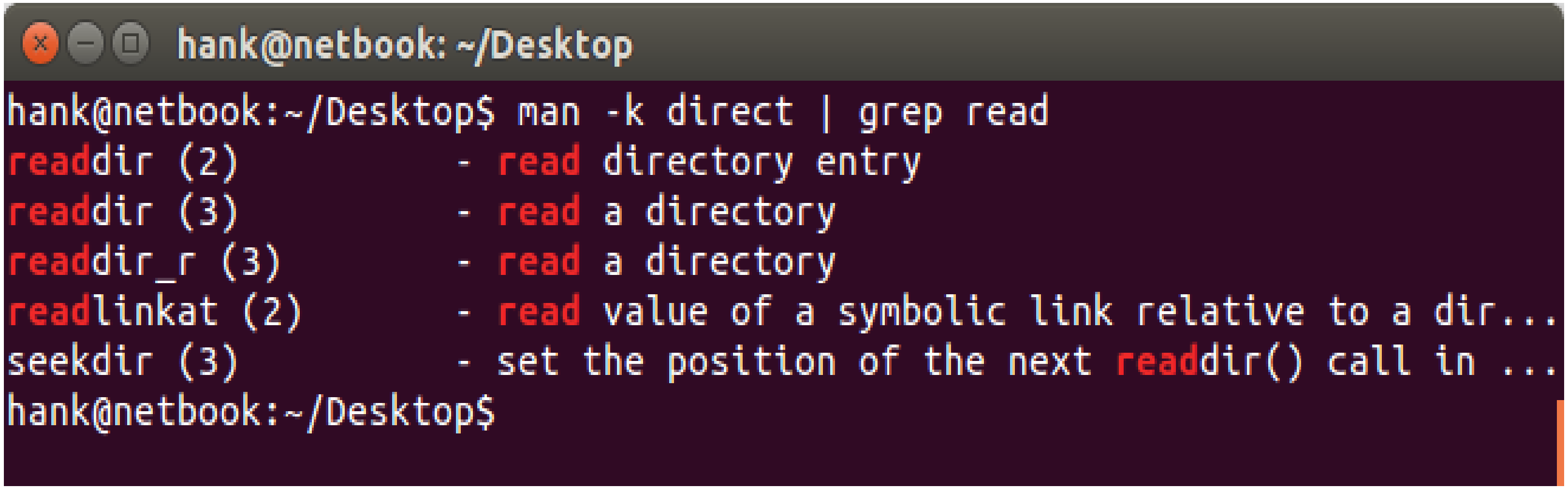


What's a directory, anyway?

- Internally, a directory stores a sequence of records.
- Each record represents one item: a file or a directory.
- Directories always contains at least . and ..
  . is the current directory, .. is the parent

How do I read a directory?

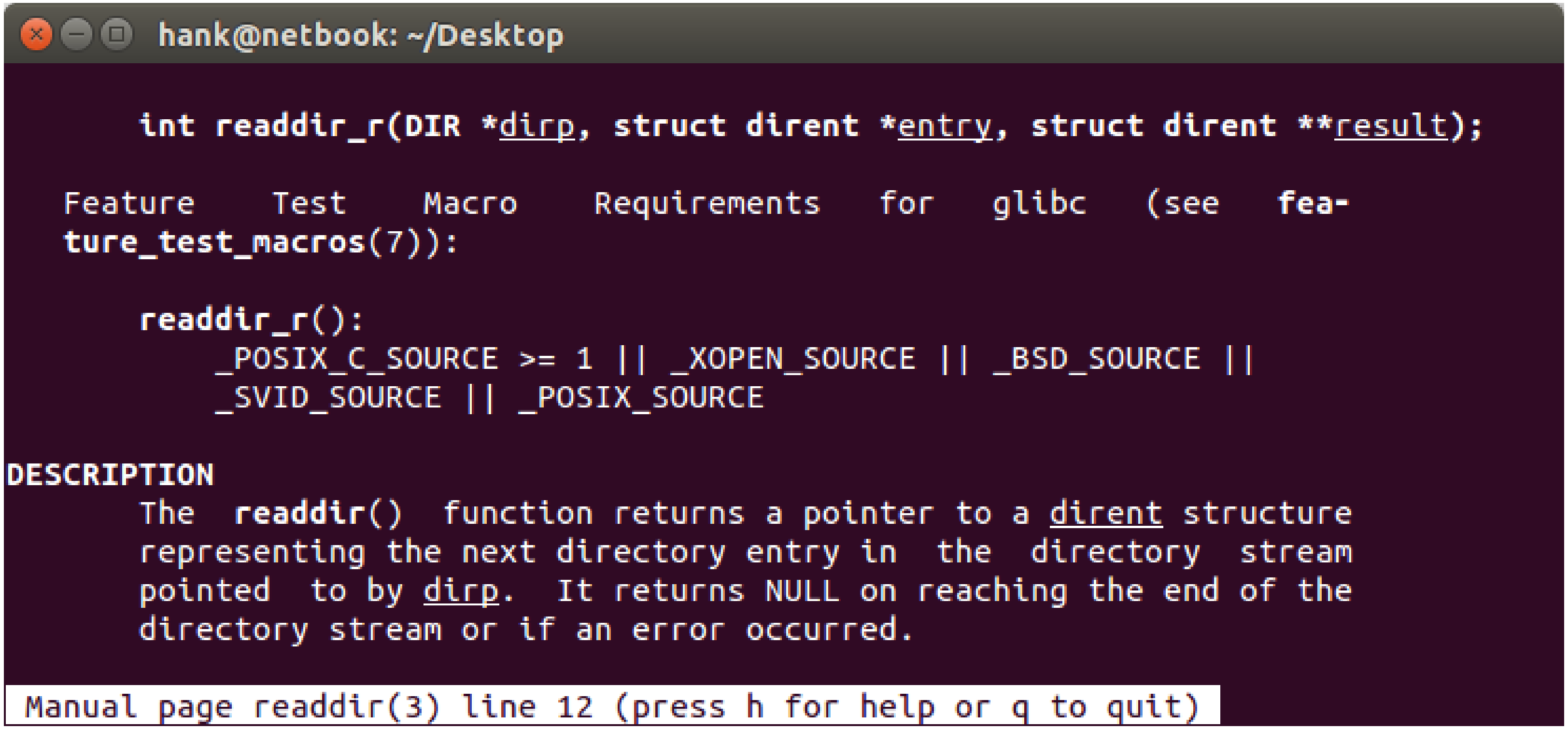
- To the man pages!
  man -k direct finds a lot of entries.
- Narrow it down:
  man -k direct | grep read



```
hank@netbook:~/Desktop$ man -k direct | grep read
readdir (2)          - read directory entry
readdir (3)          - read a directory
readdir_r (3)        - read a directory
readlinkat (2)       - read value of a symbolic link relative to a dir...
seekdir (3)          - set the position of the next readdir() call in ...
hank@netbook:~/Desktop$
```

How do I read a directory?

- Looks good, so man 3 readdir shows us



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

   Feature    Test    Macro    Requirements    for    glibc    (see    fea‐
   ture_test_macros(7)):

       readdir_r():
           _POSIX_C_SOURCE >= 1 || _XOPEN_SOURCE || _BSD_SOURCE ||
           _SVID_SOURCE || _POSIX_SOURCE

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

 Manual page readdir(3) line 12 (press h for help or q to quit)
```

How do I read a directory?

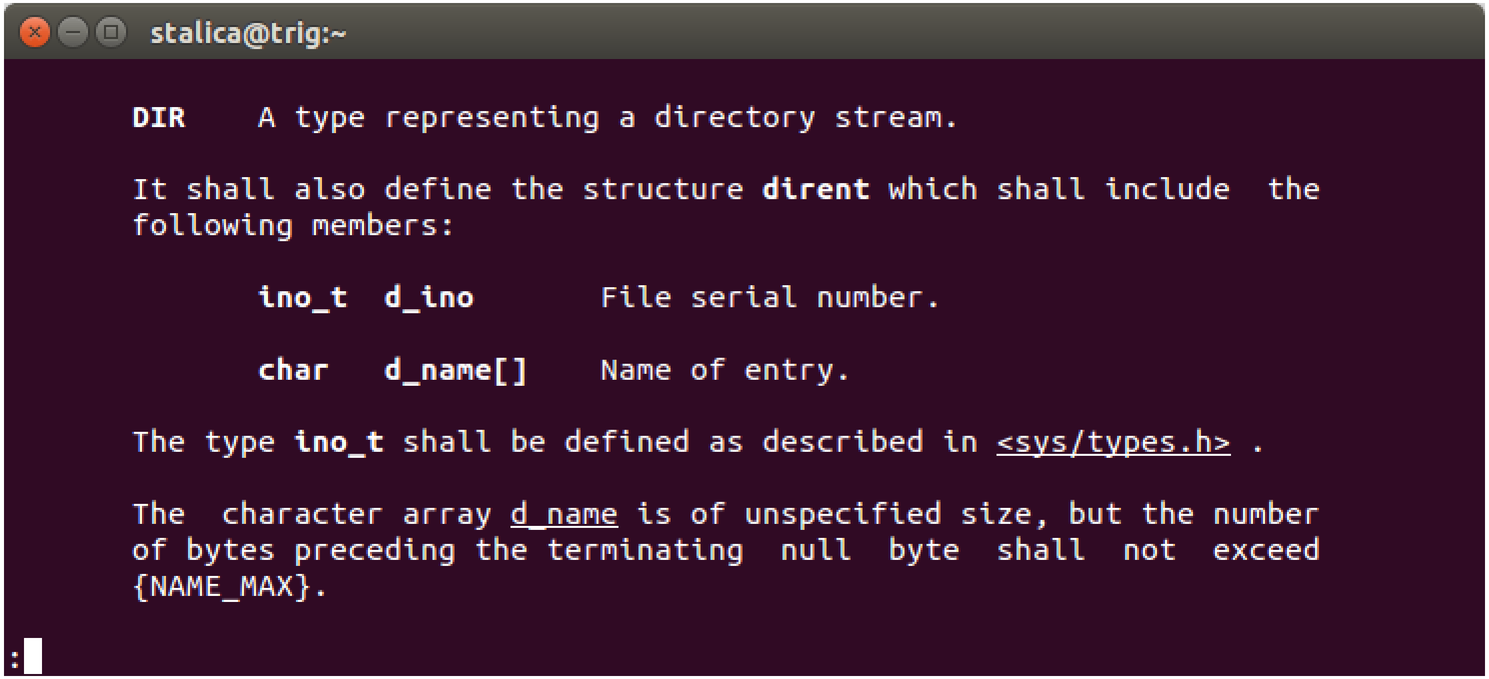
- Similar to files.
- opendir opens a connection to a directory, readdir returns a pointer to the next directory record, and closedir closes the connection.
- seekdir, telldir, and rewinddir are similar to lseek.

How do I read a directory?

- opendir(char*)
  creates a connection, returns DIR*
- readdir(DIR*)
  reads next record, returns struct dirent*
- closedir(DIR*)
  closes a connection

Reading the directory contents

struct dirent is described in the man pages and in dirent.h

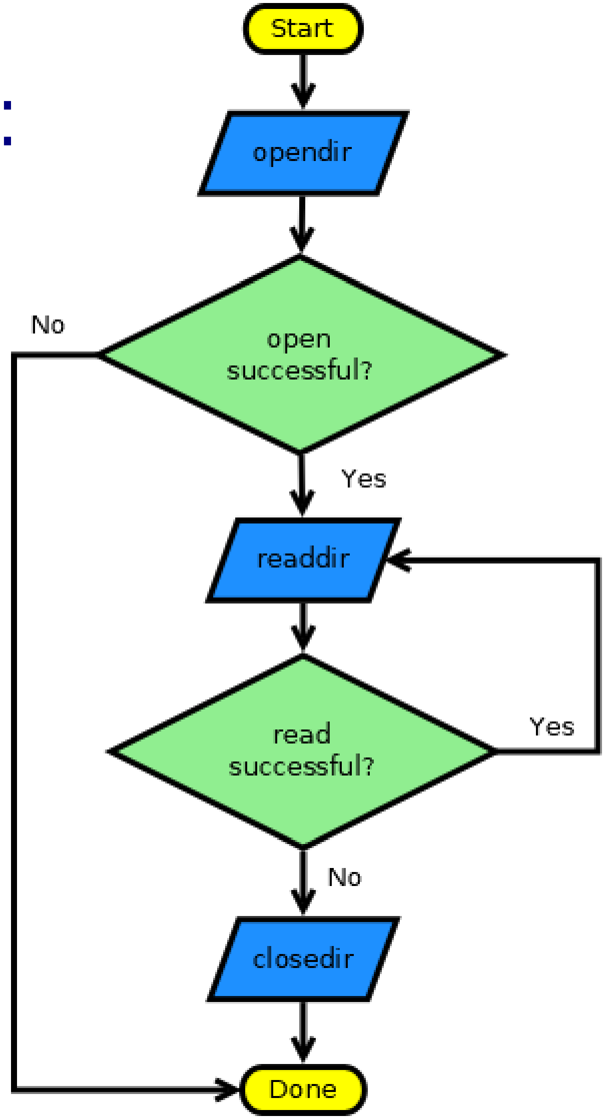


```
       DIR    A type representing a directory stream.

       It shall also define the structure dirent which shall include  the
       following 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}.

:
```

Each dirent struct has a d_name member, which stores the filename.

Q3: Can I write ls?

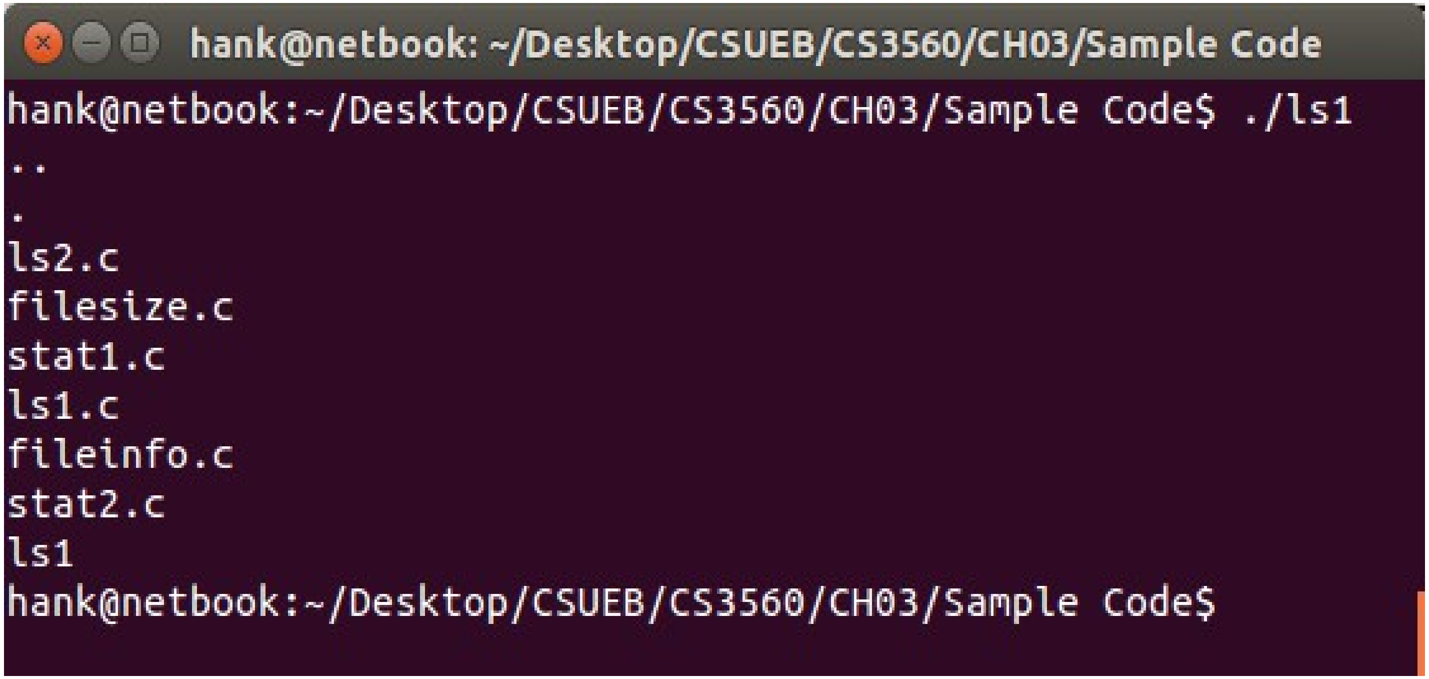
Here's the logic:



Let's examine a first attempt, ls1.c.

Q3: Can I write ls?

- Compile and run it and compare it's output to the real ls output:



```
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ ./ls1
..
.
ls2.c
filesize.c
stat1.c
ls1.c
fileinfo.c
stat2.c
ls1
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$
```

How'd it go?

- Version 1.0 lists the files, but has issues:
  Not sorted.

  Fix: read into an array, and run a sorting algorithm.

  No columns.

  Fix: read into an array, then figure out column widths and heights.

How'd it go?

Lists '.' files, but files beginning with '.' supposed to be "hidden". ls simply ignores these files.

Fix: easy to suppress and add -a option.

No -l info.

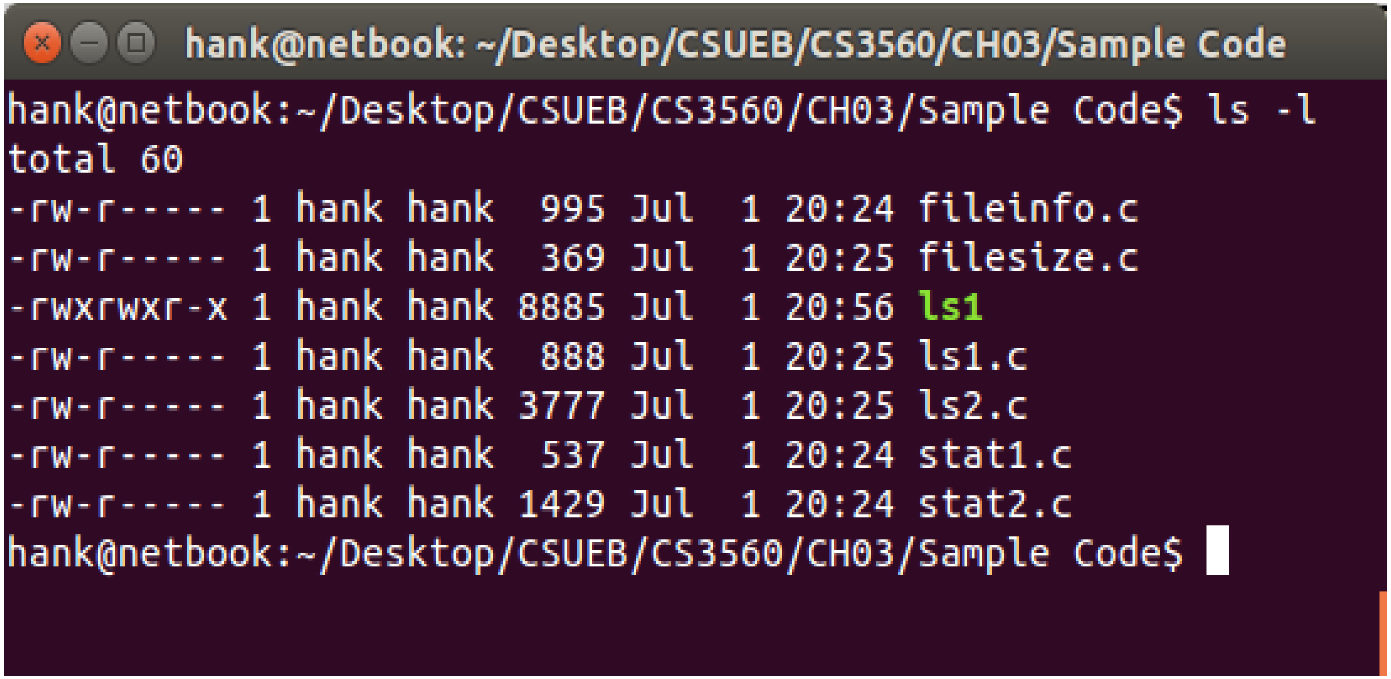
Fix: Adding -l not easy. struct dirent is missing a lot of info. Where is it?

Project 2: Writing ls -l

- Let's continue using our three question approach.
- Finding and displaying information about files is different than listing file names.
- A separate project.

What does ls -l do?

- Sample output:



```
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ ls -l
total 60
-rw-r----- 1 hank hank  995 Jul  1 20:24 fileinfo.c
-rw-r----- 1 hank hank  369 Jul  1 20:25 filesize.c
-rwxrwxr-x 1 hank hank 8885 Jul  1 20:56 ls1
-rw-r----- 1 hank hank  888 Jul  1 20:25 ls1.c
-rw-r----- 1 hank hank 3777 Jul  1 20:25 ls2.c
-rw-r----- 1 hank hank  537 Jul  1 20:24 stat1.c
-rw-r----- 1 hank hank 1429 Jul  1 20:24 stat2.c
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$
```

What does ls -l do?

- Each line consists of seven fields:

  mode:

  The first character in each line is the file *type*.
  '-' indicates a regular file.
  'd' indicates a directory

  The remaining characters indicate access permissions.

What does ls -l do?

Links

A reference (shortcut) to a file. Covered later.

Owner

Which user owns the file.

Group

Which group owns the file. A file belongs to only one group at a time. Users can belong to multiple groups.

What does ls -l do?

Size

The number of bytes in the file. Storage for directories is allocated in blocks, so directory size is always a multiple of 512. Regular files, it's the number of bytes of data in the file.

Last-modified

The last time the file was modified.

Name

The name of the file. Duh.

Q2: How does ls -l work?

Answer: the stat call gets file information.

stat(name, ptr)

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

Some notes about files

- files are stored on disk
- Files have contents and a set of attributes
- When a process wants info about a file, it defines a place to store that information.
- This place is a buffer of type *struct stat*.
- It then asks the kernel to copy that info from disk to the buffer.

stat

PURPOSE: get info about a file

INCLUDE: #include <sys/stat.h>

USAGE:

```
int result = stat(char* fn, struct stat* bufp)
```

ARGS:	fn	name of file
	bufp	pointer to buffer
RETURNS:	-1	if error
	0	if success

stat

- man 2 stat gives complete details on the stat struct.
- Let's examine stat.c to see how stat can find the size of a file.

What else can stat tell us?

- The stat man page and the stat.h header describe the *struct stat* members:

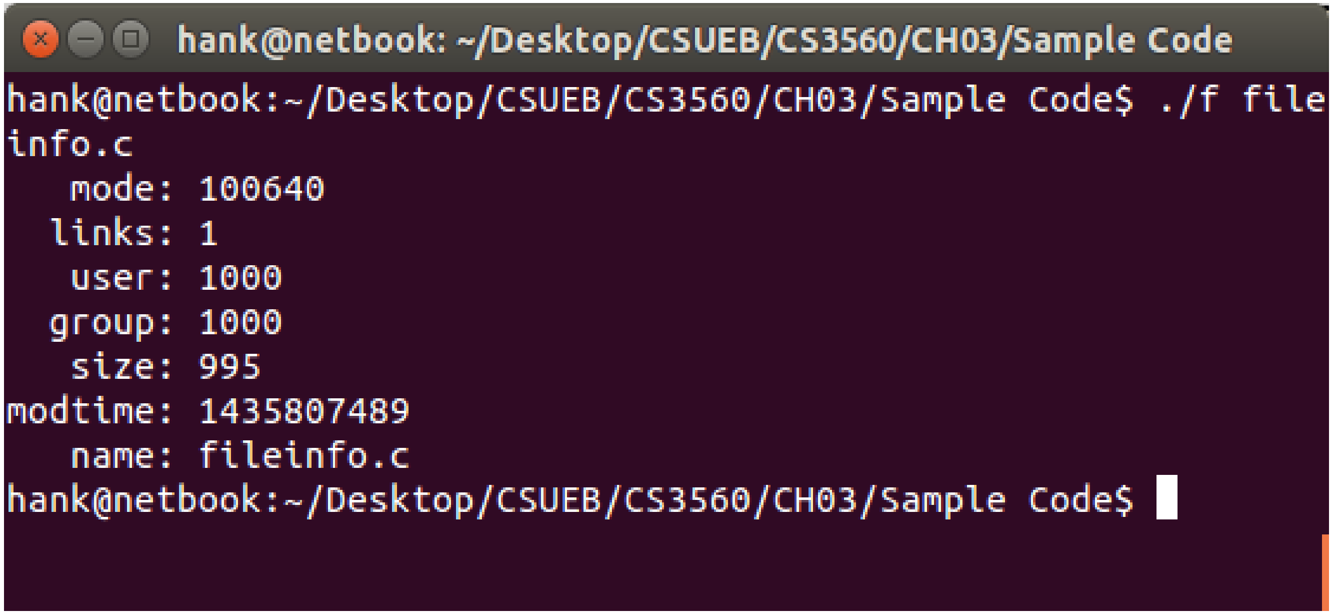
st_mode	type and permissions
st_uid	ID of owner
st_gid	ID of group
st_size	number of bytes in the file
st_nlink	number of links to the file
st_mtime	last content-modification time
st_atime	last-accessed time
st_ctime	last properties-changed time

What else can stat tell us?

- There are other members, but these are what ls -l tells us.
- Let's take a look at a sample program that will retrieve and print those attributes, fileinfo.c.

What else can stat tell us?

If we compile and run it and compare it to ls -l,



```
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ ./f file
info.c
   mode: 100640
  links: 1
   user: 1000
  group: 1000
   size: 995
modtime: 1435807489
   name: fileinfo.c
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ 
```

How'd we do?

- *Links, size,* and *name* work perfectly. *Modified-time* is a time_t. Use ctime to fix that.
- We printed *mode* as a number, but ls does something like:

  ```
  -rw-rw-r--
  ```
- *User* and *group* are numbers, but ls shows names.
- Need to convert user, group, and mode to strings.

Converting File Mode to a String

- st_mode is a 16-bit quantity.
- Separate attributes are encoded in substrings of these 16 bits.

_ _ _ _	_ _ _	_ _ _	_ _ _	_ _ _
type	sgid suid sticky	user	group	other

Converting File Mode to a String

- The first four bits represent the type of the file.
- The next three bits are used for special attributes of a file. A '1' turns the attribute on, '0' turns it off. These bits are *set-user-ID*, *set-group-ID*, and *sticky*. More on them later.
- The final 9 bits are broken into thirds.
- The first third represent owner permissions, second represent group, third other.

Converting File Mode to a String

- A value of 1 means the permission is granted, a value of 0 means it's not.
- There are special meanings into subfields of this larger string.
- How do we read the substrings? We'll zero out all the other digits other than what we want to view. This way, what we want will “show through”.

Converting File Mode to a String

- This technique is called *masking*.
- If you wanted to focus on your eyes, you'd cover your face with a mask that covers everything but your eyes. This is the idea.
- We'll use a set of masks to translate st_mode into the string displayed by ls -l.
- Very common in systems programming, need to know four things....

How to Read Subfields: Masking

- *Thing One: The Concept of Masking* Masking a value is zeroing out bits in the number so only a subfield is unaffected.
- *Thing Two: An Integer is just a string of bits*.

215 =	0	1	1	0	1	0	1	1	1
		128	64	32	16	8	4	2	1

How to Read Subfields: Masking

-

? =	0	0	0	0	1	1	0	1	0
	256	128	64	32	16	8	4	2	1

- *Thing three: The Technique of Masking*
  The *bitwise* and operation & causes one value to mask another.
  For example:

```
        0100 1100
&       0001 1101
  =     0000 1100
```

How to Read Subfields: Masking

- *Thing Four: Using Base 8*
  Working with binary masks is tedious, especially for 16 or 32 bit long masks.

  To make it easier, we group large binary numbers into three-digit bunches and convert each into a single octal digit, 0-7.

How to Read Subfields: Masking

- For example,

1000000110110100 into 3's as

1, 000, 000, 110, 110, 100

and translate each bunch to get

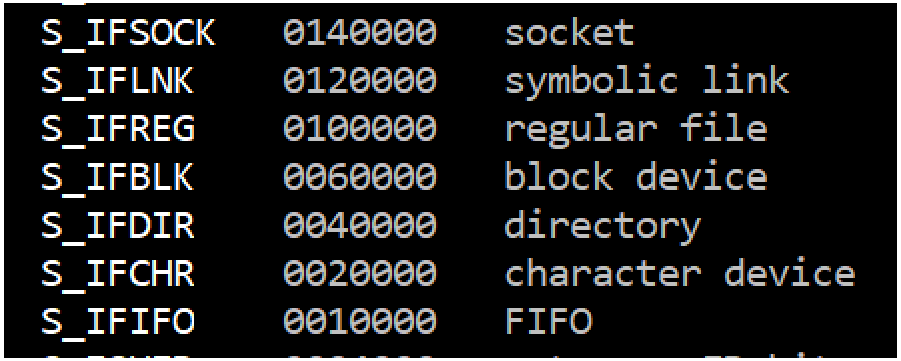
0100664

easier to work with.

Using Masking to Decode File Types

- The type of a file is coded in the mode's first four bits.
- We use masks to decode the information
- A mask zeroes all but the first four bits, then we compare the result to codes for each file type.
- The definitions are in <sys/stat.h>
- Also may be viewable in man 2 stat

Using Masking to Decode File Types



```
S_IFSOCK   0140000   socket
S_IFLNK    0120000   symbolic link
S_IFREG    0100000   regular file
S_IFBLK    0060000   block device
S_IFDIR    0040000   directory
S_IFCHR    0020000   character device
S_IFIFO    0010000   FIFO
```

Using Masking to Decode File Types

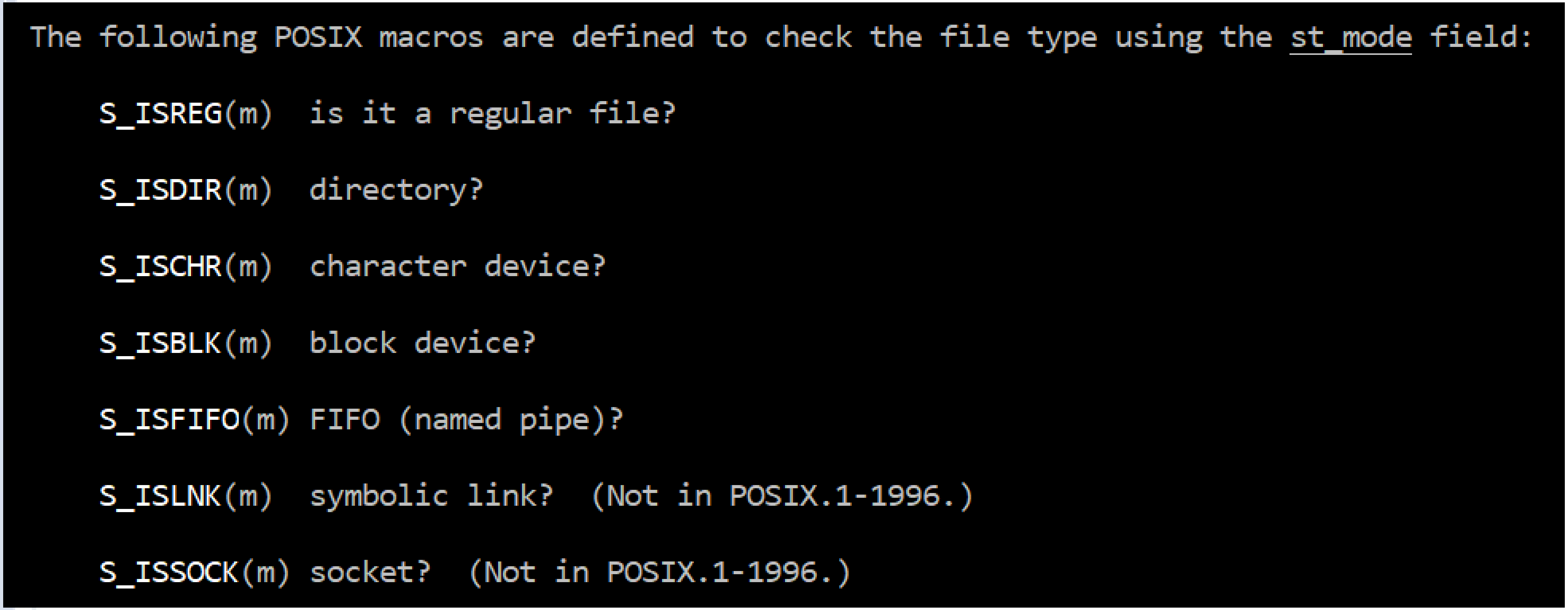
So, to check if a mode specifies a directory, we could use a fragment like this:

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

What's 0170000 and 0040000?
    See man 2 stat

Using Masking to Decode File Types

- Instead of writing this code, we can use macros in <sys/stat.h>, listed in man pages:



```
The following POSIX macros are defined to check the file type using the st_mode field:

    S_ISREG(m)  is it a regular file?

    S_ISDIR(m)  directory?

    S_ISCHR(m)  character device?

    S_ISBLK(m)  block device?

    S_ISFIFO(m) FIFO (named pipe)?

    S_ISLNK(m)  symbolic link?  (Not in POSIX.1-1996.)

    S_ISSOCK(m) socket?  (Not in POSIX.1-1996.)
```

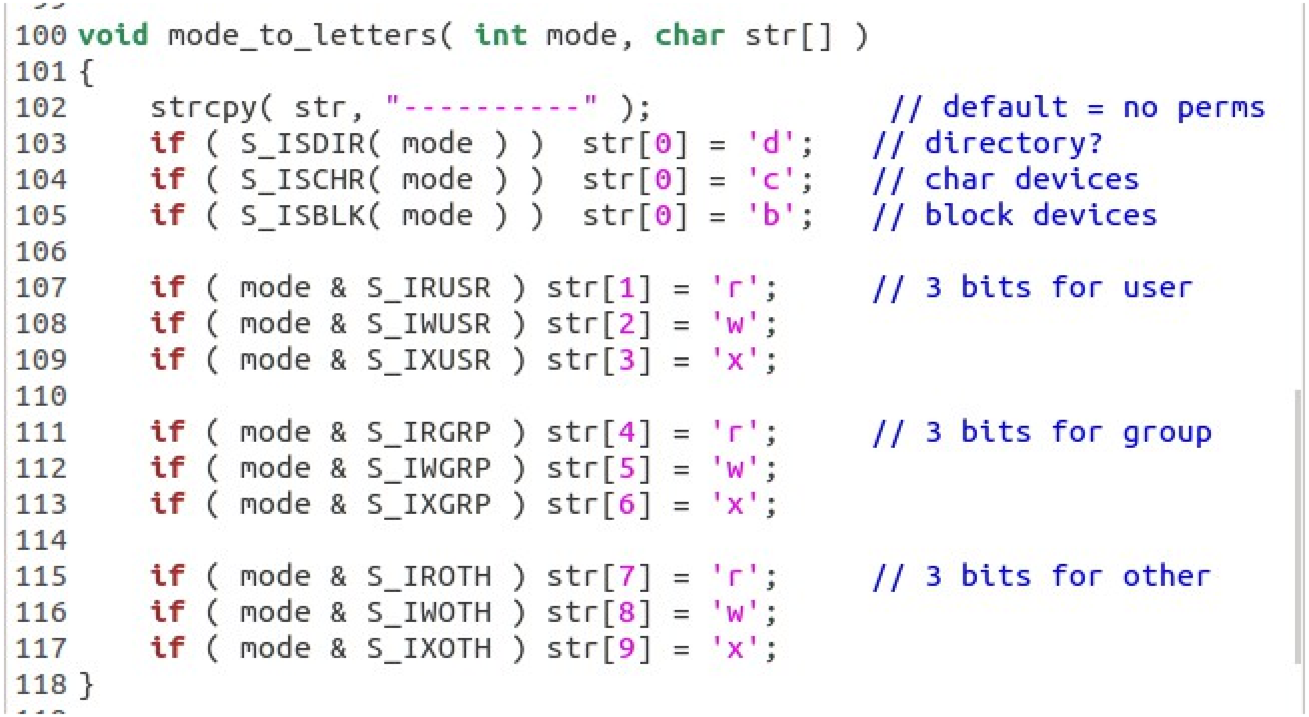
Using Masking to Decode File Types

Thanks to these macros, we can write:

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

Using Masking to Decode Permission Bits

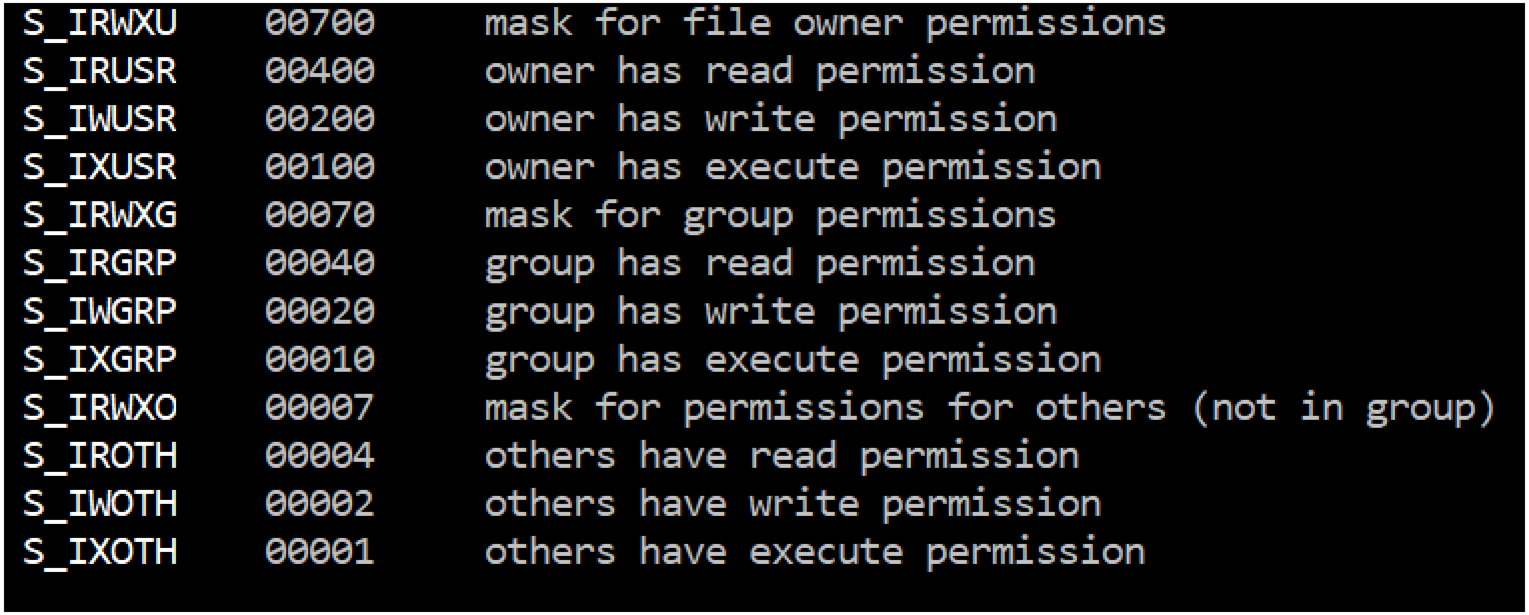
Each bit has a mask available in stat.h. Here's code to test each bit separately:



```
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 devices

    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';
}
```

Using Masking to Decode Permission Bits

From the stat man page :

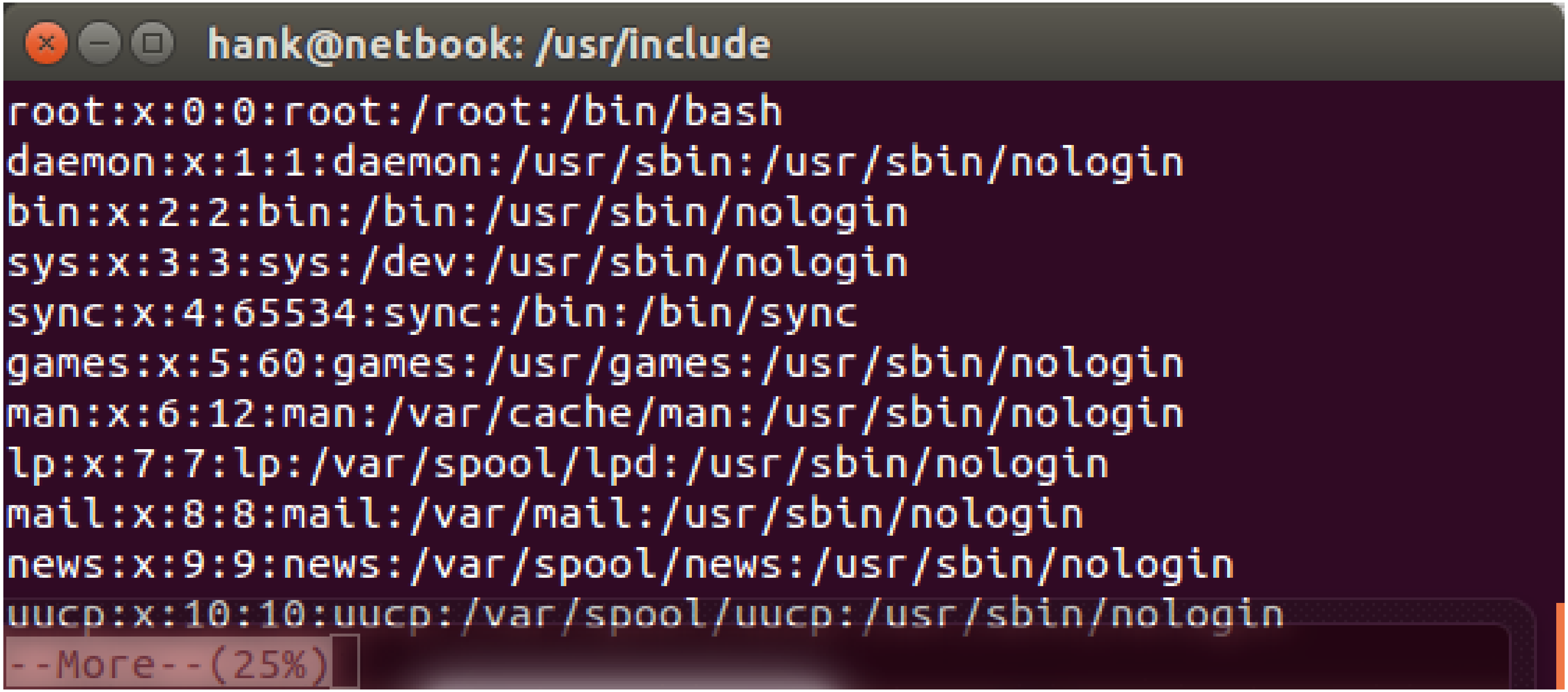


```
S_IRWXU     00700     mask for file owner permissions
S_IRUSR     00400     owner has read permission
S_IWUSR     00200     owner has write permission
S_IXUSR     00100     owner has execute permission
S_IRWXG     00070     mask for group permissions
S_IRGRP     00040     group has read permission
S_IWGRP     00020     group has write permission
S_IXGRP     00010     group has execute permission
S_IRWXO     00007     mask for permissions for others (not in group)
S_IROTH     00004     others have read permission
S_IWOTH     00002     others have write permission
S_IXOTH     00001     others have execute permission
```

Converting User/Group ID to Strings

- The owner and group of the file are stored as numbers: *user id* and *group id*.
- Man pages show us a ton of info, see what you find, but here are some facts....

Fact One: /etc/passwd is the list of users



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
--More--(25%)
```

Fact One: /etc/passwd is the list of users

- /etc/passwd is a plain text file listing all users and their ids.
- The first field is the user name, third field is the user id, and the fourth field is the group to which the user belongs.
- This could work, but searching this file is a bit tedious and might not work on all systems.

Fact Two: /etc/passwd is not always a complete list of users

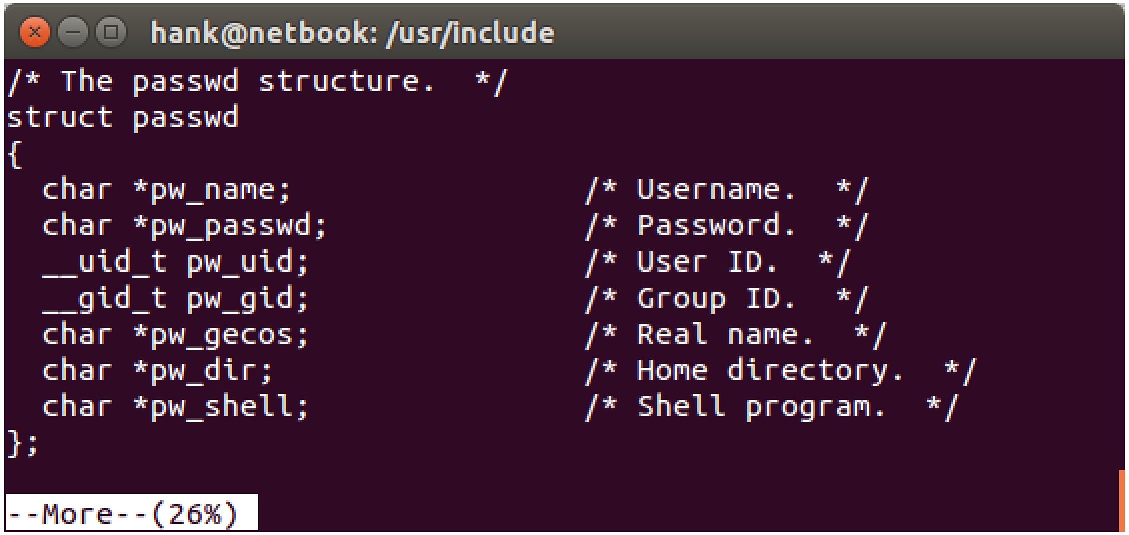
- Every Unix system has this file, but often incomplete.
- To support networks, a minimal /etc/passwd file exists for local users, and a separate, centralized database for network users.
- Programs requiring user info consult this central database.

Fact Three: getpwuid provides access to the complete list of users

- getpwuid() provides access to user information.
- Searches /etc/passwd and the centralized database.
- Accepts UID as an argument and returns *struct passwd*,* defined in /usr/include/pwd.h

Fact Three: getpwuid provides access to the complete list of users

The structure of /usr/include/pwd.h:

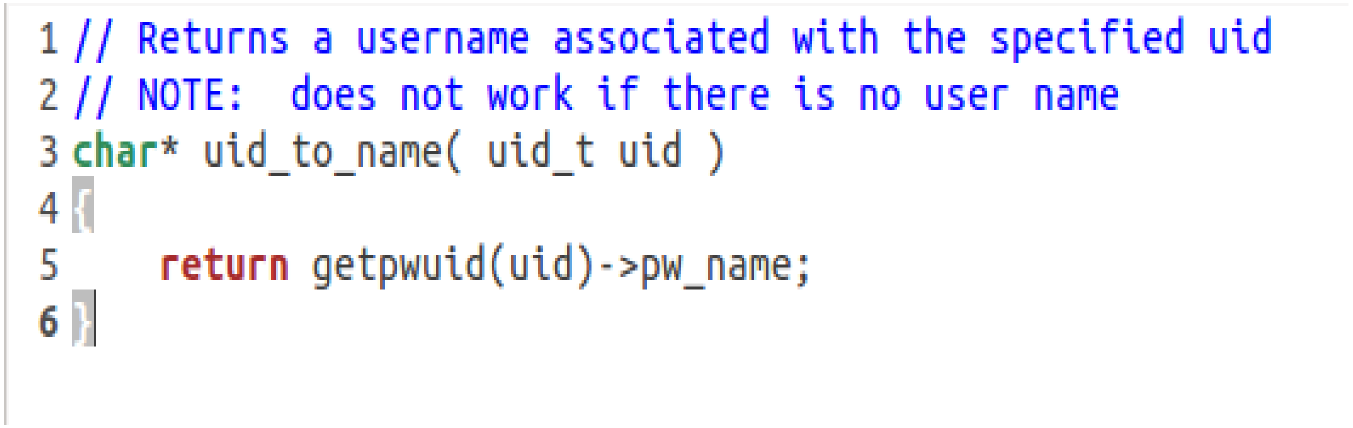


```
/* The passwd structure.  */
struct passwd
{
  char *pw_name;                /* Username.  */
  char *pw_passwd;              /* Password.  */
  __uid_t pw_uid;               /* User ID.  */
  __gid_t pw_gid;               /* Group ID.  */
  char *pw_gecos;               /* Real name.  */
  char *pw_dir;                 /* Home directory.  */
  char *pw_shell;               /* Shell program.  */
};

--More--(26%)
```

We have the information we need to fill in the user name for the long format of ls.

Fact Three: getpwuid provides access to the complete list of users

- Here's a simple solution:



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

- Solution is simple, but not robust. Returns NULL if there is no pw_name member reference. Is has a solution.

Fact four: Some UIDs do not correspond to lognames

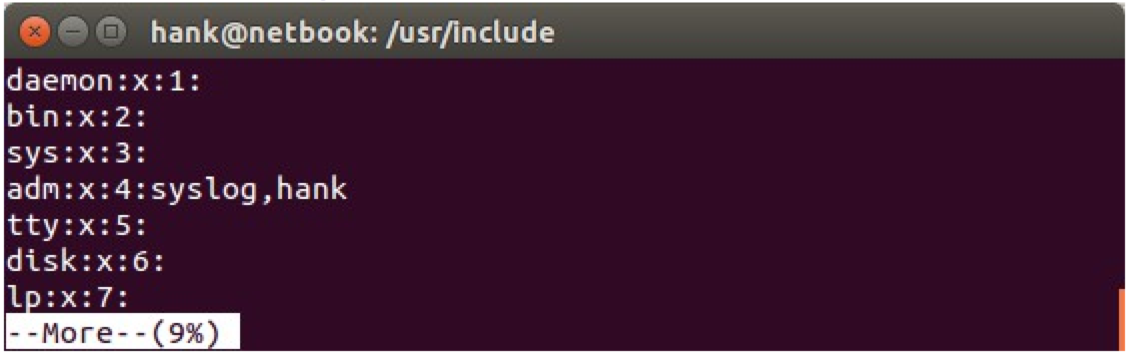
- Say you have an account on a Unix machine with username pat and ID 2000.
- Files you create are owned by you.
- stat returns a struct for your files with 2000 in the st_uid field.
- That number is an attribute of the file.

Fact four: Some UIDs do not correspond to lognames

- Say you move, and the admin deletes your account, removing association of pat with 2000.
- What happens if a new user is added and assigned the user id 2000?
- Any files you left behind are now owned by that user who has permission to read, write, and delete those files.

Fact Five: /etc/group is the list of groups

- Unix provides a system for defining groups and assigning users to groups.
- The file /etc/group is a plain text file:



```
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,hank
tty:x:5:
disk:x:6:
lp:x:7:
--More--(9%)
```

- Line format :

  group name:password:gid:members

Fact Six: A user can belong to more than one group

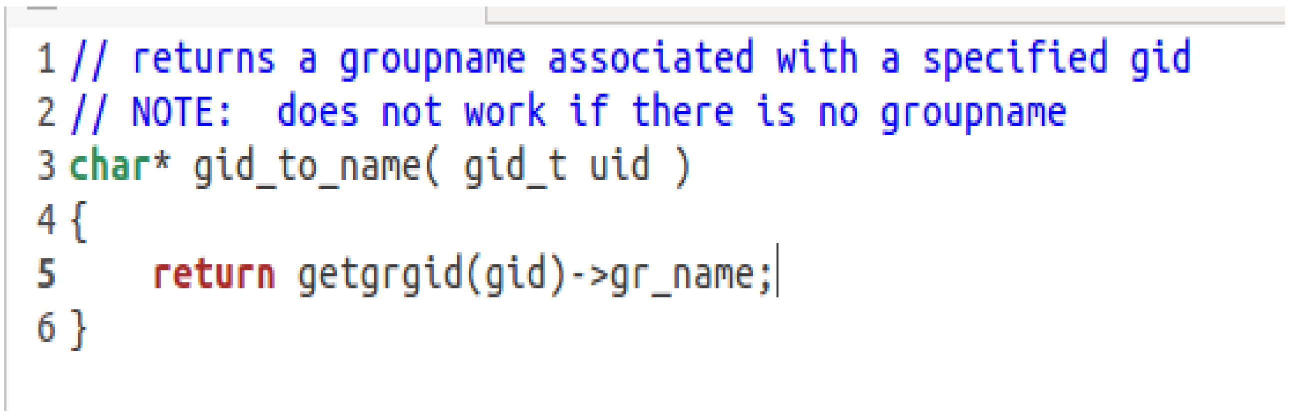
- Adding a username to an entry in /etc/group adds the user to that group.
- This list is used with permission bits for group access.
- If a file and user belong to the same group, and the file is group writable, the user can write to it.

Fact Seven: getgrgid provides access to the list of groups

- Data in /etc/group can also be moved to a central database.
- Accessible using getgrgid().
- man getgrgid for more info.

Fact Seven: getgrgid provides access to the list of groups

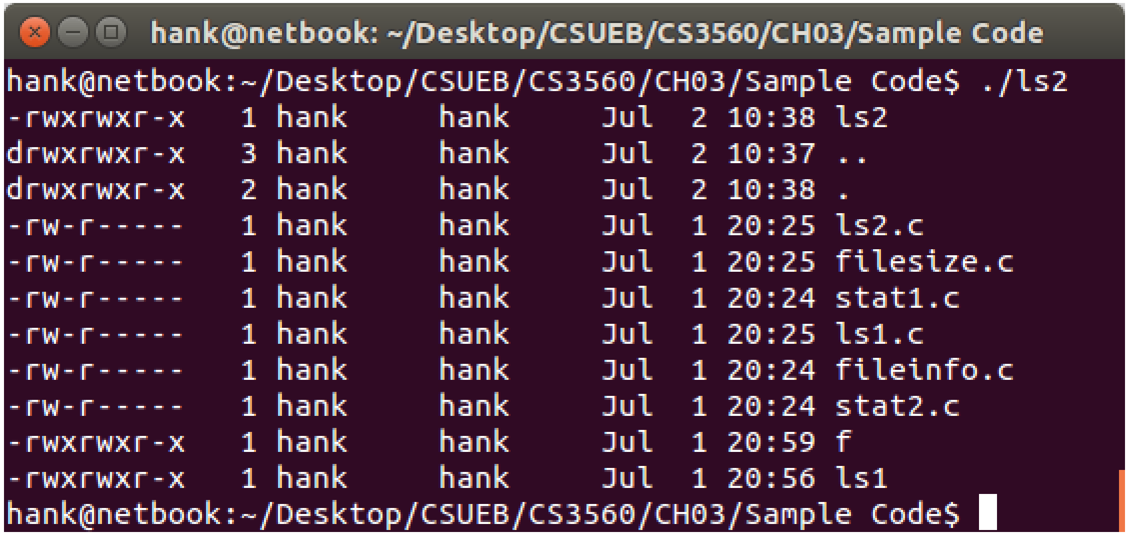
To solve our problem, we can use code like this:



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

Putting it All Together: ls2.c

- Let's take a look at the code of ls2.c, incorporating everything we've learned.
- Here's an example run:



```
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ ./ls2
-rwxrwxr-x   1 hank     hank     Jul  2 10:38 ls2
drwxrwxr-x   3 hank     hank     Jul  2 10:37 ..
drwxrwxr-x   2 hank     hank     Jul  2 10:38 .
-rw-r-----   1 hank     hank     Jul  1 20:25 ls2.c
-rw-r-----   1 hank     hank     Jul  1 20:25 filesize.c
-rw-r-----   1 hank     hank     Jul  1 20:24 stat1.c
-rw-r-----   1 hank     hank     Jul  1 20:25 ls1.c
-rw-r-----   1 hank     hank     Jul  1 20:24 fileinfo.c
-rw-r-----   1 hank     hank     Jul  1 20:24 stat2.c
-rwxrwxr-x   1 hank     hank     Jul  1 20:59 f
-rwxrwxr-x   1 hank     hank     Jul  1 20:56 ls1
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$
```

How'd we do?

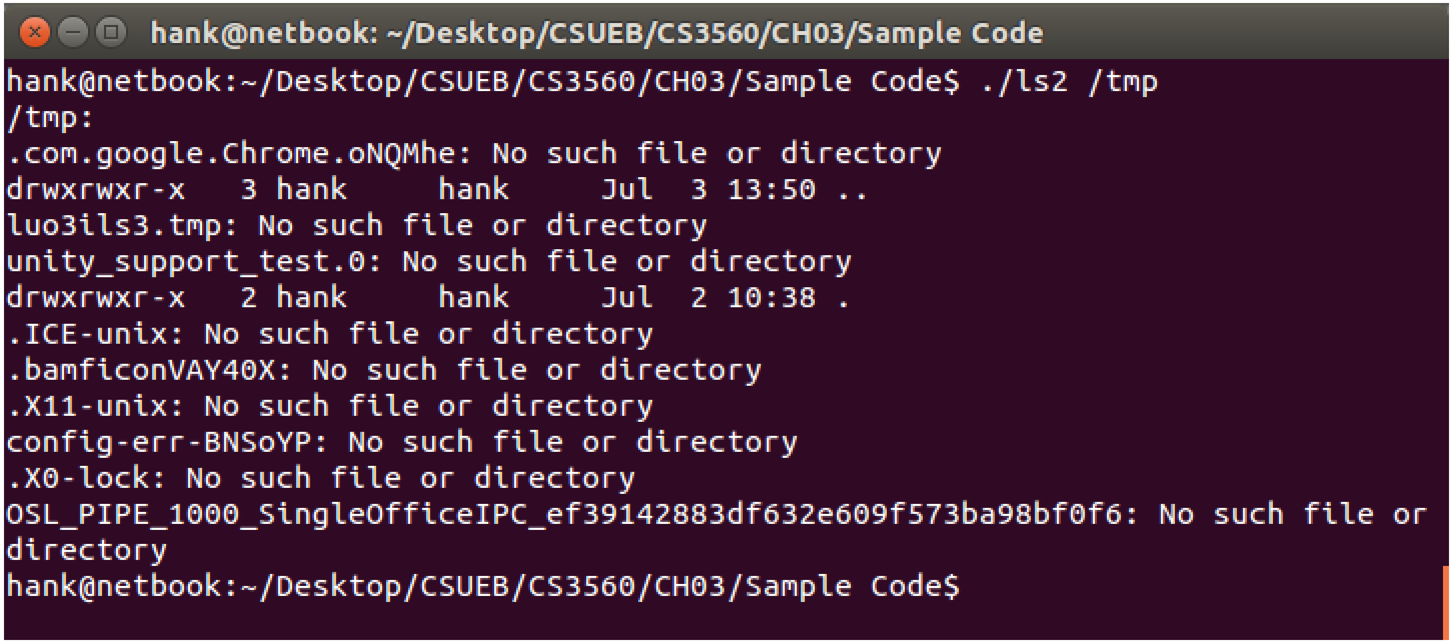
- Displays info in the standard ls -l format.
- Output looks good.
- Columns line up.
- Translation from bit patterns and ID numbers to strings works.

How'd we do?

- Needs more work though.
- The *real* ls prints total line at the top.
- Still doesn't sort file names.
- Still doesn't handle -a.
- Doesn't arrange names into columns.
- Assumes each argument is a file name.

How'd we do?

- Worse, ls2 doesn't list files in *other* directories correctly. Fixing this is left as an exercise.
- Try ls2 /tmp:



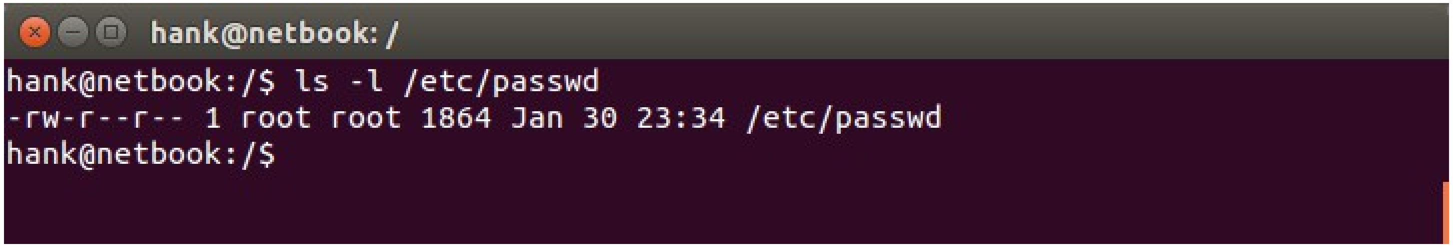
```
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$ ./ls2 /tmp
/tmp:
.com.google.Chrome.oNQMhe: No such file or directory
drwxrwxr-x   3 hank     hank     Jul  3 13:50 ..
luo3ils3.tmp: No such file or directory
unity_support_test.0: No such file or directory
drwxrwxr-x   2 hank     hank     Jul  2 10:38 .
.ICE-unix: No such file or directory
.bamficonVAY40X: No such file or directory
.X11-unix: No such file or directory
config-err-BNSoYP: No such file or directory
.X0-lock: No such file or directory
OSL_PIPE_1000_SingleOfficeIPC_ef39142883df632e609f573ba98bf0f6: No such file or
directory
hank@netbook:~/Desktop/CSUEB/CS3560/CH03/Sample Code$
```

The three special bits

- st_mode member of structure stat has 16 bits. Four are used for type, nine are used for access permissions.
- The three others are used for special file properties.

The set-user-id bit

- How can a regular user change their password?

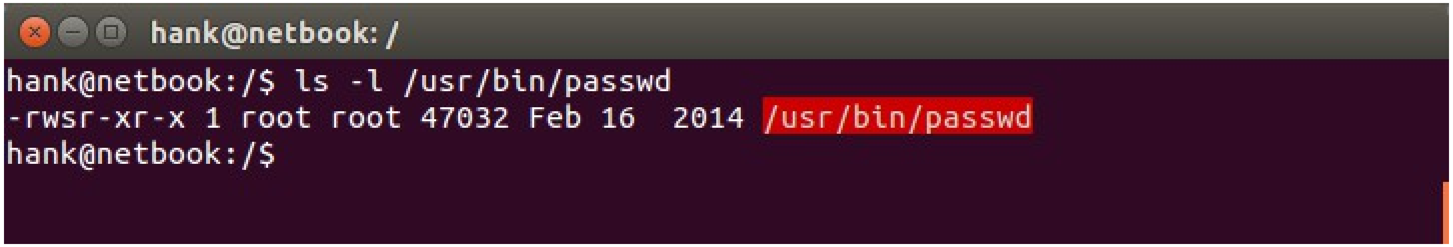


```
hank@netbook:/$ ls -l /etc/passwd
-rw-r--r-- 1 root root 1864 Jan 30 23:34 /etc/passwd
hank@netbook:/$
```

- Owner of this file is *root*. Regular user isn't root.

The set-user-id bit

- Solution? Give permission to the program to change the password, not the regular user.



```
hank@netbook:/$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 Feb 16  2014 /usr/bin/passwd
hank@netbook:/$
```

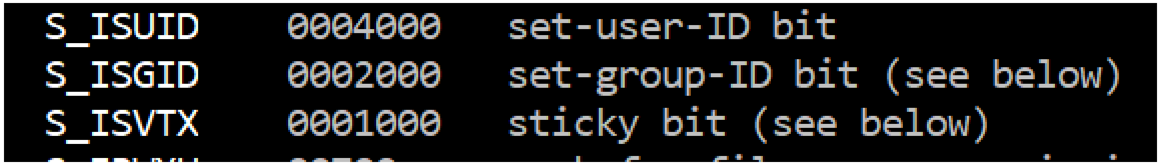
- The SUID bit tells the kernel to run the program as though it were being run by the owner of the program.

The set-user-id bit

User root owns /etc/passwd so programs running as root can modify it.

The passwd command knows who you are, so you can't change other user passwords.

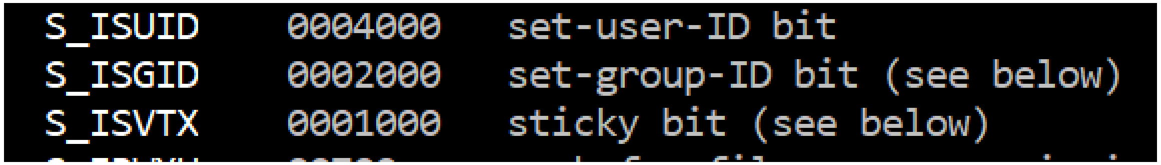
We can test whether the SUID bit is on by using the appropriate mask in <sys/stat.h>



```
S_ISUID     0004000   set-user-ID bit
S_ISGID     0002000   set-group-ID bit (see below)
S_ISVTX     0001000   sticky bit (see below)
```

The set-group-id bit

- Similar to set-user-id bit, but for groups.
- Can test using the following mask:



```
S_ISUID     0004000   set-user-ID bit
S_ISGID     0002000   set-group-ID bit (see below)
S_ISVTX     0001000   sticky bit (see below)
```

The sticky bit

- Some directories are meant to hold temporary files.
- For example, /tmp, and they are publicly writable, so anyone can *delete* files there.
- The sticky bit changes this so only owners of a file in the “sticky” directory can delete them.

The special bits and ls -l

- Each file has 12 attribute bits, but ls uses only 9 spots to display them. How does that work?
- man ls has details, but consider this example display:

  ```
  -rwsr-sr-t
  ```

  s is used in place of x to indicate the group executable bits have been augmented by set-user and set-group ID bits.

  t means the sticky bit is on.

Summary of ls

- *Directories and files*
  Data is stored in files. Internally, directories are special kinds of files; directories contain a list of names of files. Even contains a name for itself. Unix provides system calls to open, read, seek, and close directories, but not a write call.

Summary of ls

- *Users and Groups*
  Each user is assigned a username and an id number. Users use the username to login and communicate with other users. System uses UID to identify owner of files. Users belong to groups and each group has a group name and id number.

Summary of ls

- *File Attributes*
  Every file has properties. A program can get these properties using the stat system call.
- *File Ownership*
  Files have an owner. The UID of the owner is a property of the file. Files belong to groups. The GID is a property of the file.

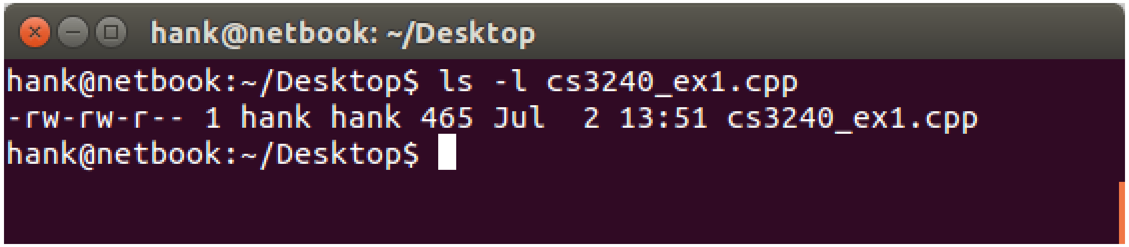
Summary of ls

- *Access Permissions*
  Users can read, write, and execute files. Each file has a set of permission bits to determine what users can do what operations. Permissions to read, write, and execute a file are controlled at the user, group, and other levels.

Setting and Modifying the Properties of A File

A typical file

- ls -l shows us several properties of a file.
- Let's consider these properties from left to right of this file:



```
hank@netbook:~/Desktop$ ls -l cs3240_ex1.cpp
-rw-rw-r-- 1 hank hank 465 Jul  2 13:51 cs3240_ex1.cpp
hank@netbook:~/Desktop$ 
```

Type of a File

- A file has a type. It can be a regular file, a directory, a device file, a socket, a symbolic link, or a named pipe.
- The type of file is established when the file is created.
- 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.
- The bits are set when the file is created and can be modified using the chmod system call.
- Consider:

  ```
  fd = creat(“newfile”, 0744);
  ```

  the second argument requests an initial set of permission bits   rwxr--r--

Permission Bits and Special Bits

- That second argument is a request, not an order.
- That requested mode has a mask applied to it by the kernel. The result is the bits actually set.
- This mask is called the *file-creation mask*, it specifies which bits should be turned *off*.

Permission Bits and Special Bits

- For example, if you want to prevent programs from creating files that can be modified by the group or others, you want to turn off:

  ```
  ----w--w-
  ```

  which is 022 in octal.
- Then, make the system call umask(022);
- This specifies which bits to turn off, which is backwards from what we've seen so far.

Changing the Mode of a File

- Programs can modify the permission and special bits, using the *chmod* system call.
- Here are two examples that do the same thing:

```
                chmod(“myfile”, 04764 );

chmod(“myfile”, S_ISUID | S_IRWXU | S_IRGRP | S_IWGRP |
           S_IROTH);
```

Changing the Mode of a File

- In the first case, the new bit set is expressed in octal.
- In the second case, the masks from stat.h are combined using *bitwise or* into a single value
- The second case is more portable.
- chmod is not affected by the *file-creation mask*.

chmod

PURPOSE: change permission and special bits of 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 the mode

RETURNS:

-1	if error
0	if success

A Shell Command

- The shell command chmod is a regular command-line program for modifying the bits.
- Can specify the bit pattern in octal (e.g., 04764) or symbolic notation (e.g., u=rws g=rw o=r)
- man chmod for more info

Number of Links to a File

- More on this later, but the number of links is simply the number of times the file is referenced in directories.
- To increase this count, make more references (use link).
- To decrease, remove references (use unlink).

Owner and Group of a File

- The owner of a file is the user who creates it.
- Technically, the kernel creates it when a process executes the creat system call.
- The kernel sets the owner of the file to the effective user ID of the process that calls creat.

Owner and Group of a File

Usually, the group of a file is set to the effective group ID of the process creating the file.

Unusually, the group ID is set to the group ID of the parent directory.

Changing the Owner and Group

- A program can do it using the chown system call:

  ```
  chown(“filename”, 200, 40);
  ```

  Changes user id to 200, and group id to 40 for file “filename”. If either is -1, that attribute does not change.
- The owner of a file can change the group ID to any group which the user belongs.
- The super user (system admin, root user) can change either to anything.

chown

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

A Shell Command

- chown and chgrp are shell commands for modifying the user id and group id for files
- See manpage for more details.
- chown and chgrp allow the user to specify the IDs as nubers or as usernames and group names.

Size of a File

- The size of a file, directory, and named pipe (talk about pipes later) represents the number of bytes stored.
- Programs increase the size of files by adding data to it.
- Programs can zero the size of a file by using creat.
- May not shorten a file to a nonzero length.

Modification and Access Time

- Files have three timestamps: last modification time, last read time, and time file properties were last changed.
- Kernel automatically updates these times as programs read/write the file.
- You can write programs to set these values to arbitrary times.

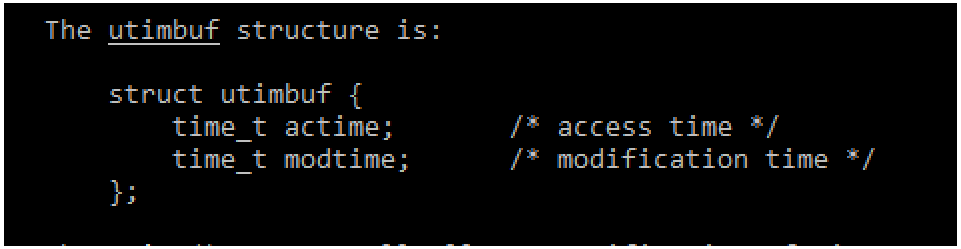
Changing Modification/Access Times

utime system call sets file modification and access times.

To use utime, create a struct containing two time_t elements, one for the access time and one for the modification time.

Then, call utime with the name of the file and a pointer to that struct.

Kernel sets these times to the values you specify.



```
The utimbuf structure is:

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

utime

PURPOSE: change file access and mod time

INCLUDE:

```
#include <sys/time.h>
#include <sys/types.h>
#include <utime.h>
```

USAGE: int utime(char* path, struct utimbuf* n)

ARGS:

path	path to a file
n	pointer to a struct utimbuf see utime.h for details

RETURNS:

-1	if error
0	if success

A Shell Command

- touch is used for setting the modification and access times for files.
- As usual, consult the manpage for more info.

Name of a File

- When you create a file, you give it a name.
- The mv command allows you to change the name and move a file from one directory to another.
- The creat system call sets the name and initial mode for a file
- The rename system call changes the name.

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

Main Ideas

- A disk contains files and directories. Files have contents and properties. Directories also have contents and properties. A file can contain any sort of data. A directory can only contain a list of names.
- The names in a directory refer to files and to other directories. The kernel provides system calls for reading the contents of directories, for reading the properites of files, and for modifying most file attributes.

Main Ideas

- The type of a file, the access permissions, and three special attributes are stored as bit patterns in an integer. Bit masks are used to examine bit patterns.
- The owner and group of a file are stored as numbers. The relationship between these numbers and user names and group names is established by the passwd and group databases.

Ciao!