Operating Systems

CSE 231

Instructor: Sambuddho Chakravarty

(Semester: Winter 2020)

Week 10: Dec 23 – Dec 26

File Concept

- Contiguous logical address space

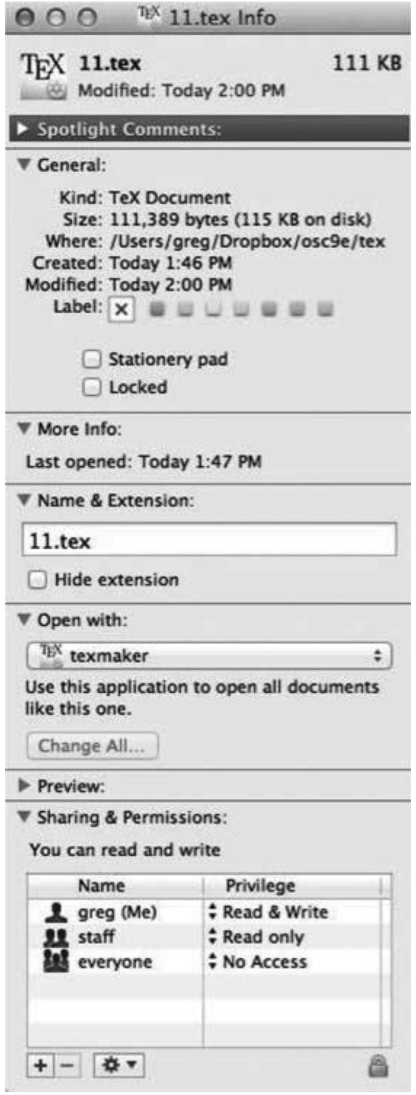
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

- Contents defined by file’s creator
  - Many types
    - Consider **text file, source file, executable file**

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

File info Window on Mac OS X



11.tex 111 KB
Modified: Today 2:00 PM

Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label: ☒

□ Stationery pad
□ Locked

Last opened: Today 1:47 PM

11.tex
□ Hide extension

texmaker
Use this application to open all documents like this one.

You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

File Operations

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry ***F_i*** in memory to directory structure on disk

Open Files

- Several pieces of data are needed to manage open files:
    - **Open-file table**: tracks open files
    - File pointer: pointer to last read/write location, per process that has the file open
    - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
    - Disk location of the file: cache of data access information
    - Access rights: per-process access mode information

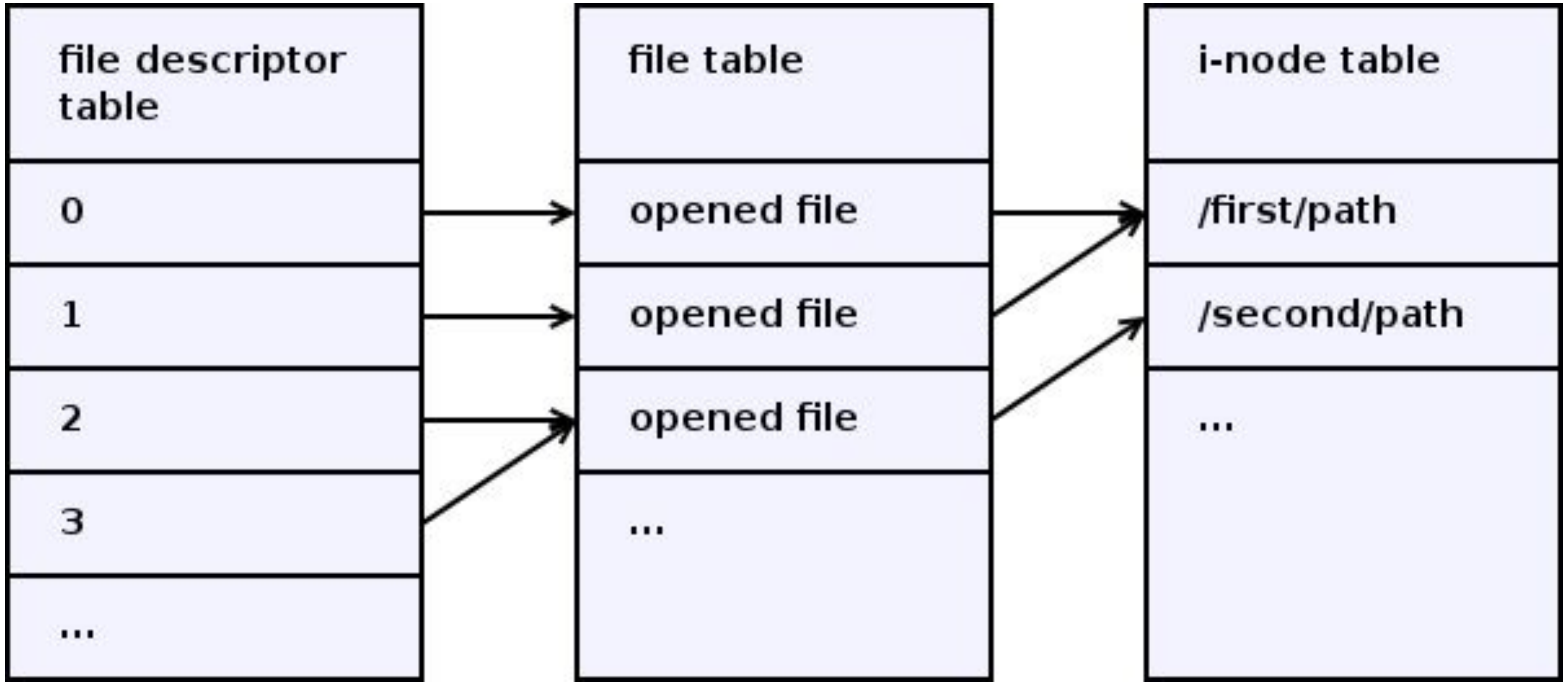
Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

Linux File Locking

- Advisory locking – A process explicitly acquires and releases locks, and are ignored if a process is not aware of locks – flock(), fcntl(), lockf().
- All locks support blocking and non-blocking operations.
- Locks are allowed only on files, but not directories.
- Locks are automatically removed when the process exits or terminates. It’s guaranteed that if a lock is acquired, the process acquiring the lock is still alive.

File descriptors and i-nodes



file descriptor table

0

1

2

3

...

file table

opened file

opened file

opened file

...

i-node table

/first/path

/second/path

...

-A file descriptor is an index in the per-process file descriptor table (points to a file table object Entry)

-A file object represents an opened file. It contains things likes current read/write offset, non-blocking flag and other non-persistent state.

-An i-node represents a filesystem object. It contains things like file meta-information (e.g. owner and permissions) and references to data blocks.

BSD locks (flock)

- Features
  - Wide availability
  - Locks the entire file
  - > 2.6.11, it works with NSF as well
  - Associated with file objects i.e.
    - Duplicated file descriptors, e.g. created using dup2 or fork, refer to the same lock.
    - Distinct file descriptors, e.g. created using two open calls (even for the same file), refer to different locks.

BSD locks(flocks)

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

// acquire shared lock
if (flock(fd, LOCK_SH) == -1) {
    exit(1);
}

// non-atomically upgrade to exclusive lock
// do it in non-blocking mode, i.e. fail if can't upgrade immediately
if (flock(fd, LOCK_EX | LOCK_NB) == -1) {
    exit(1);
}

// release lock
// lock is also released automatically when close() is called or process exits
if (flock(fd, LOCK_UN) == -1) {
    exit(1);
}
```

POSIX locks (lockf)

Features:

- Can be applied to a byte range (optionally automatically expanding when data is appended in future).
- Associated with an [i-node, pid] pair instead of a file object.
- Supports only exclusive locks.

POSIX locks(lockf)

```
#include <unistd.h>

// set current position to byte 10
if (lseek(fd, 10, SEEK_SET) == -1) {
    exit(1);
}

// acquire exclusive lock for bytes in range [10; 15)
// F_LOCK specifies blocking mode
if (lockf(fd, F_LOCK, 5) == -1) {
    exit(1);
}

// release lock for bytes in range [10; 15)
if (lockf(fd, F_ULOCK, 5) == -1) {
    exit(1);
}
```

POSIX fcntl()

Features:

- POSIX compliant.
- Can be applied to a byte range.
- Associated with an [i-node, pid] pair instead of a file object.

All file descriptors opened by the same process for the same file refer to the same lock (even distinct file descriptors, e.g. created using two open() calls)

POSIX fcntl()

Therefore, all process’ threads always share the same lock for the same file. In particular:

- The lock acquired through some file descriptor by some thread may be released through another file descriptor by another thread;

- When any thread calls close() on any descriptor referring to given file, the lock is released for the whole process, even if there are other opened descriptors referring this file.

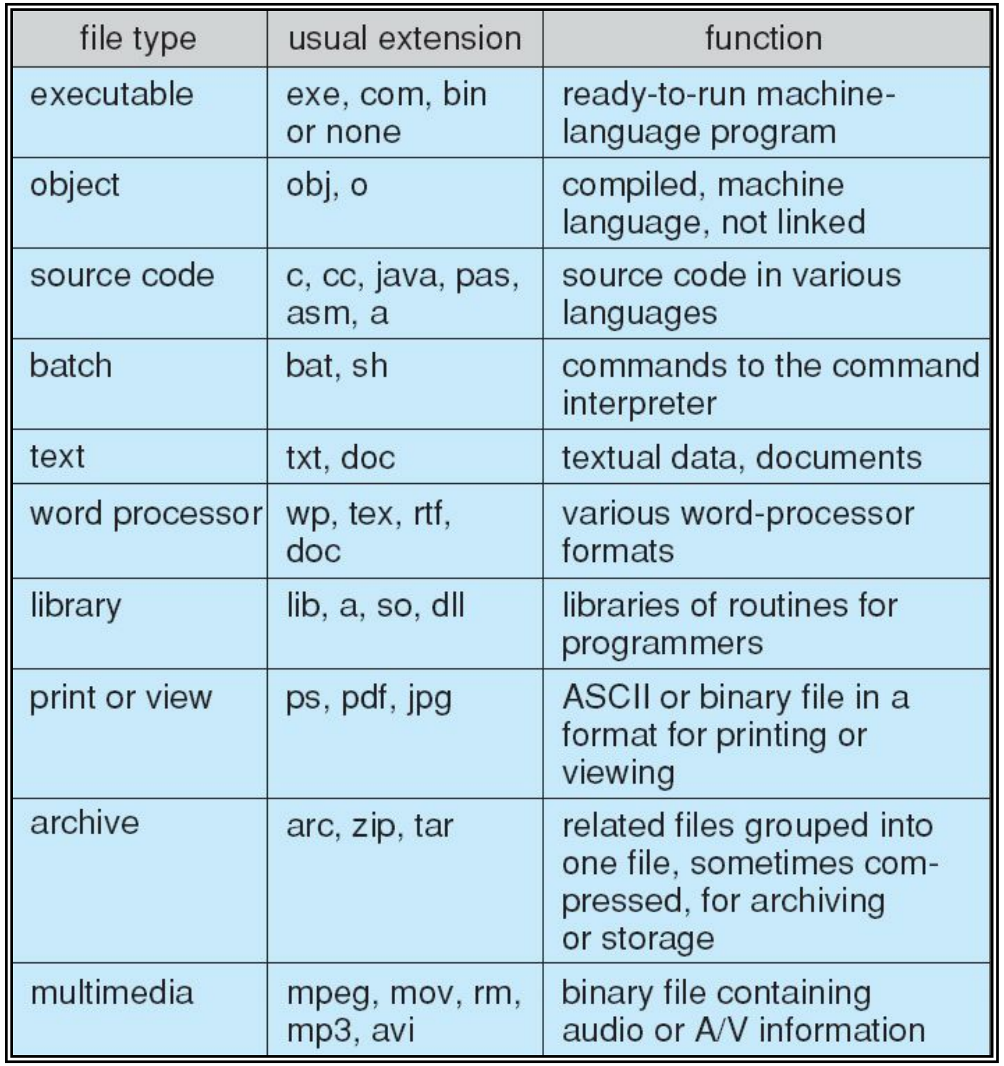
Inconvenient to use POSIX record locks -- when you want to synchronize threads as well as processes because all threads always share the same lock. Also problems when designing shared libs.

POSIX fcntl() example

```
struct flock fl;
memset(&fl, 0, sizeof(fl));
// lock in shared mode
fl.l_type = F_RDLCK;
// lock entire file
fl.l_whence = SEEK_SET; // offset base is start of the file
fl.l_start = 0;         // starting offset is zero
fl.l_len = 0;           // len is zero, which is a special value representing end
                        // of file (no matter how large the file grows in future)
// F_SETLKW specifies blocking mode
if (fcntl(fd, F_SETLKW, &fl) == -1) {
    exit(1);
}
```

```
// atomically upgrade shared lock to exclusive lock, but only
// for bytes in range [10; 15)
// after this call, the process will hold three lock regions:
//  [0; 10)        - shared lock
//  [10; 15)       - exclusive lock
//  [15; SEEK_END) - shared lock
fl.l_type = F_WRLCK;
fl.l_start = 10;
fl.l_len = 5;
// F_SETLKW specifies non-blocking mode
if (fcntl(fd, F_SETLK, &fl) == -1) {
    exit(1);}
// release lock for bytes in range [10; 15)
fl.l_type = F_UNLCK;
if (fcntl(fd, F_SETLK, &fl) == -1) {
    exit(1);
}
// close file and release locks for all regions
// remember that locks are released when process calls close()
// on any descriptor for a lock file
close(fd);
```

File Types – Name, Extension (However they have little meaning in Linux/Unix Systems)



file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

- **Sequential Access**

```
read next
write next
reset
```

no read after last write

(rewrite)

- **Direct Access –** file is fixed length logical records

```
read n
write n
position to n
   read next
   write next
rewrite n
```

n = relative block number

Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

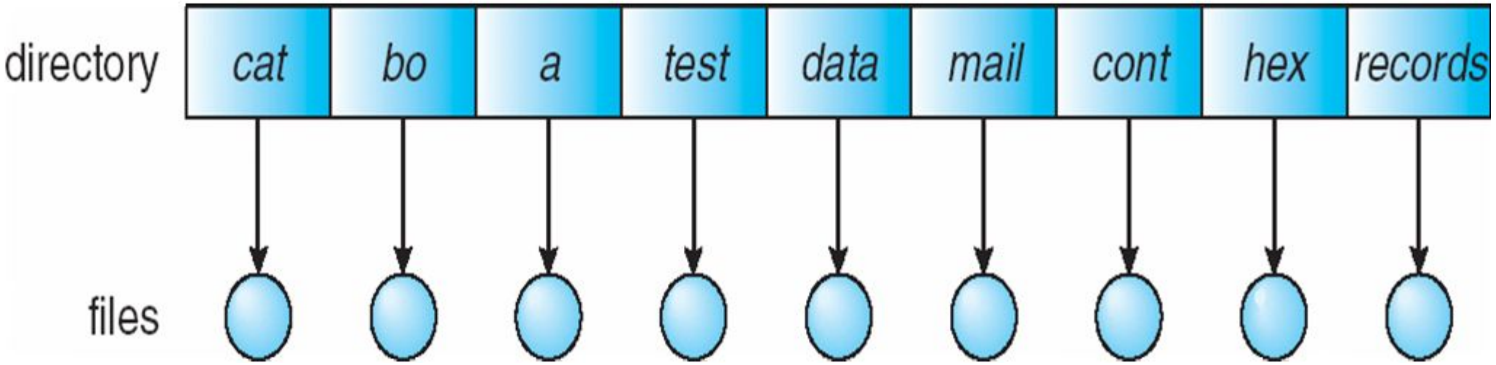
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (Logically) to Obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A single directory for all users

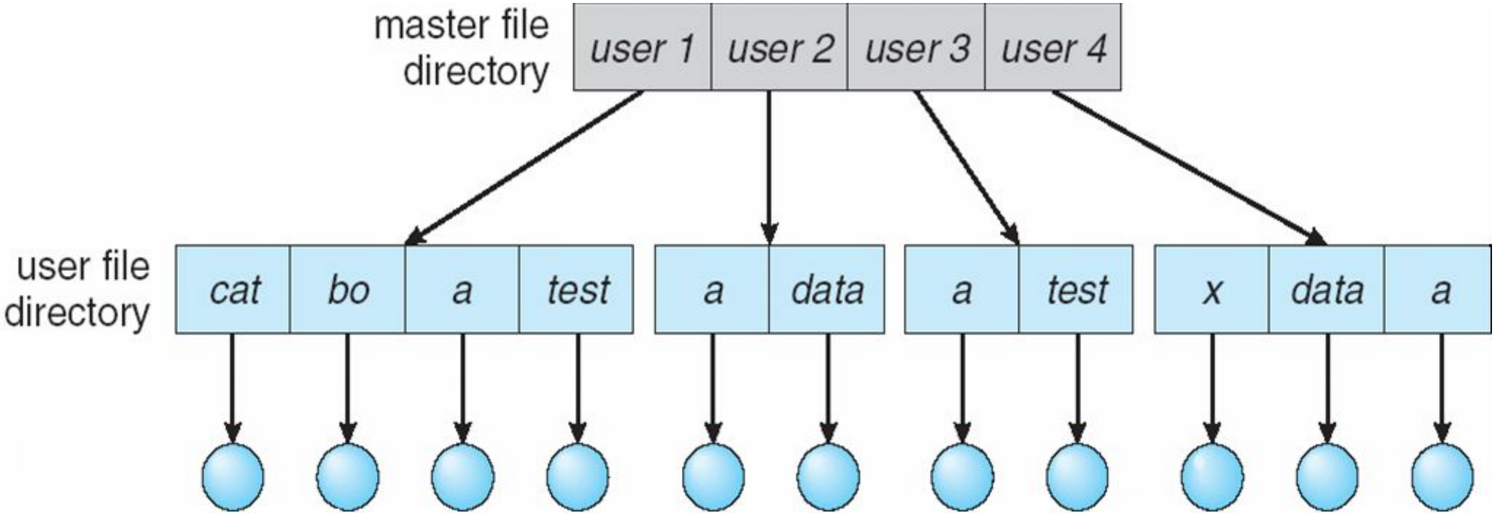


Naming problem

Grouping problem

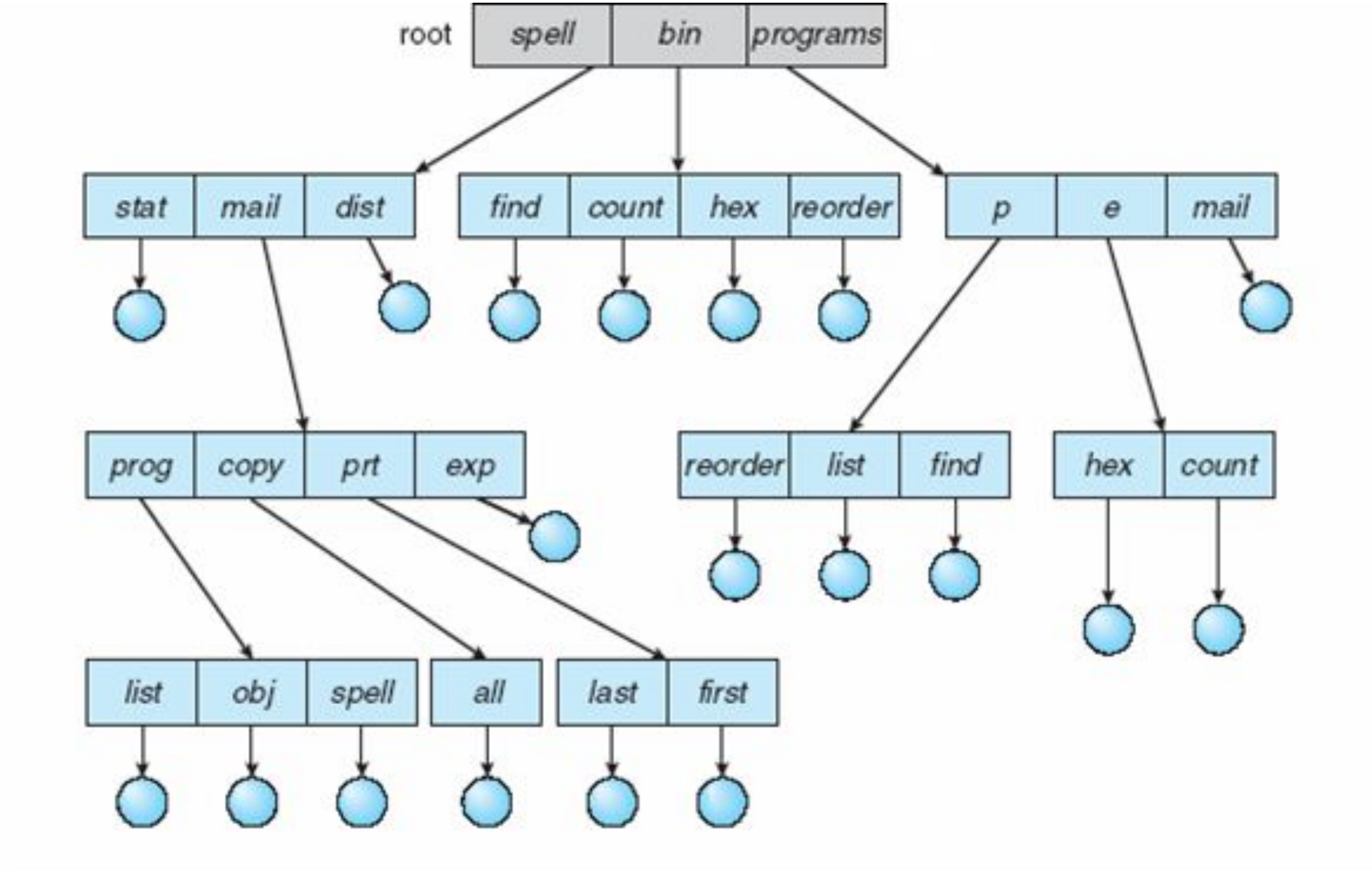
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **`cd /spell/mail/prog`**
  - **`type list`**

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

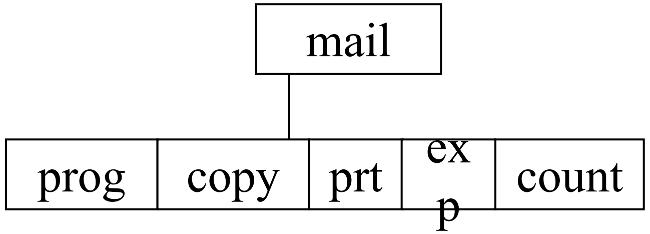
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory **`/mail`**

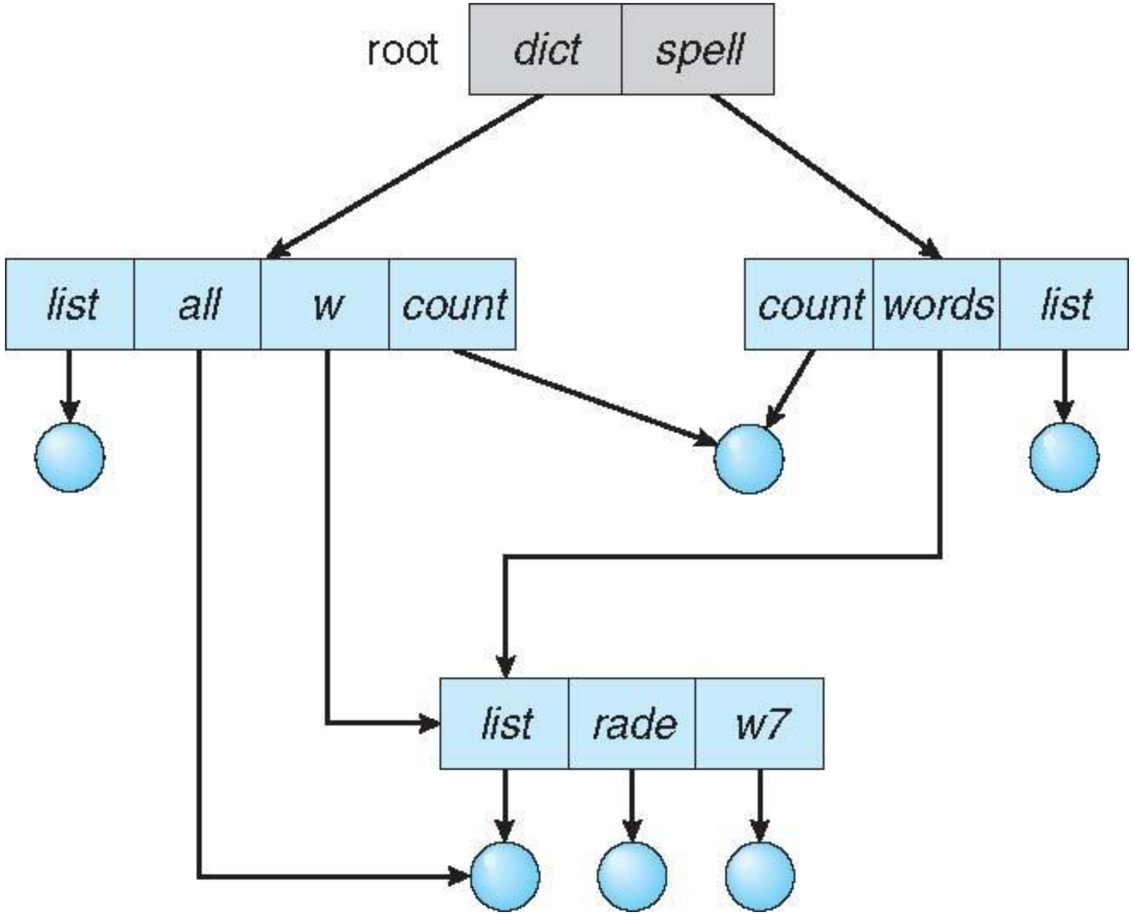
```
mkdir count
```



Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

- Have shared subdirectories and files



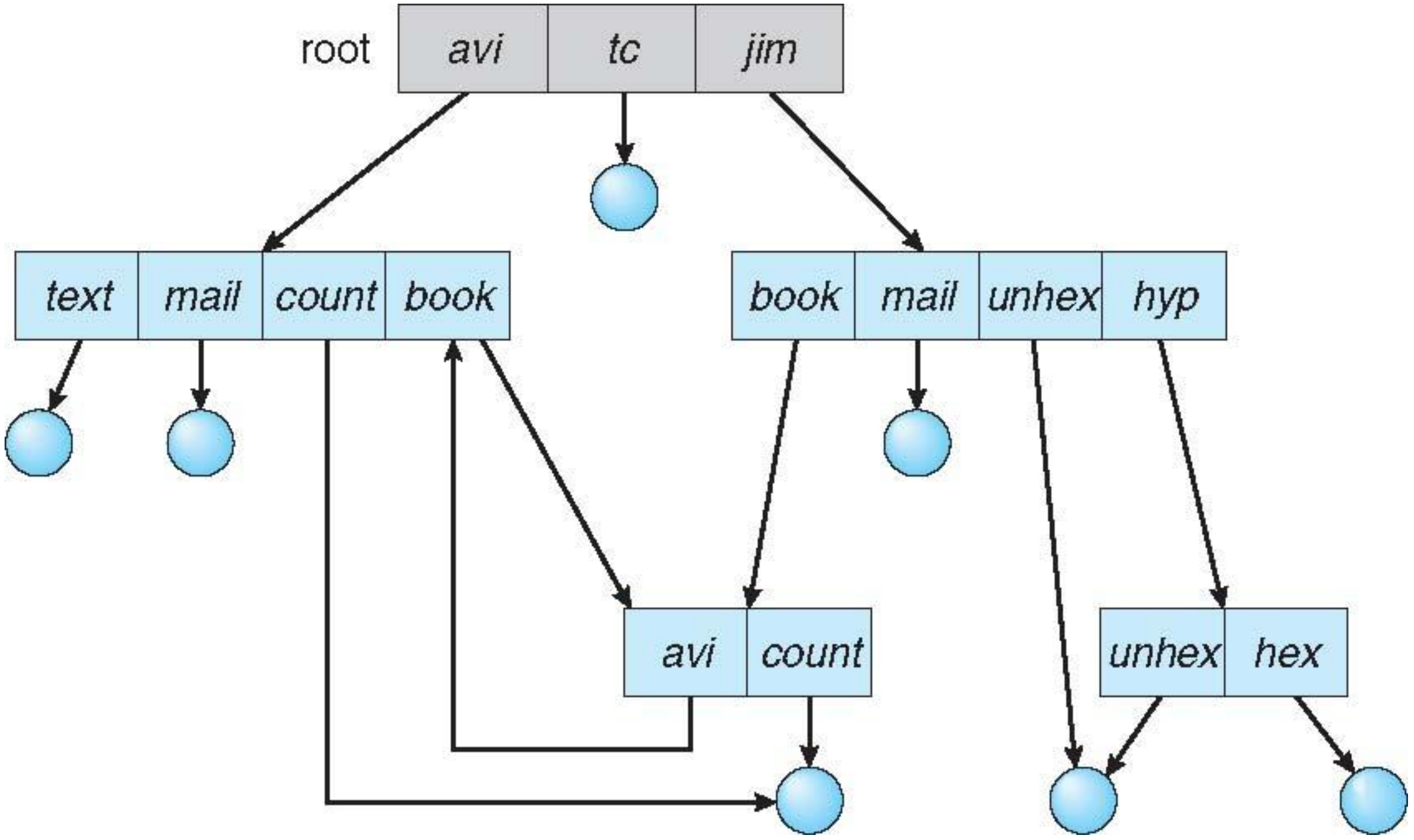
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If ***dict*** deletes ***list*** ⇒ dangling pointer

  Solutions:

  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

General Graph Directory



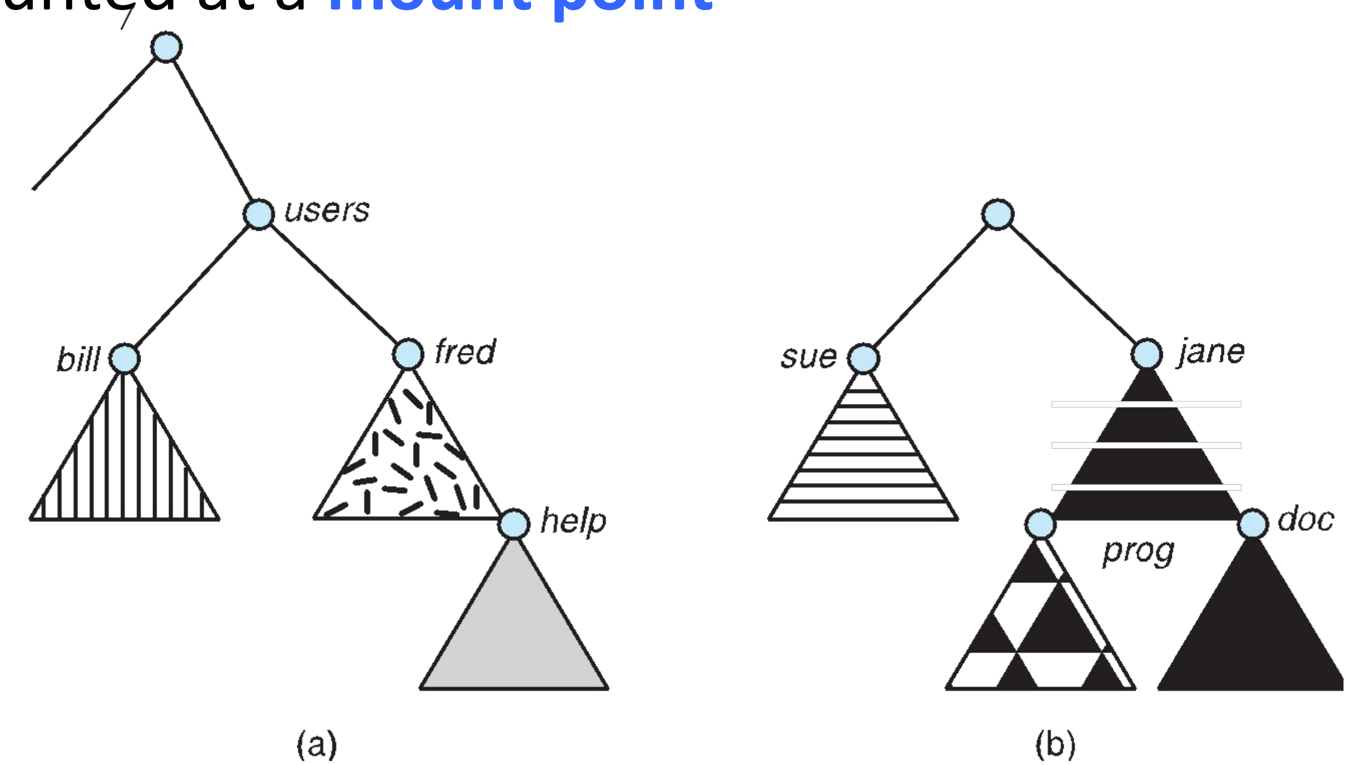
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

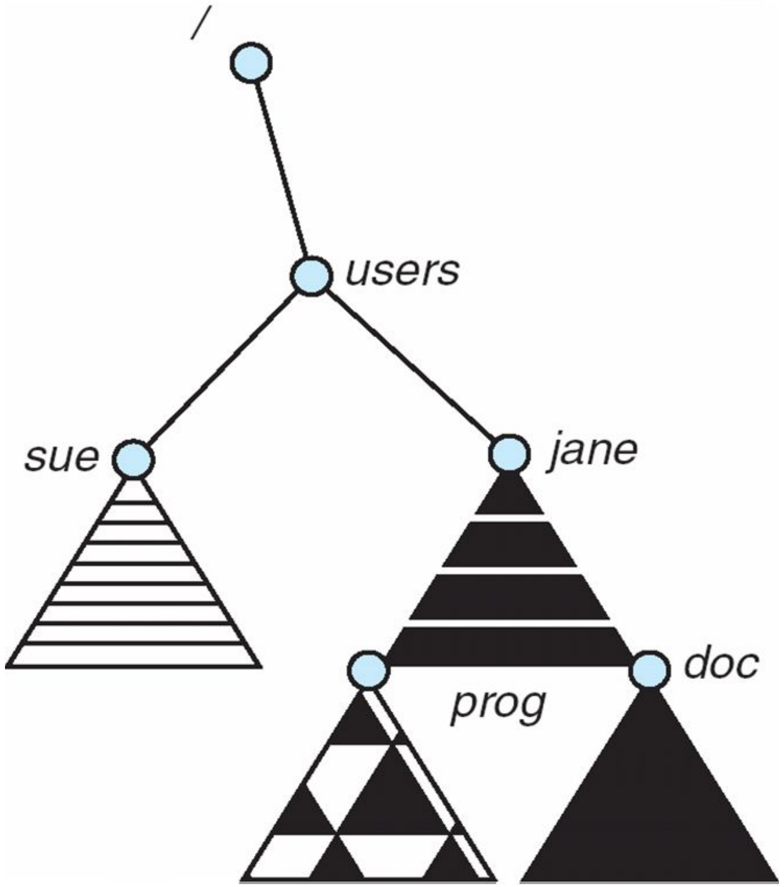
File System Mounting

- A file system must be **mounted** before it can be accessed

- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**



Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user **Group IDs** allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously
    - Similar to Ch 5 process synchronization algorithms
        - Tend to be less complex due to disk I/O and network latency (for remote file systems
    - Andrew File System (AFS) implemented complex remote file sharing semantics
    - Unix file system (UFS) implements:
        - Writes to an open file visible immediately to other users of the same open file
        - Sharing file pointer to allow multiple users to read and write concurrently
    - AFS has session semantics
        - Writes only visible to sessions starting after the file is closed

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

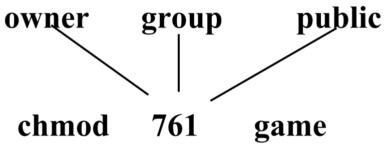
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

RWX
a) **owner access** 7 ⇒ 1 1 1
RWX
b) **group access** 6 ⇒ 1 1 0
RWX
c) **public access** 1 ⇒ 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



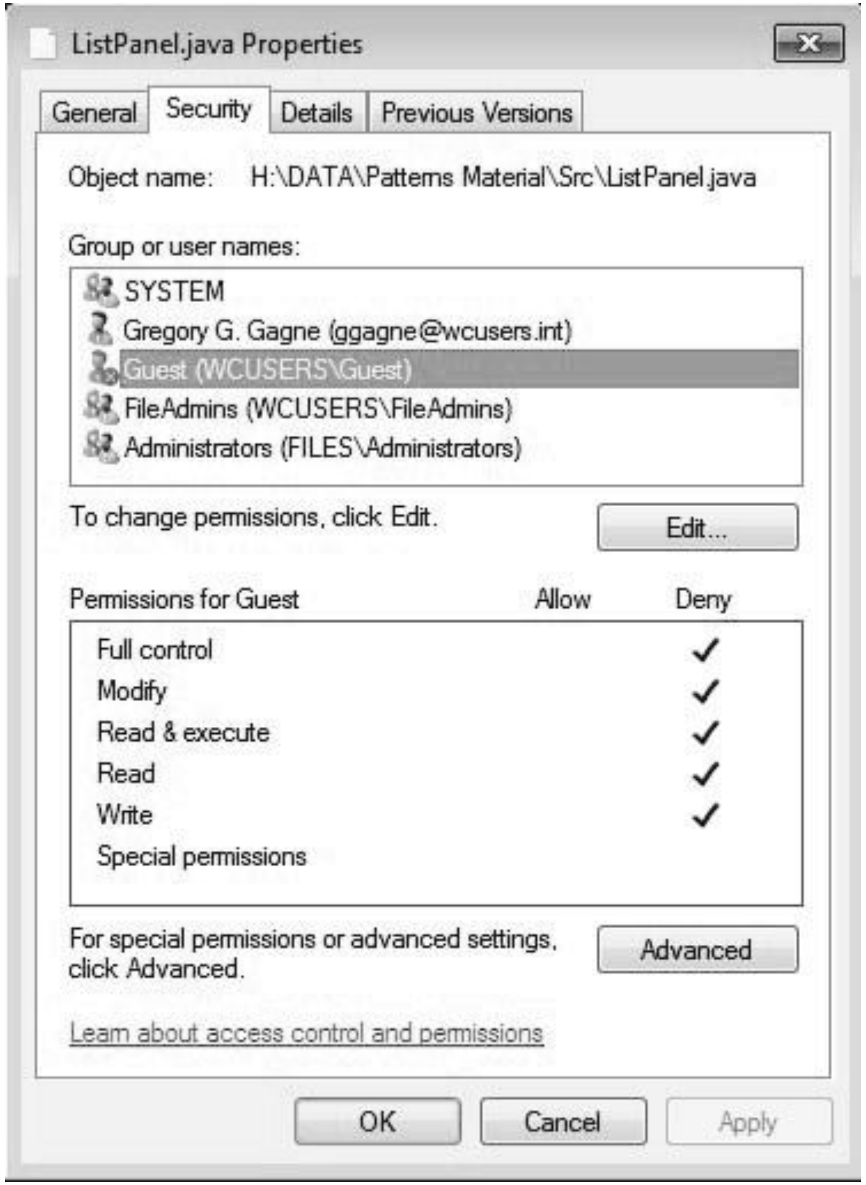
owner group public

chmod 761 game

Attach a group to a file

```
chgrp G game
```

Windows 7 Access-Control List Management



Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:

- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

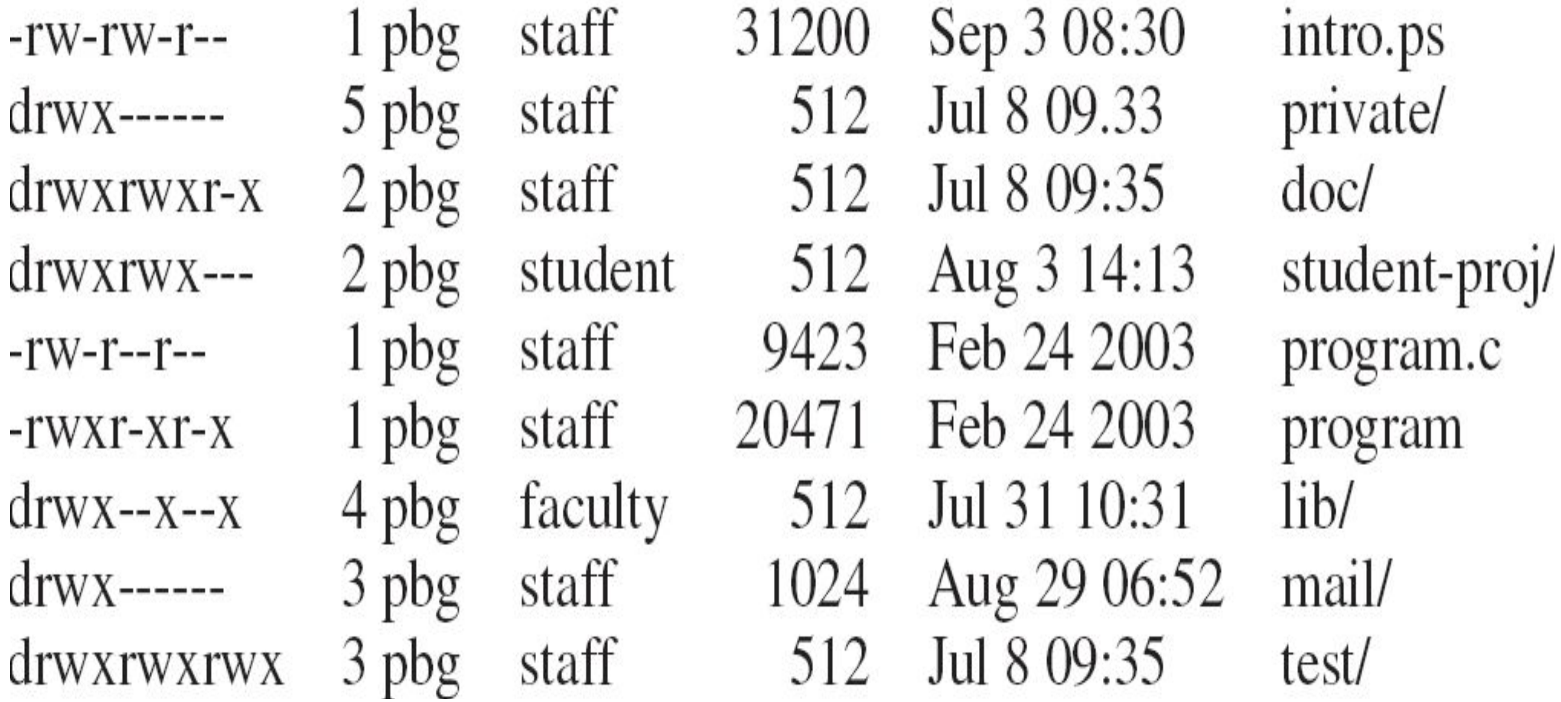
To change permissions, click Edit.

Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

A Sample UNIX Directory Listing



```
-rw-rw-r--    1 pbg   staff     31200   Sep 3 08:30     intro.ps
drwx------    5 pbg   staff       512   Jul 8 09.33     private/
drwxrwxr-x    2 pbg   staff       512   Jul 8 09:35     doc/
drwxrwx---    2 pbg   student     512   Aug 3 14:13     student-proj/
-rw-r--r--    1 pbg   staff      9423   Feb 24 2003     program.c
-rwxr-xr-x    1 pbg   staff     20471   Feb 24 2003     program
drwx--x--x    4 pbg   faculty     512   Jul 31 10:31    lib/
drwx------    3 pbg   staff      1024   Aug 29 06:52    mail/
drwxrwxrwx    3 pbg   staff       512   Jul 8 09:35     test/
```