



# Chapter 4: Files and Directories

---

CMPS 105: Systems Programming  
Prof. Scott Brandt  
T Th 2-3:45  
Soc Sci 2, Rm. 167



# Files and Directories

---

- Chapter 3 covered basic file I/O
- Chapter 4 covers more details
  - stat
  - File attributes
  - Special files
  - Directories



# Stat, fstat, lstat

---

- `sys/types.h, sys/stat.h`
- `int stat (const char *pathname, struct stat *buf)`
- `int fstat(int fildes, struct stat *buf)`
- `int lstat(const char *pathname, struct stat *buf)`
- All three return 0 or -1 (on error)
- Provide information about the named file
  - `fstat` works on open files
  - `lstat` is like `stat`, but provides info about symbolic link on symbolic links



# Stat details

---

```
struct stat {
    mode_t      st_mode; /* file type and mode (perms) */
    ino_t       st_ino;  /* i-node number */
    dev_t       st_dev;  /* device number (filesystem) */
    dev_t       st_rdev; /* device number for special files */
    nlink_t     st_nlink; /* number of links */
    uid_t       st_uid;  /* user id of owner */
    gid_t       st_gid;  /* group id of owner */
    off_t       st_size; /* size in bytes, for regular files */
    time_t      st_atime; /* time of last access */
    time_t      st_mtime; /* time of last modification */
    time_t      st_ctime; /* time of last file status change */
    long        st_blksize; /* best I/O block size */
    long        st_blocks; /* number of 512-byte blocks allocated */
};
```



# File Types I

---

- Regular files
  - Most common
  - Contain data (text, binary, etc.)
  - Kernel considers contents to be a stream of bytes (or blocks of bytes)
- Directory files
  - Contains the names of other files
  - Also contains pointers to other files
  - Read permission = read contents of directory
  - Write permission = create new files in the directory
  - Execute permission = access files in the directory



# File Types II

---

- Character special file
  - A type of file used for certain types of devices
  - Character-oriented devices: keyboard, mouse, ...
- Block special file
  - A type of file used for certain types of devices
  - Block-oriented devices: disk, tape, ...



# Device access via the file system

---

- Devices need to be accessible to processes
- Devices need to be nameable by processes
- Devices are generally read and written
- File systems provide all of this
  - We use the file system to interface to the devices
  - The read and write calls executed by the OS are specific to the individual devices



# File Types III

---

- FIFO

- A type of file used for interprocess communication (IPC) between files
- Also called a named pipe

- Socket

- A type of file used for network communication between processes
- Can also be used for processes on the same machine





# File Types IV

---

- Symbolic Link
  - A type of file that points to another file
- A hard link is a name for a file
  - Different hard links to the same file are really two different names for the file
- A soft link always contains the name of a file
  - It refers to the file indirectly through the “real” name of the file



# Determining file type

---

- File type is encoded in the `st_mode` member of the `stat` data structure
- Macros
  - `S_ISREG()` /\* regular file \*/
  - `S_ISDIR()` /\* directory file \*/
  - `S_ISCHR()` /\* character special file \*/
  - `S_ISBLK()` /\* block special file \*/
  - `S_ISFIFO()` /\* pipe or FIFO \*/
  - `S_ISLNK()` /\* symbolic link \*/
  - `S_ISSOCK()` /\* socket \*/

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

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

```
int main(int argc, char *argv[]) {
```

```
    int i;
```

```
    struct stat buf;
```

```
    char *ptr;
```

```
    for(i = 1; i < argc; i++) {
```

```
        printf("%s: ", argv[i]);
```

```
        if(lstat(argv[i], &buf) < 0) {
```

```
            err_ret("lstat error");
```

```
            continue;
```

```
        }
```

```
        if(S_ISREG(buf.st_mode)) ptr = "regular";
```

```
        else if(S_ISDIR(buf.st_mode)) ptr = "directory";
```

```
        else if(S_ISCHR(buf.st_mode)) ptr = "character special";
```

```
        else if(S_ISBLK(buf.st_mode)) ptr = "block special";
```

```
        else if(S_ISFIFO(buf.st_mode)) ptr = "FIFO";
```

```
    #ifdef S_ISLNK
```

```
        else if(S_ISLNK(buf.st_mode)) ptr = "symbolic link";
```

```
    #endif
```

```
    #ifdef S_ISSOCK
```

```
        else if(S_ISSOCK(buf.st_mode)) ptr = "socket";
```

```
    #endif
```

```
        else ptr = "unknown";
```

```
        printf("%s\n", ptr);
```

```
    }
```

```
    exit(0);
```

```
}
```



# File type frequencies

---

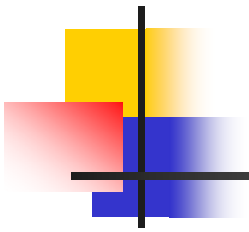
<b>File Type</b>	<b>Count</b>	<b>Percentage</b>
Regular file	30,369	91.7%
Directory	1,901	5.7%
Symbolic link	416	1.3%
Character special	373	1.1
Block special	61	0.2
Socket	5	0.0
FIFO	1	0.0

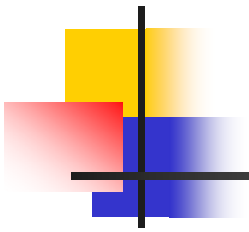


# Set-User-ID and Set-Group-ID

---

- Every process has six or more IDs
- Who we really are
  - Real user ID
  - Real group ID
  - Taken from our entry in the password file
  - Don't generally change

- 
- 
- Who we are currently pretending to be
    - Effective user ID
    - Effective group ID
    - Supplementary group IDs
    - Used for file access permission checks
    - Normally the same as the real user and group ID
    - Can be changed via set-uid and set-gid bits in programs
    - Passwd is a set-uid program

- 
- 
- Saved by `exec()` functions
    - Saved set-user-ID
    - Saved set-group-ID
    - Copies of effective user ID and effective group ID when a program is executed
    - Only meaningful when running a set-uid or set-gid program



# File Access Permissions

---

- `st_mode` also include access permissions for the file
- All file types have permissions
- Nine permission bits
  - `S_IRUSR` /\* user-read \*/
  - `S_IWUSR` /\* user-write \*/
  - `S_IXUSR` /\* user-execute \*/
  - `S_IRGRP` /\* group-read \*/
  - `S_IWGRP` /\* group-write \*/
  - `S_IXGRP` /\* group-execute \*/
  - `S_IROTH` /\* other-read \*/
  - `S_IWOTH` /\* other-write \*/
  - `S_IXOTH` /\* other-execute \*/





# Rules

---

- To open a file, must have execute permission on the directory
  - Directory read = read names of files
  - Directory execute = access files
- Read permission for a file determines if we can read a file
- Write permission for a file determines if we can write the file
  - Also needed for truncation
- To create a new file, must have write and execute permission for the directory
- To delete a file, must have write and execute permission for the directory
  - Do not need read or write permission for the file itself
- To execute a file, must have execute permission for the file and execute permission for the directory



# File Access Permission Checks

---

- If effective user ID is zero, access is allowed
- If the effective user ID = owner ID
  - If permissions allow access, access is allowed
  - Else, access is denied
- If the effective group ID (or one of the supplementary group IDs) = group ID of the file
  - If permissions allow access, access is allowed
  - Else, access is denied
- If the appropriate other access is allowed, access is allowed
- Else, access is denied



# Ownership of new files and directories

---

- The user ID of a new file is set to the effective user ID of the process that creates it
- The group ID of the new file will be either
  - The effective group ID of the process, or
  - The group ID of the parent directory



# Access function

---

- Unistd.h
- `int access(const char *pathname, int mode);`
- Checks to see if access is allowed
- Returns 0 or -1 (on error)
- Modes: `R_OK`, `W_OK`, `X_OK`, `F_OK` (existence)



# Umask function

---

- `Sys/types.h, sys/stat.h`
- `Mode_t umask (mode_t cmask);`
- Sets the file mode creation mask for the process
- Returns the previous value
- All subsequent file creates are filtered through `cmask`
- Any bits that are on in `cmask` are turned off in the file's mode



# Chmod and fchmod

---

- `sys/types.h, sys/stat.h`
- `int chmod(const char *pathname, mode_t mode);`
- `int fchmod(int fildes, mode_t mode);`
- Changes permission bits of a file
- Must be owner or superuser



# Sticky bit

---

- For files: used to keep the file in memory for later execution
- For directories: delete or rename of files in the directory can only be done by owner of file or directory (or superuser)



# Chown, fchown, and lchown

---

- `Sys/types.h, unistd.h`
- `Int chown(const char *pathname, uid_t owner, gid_t group);`
- `Int fchown(int fildes, uid_t owner, gid_t group);`
- `Int lchown(const char *pathname, uid_t owner, gid_t group);`
- Changes owner of a file (lchown: symlink)





# File size

---

- St\_size in stat structure
  - Only meaningful for regular files, directories, or sym links
  - Files: size in bytes
  - Directories: size in bytes
  - Sym links: size of filename linked to
- St\_blksize and st\_blocks
- Files with holes, ls, du, wc -c, cat core > core.copy (gets all of the zeroes)



# File truncation

---

- `sys/types.h, unistd.h`
- Open with `O_TRUNC`
- `int truncate(const char *pathname, off_t length);`
- `int ftruncate(int fildes, off_t length);`
- Truncates to length



# File systems

---

- See Section 4.14 (p.92) for pictures
- Partitions
- Data blocks
- Inodes
- Directories



# Link

---

- `#include <unistd.h>`
- `int link(const char *pathname, const char *newpath);`
- Creates a new directory entry for the file `<pathname>`
- Only superuser can link to directories
- increments link count



# unlink

---

- `#include <unistd.h>`
- `int unlink(const char *pathname);`
- Removes a link to a file
- Decrements the link count
- If link count = 0, removes the file
- The file stays around as long as any process has it open!
  - Useful if a program wants to guarantee that its temporary files go away after it terminates
- Unlink removes symbolic links



# remove

---

- `#include <stdio.h>`
- `int remove(const char *pathname);`
- Identical to `unlink`
- Removes directories



# Rename

---

- `#include <stdio.h>`
- `int rename(const char *pathname, const char *newname);`
- Renames files and directories
- In general, `newname` is deleted
  - If directory, must be empty
- Permissions must allow deletion of `pathname` and creation of `newname`



# Symbolic Links

---

- Indirect link to a file
- Contains the *name* of the file it links to
- Different from hard links, which are additional names for the same file
- Symbolic links (also called soft links) are evaluated at the time they are referenced
- Can create loops!





# symlink/readlink

---

- `#include <unistd.h>`
- `int symlink(const char *actualpath, const char *sympath);`
- Creates a symbolic link
- `int readlink(const char *pathname, char *buf, int bufsize);`
- Reads the contents of a symbolic link (not the file it links to)



# File times

---

- `st_atime`: last access time of the file
- `st_mtime`: last modification time of the file
- `st_ctime`: last change time of the i-node status
- `atime` can be used to detect unused files
- `mtime` and `ctime` can be used to archive only those files that have changed



# utime

---

- `sys/types.h`, `utime.h`
- `int utime(const char *pathname, const struct utimbuf *times);`
- Can be used to change `atime` and `mtime`
- `struct utimbuf {`
  - `time_t actime;`
  - `time_t modtime;`
- `}`
- Null pointer = use current time
- Used by `touch`, `tar`, and `cpio`

# How file operations affect times

Function	Referenced file	Parent Directory	Note
chmod, fchmod	c		
chown, fchown	c		
creat	a, m, c	m, c	O_CREAT new file
creat	m, c		O_TRUNC existing file
exec	a		
lchown	c		
link	c	m, c	
mkdir	a, m, c	m, c	
mknod	a, m, c	m, c	
open	a, m, c	m, c	O_CREAT new file
open	m, c		O_TRUNC existing file
pipe	a, m, c		
read	a		
remove	c	m, c	remove file = unlink
remove		m, c	remove directory = rmdir
rename	c	m, c	for both arguments
rmdir		m, c	
truncate, ftruncate	m, c		
unlink	c	m, c	
utime	a, m, c		
write	m, c		



# mkdir and rmdir

---

- `sys/types.h, sys/stat.h`
- `int mkdir(const char *pathname, mode_t mode);`
- Creates an empty directory
- `int rmdir(const char *pathname);`
- Directory must be empty for rmdir to succeed



# Reading directories

---

- Anyone can read directories, only kernel can write them
- `sys/types.h`, `dirent.h`
- `DIR *opendir(const char *pathname);`
- `struct dirent *readdir(DIR *dp);`
- `void rewinddir(DIR *dp);`
- `int closedir(DIR *dp);`
- `struct dirent {`
  - `ino_t d_ino;`
  - `char d_name[NAME_MAX + 1];`
- `}`



# chdir, fchdir, and getcwd

---

- `unistd.h`
- `int chdir(const char *pathname);`
- `int fchdir(int fildes);`
- These change the current working directory
- `char *getcwd(char *buf, size_t size);`
- Returns current working directory



# Special device files

---

- Filesystems identified by major and minor device numbers
- `st_dev` and `st_rdev` fields of `stat` info
- Macros: `major` and `minor`





# sync and fsync

---

- `unistd.h`
- `void sync(void);`
- `int fsync(int fildes);`
- sync flushes file system
  - Normally happens every 30 seconds to five minutes (depending on the file system)
- fsync flushes one file