CS307&CS356: Operating Systems

Dept. of Computer Science & Engineering

Chentao Wu

wuct@cs.sjtu.edu.cn

Download lectures

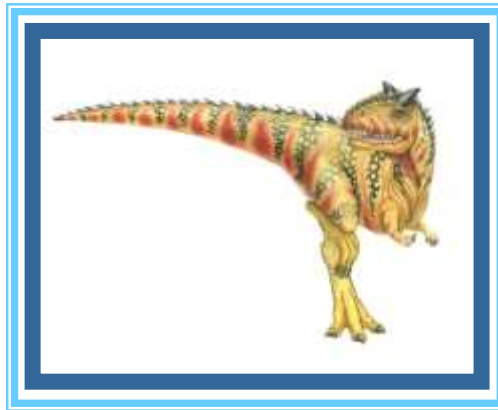- ftp://public.sjtu.edu.cn
- User: wuct
- Password: wuct123456

- http://www.cs.sjtu.edu.cn/~wuct/cse/

Chapter 13:
File-System Interface

Chapter 13: File-System Interface

- File Concept

- Access Methods

- Disk and Directory Structure

- File-System Mounting

- File Sharing

- Protection

Objectives

- To explain the function of file systems

- To describe the interfaces to file systems

- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures

- To explore file-system protection

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 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

File Locking Example – Java API

```java
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
                RandomAccessFile raf = new
    RandomAccessFile("file.txt", "rw");
                // get the channel for the file
                FileChannel ch = raf.getChannel();
                // this locks the first half of the file - exclusive
                exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
                /** Now modify the data . . . */
                // release the lock
                exclusiveLock.release();
```

```
                // this locks the second half of the file - shared
                sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                                     SHARED);
                /** Now read the data . . . */
                // release the lock
                sharedLock.release();
        } catch (java.io.IOException ioe) {
                System.err.println(ioe);
        }finally {
                if (exclusiveLock != null)
                exclusiveLock.release();
                if (sharedLock != null)
                sharedLock.release();
        }
    }
}
```

File Types – Name, Extension

| 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 com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- **None - sequence of words, bytes**

- **Simple record structure**
  - Lines
  - Fixed length
  - Variable length

- **Complex Structures**
  - Formatted document
  - Relocatable load file

- **Can simulate last two with first method by inserting appropriate control characters**

- **Who decides:**
  - Operating system
  - Program

Sequential-access File
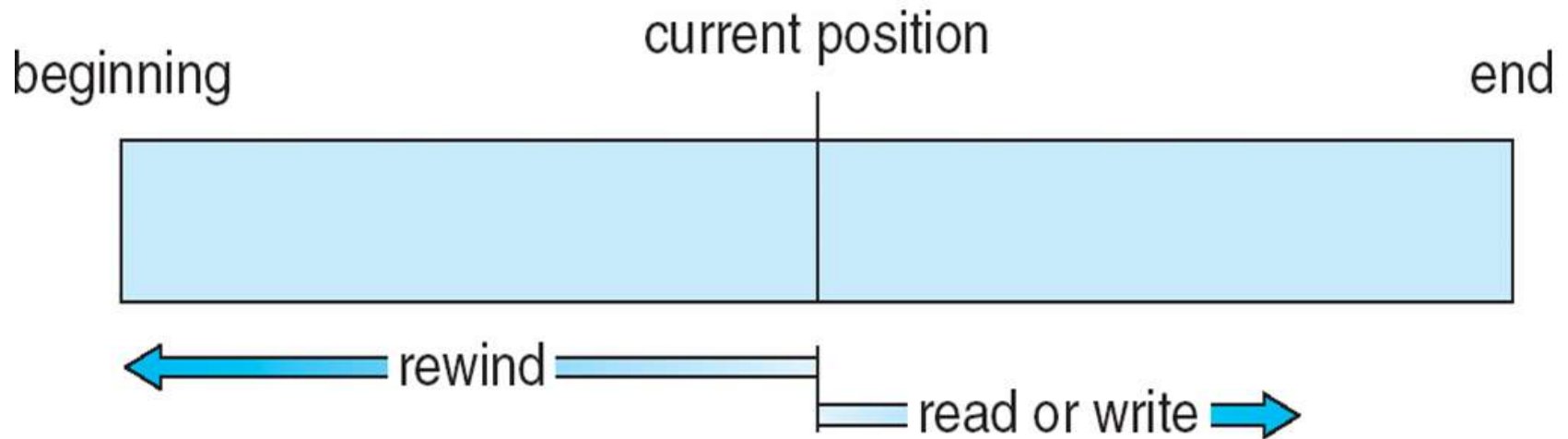
Access Methods

- **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

- Relative block numbers allow OS to decide where file should be placed

  - See allocation problem in Ch 12

sequential access	implementation for direct access
reset	$cp = 0$;
read next	read cp; $cp = cp + 1$;
write next	write cp; $cp = cp + 1$;

Other Access Methods

- Can be built on top of base methods

- General involve creation of an index for the file

- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)

- If too large, index (in memory) of the index (on disk)

- IBM indexed sequential-access method (ISAM)

  - Small master index, points to disk blocks of secondary index

  - File kept sorted on a defined key

  - All done by the OS

- VMS operating system provides index and relative files as another example (see next slide)
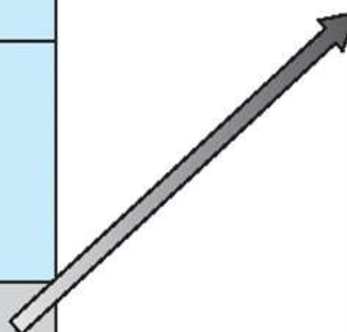
Example of Index and Relative Files

Directory Structure

■ A collection of nodes containing information about all files

Directory

Files

F 1   F 2   F 3   F 4   F n
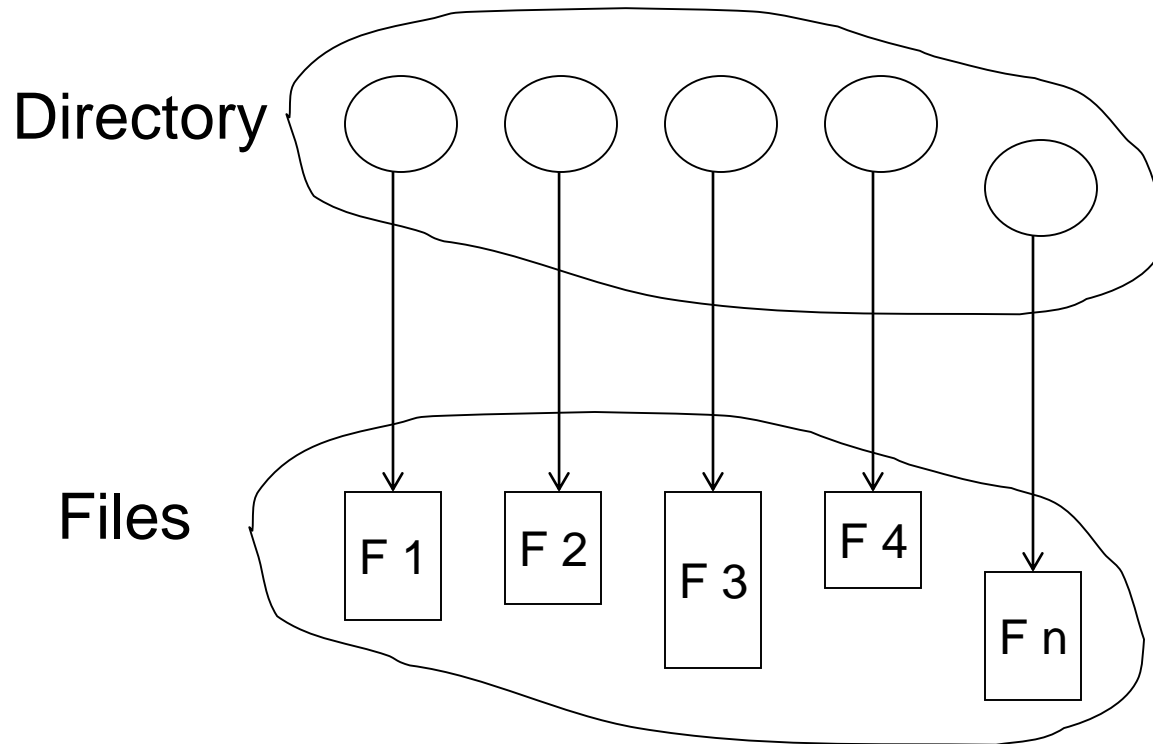
Both the directory structure and the files reside on disk

Disk Structure

- Disk can be subdivided into **partitions**

- Disks or partitions can be **RAID** protected against failure

- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system

- Partitions also known as minidisks, slices

- Entity containing file system known as a **volume**

- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**

- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization
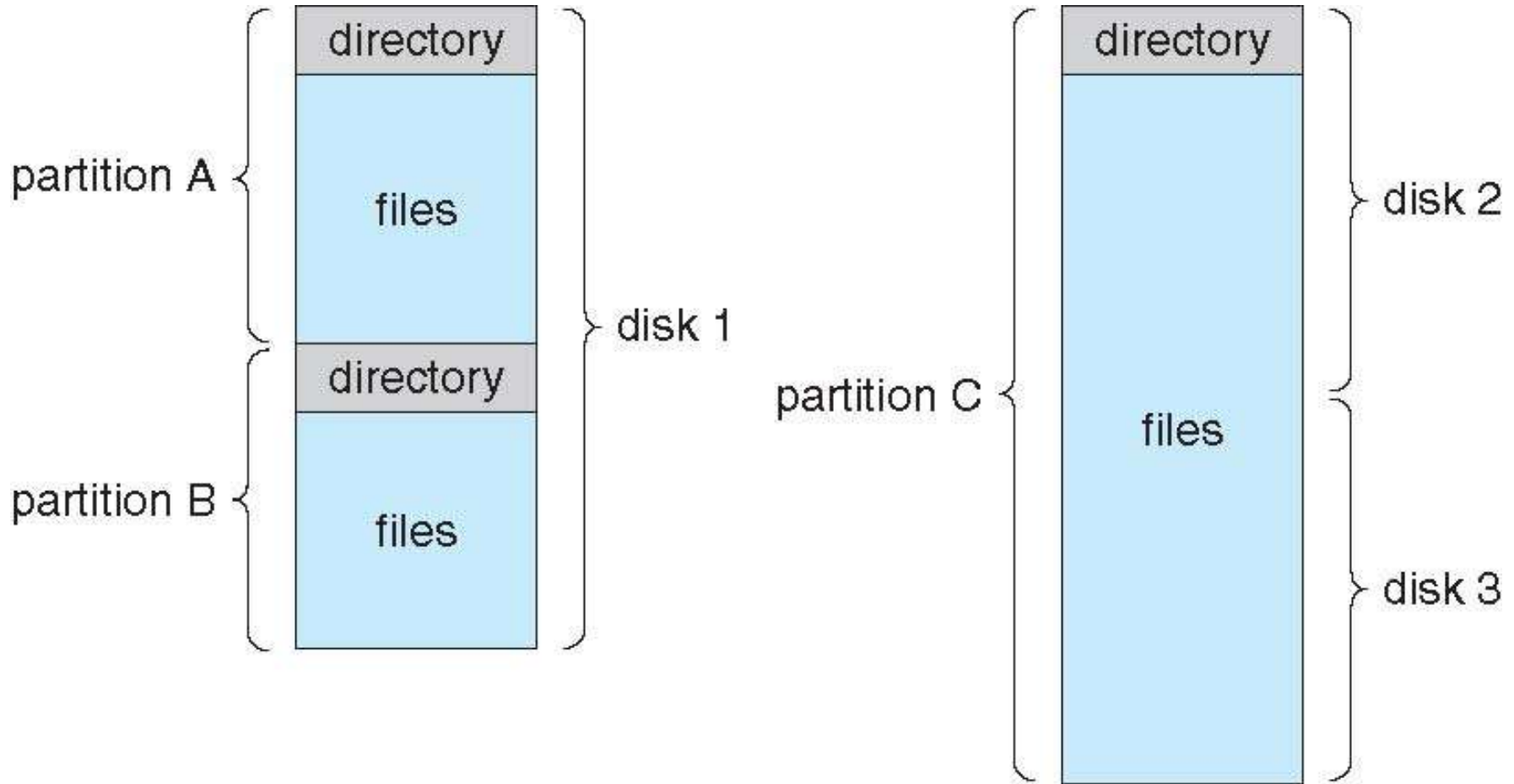
Types of File Systems

- 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

Directory Organization

The directory is organized 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

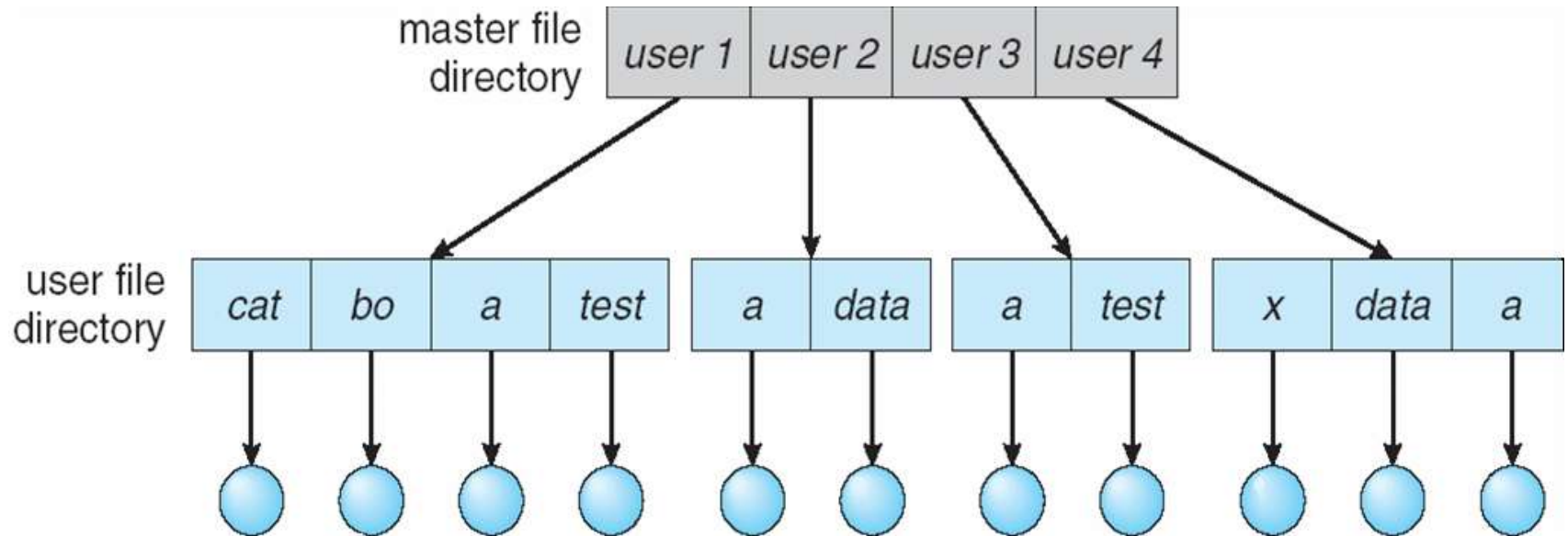| directory	cat	bo	a	test	data	mail	cont	hex	records

files ○ ○ ○ ○ ○ ○ ○ ○ ○
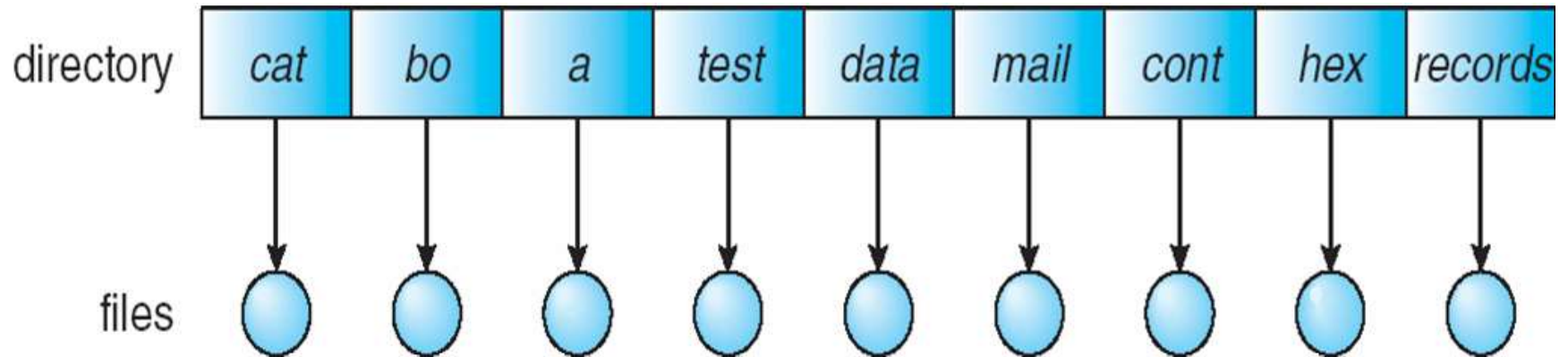
- Naming problem
- Grouping problem

Two-Level Directory
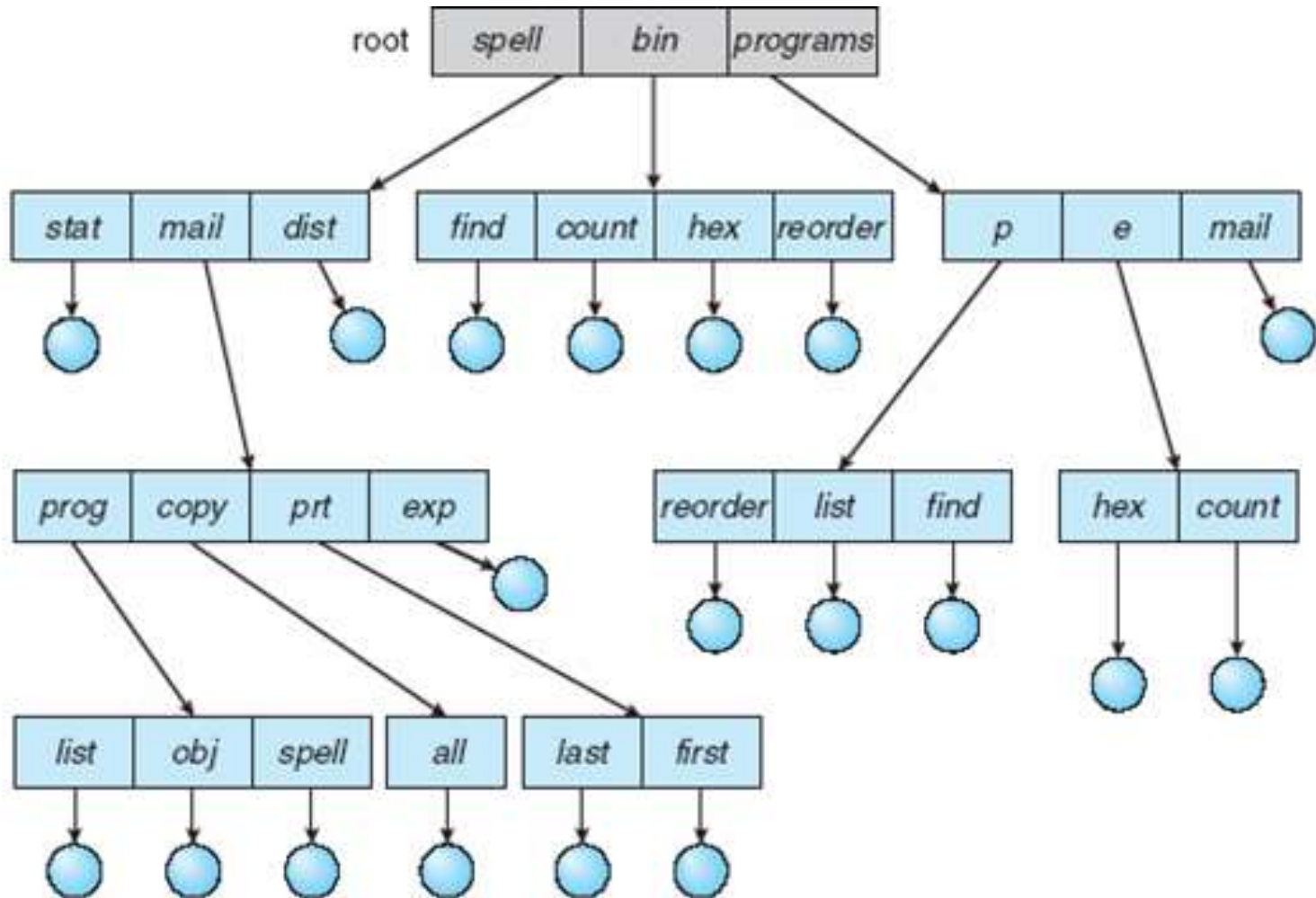
- Separate directory for each user

- 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
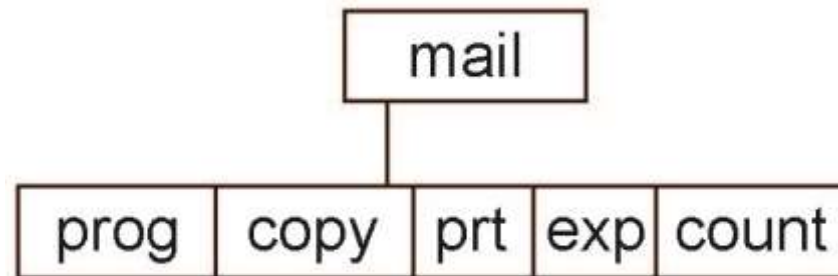
- Delete a file

    rm <file-name>

- Creating a new subdirectory is done in current directory

    mkdir <dir-name>

  Example: if in current directory **/mail**
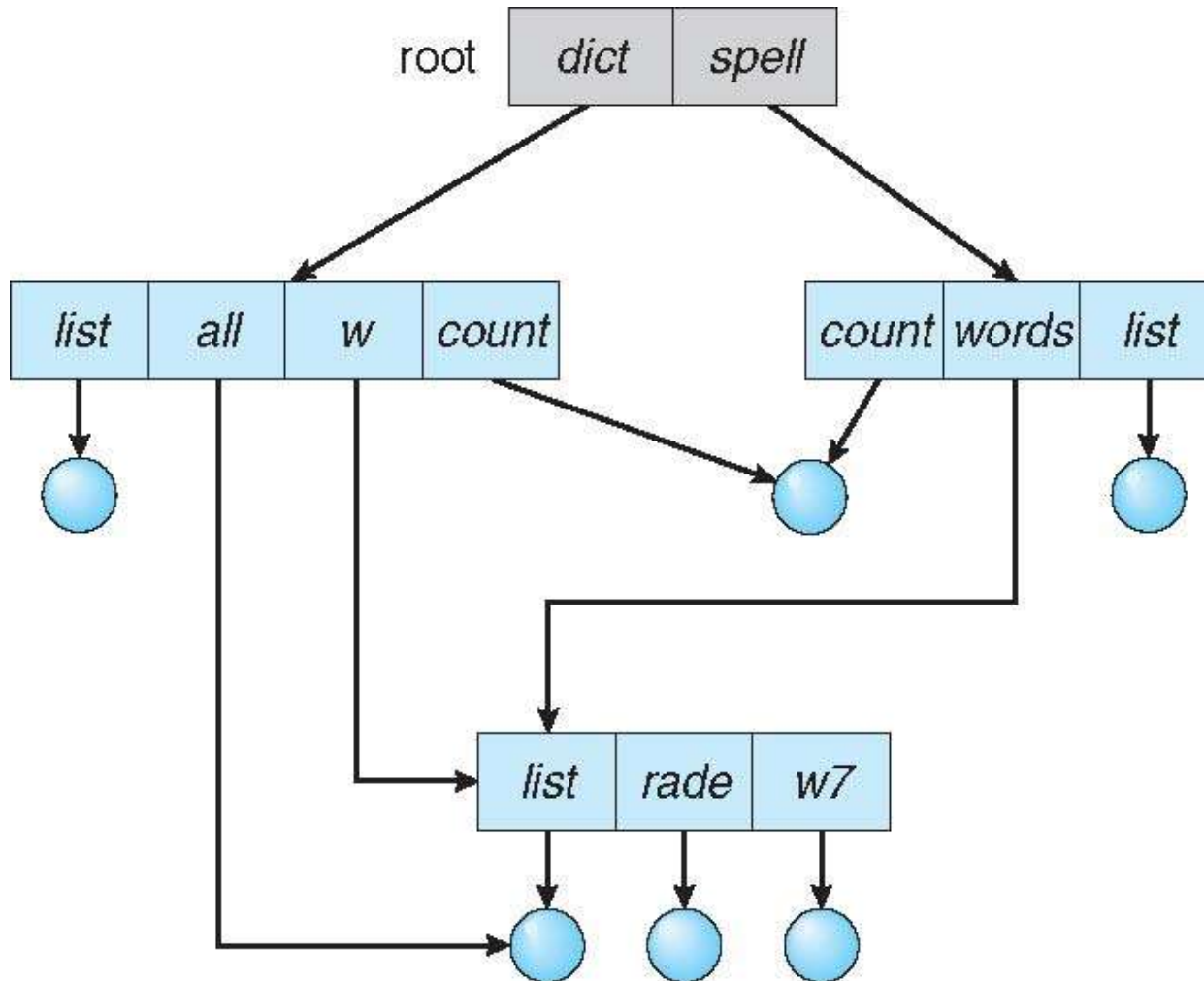
        mkdir count

Deleting "mail" $\Rightarrow$ 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* $\Rightarrow$ dangling pointer
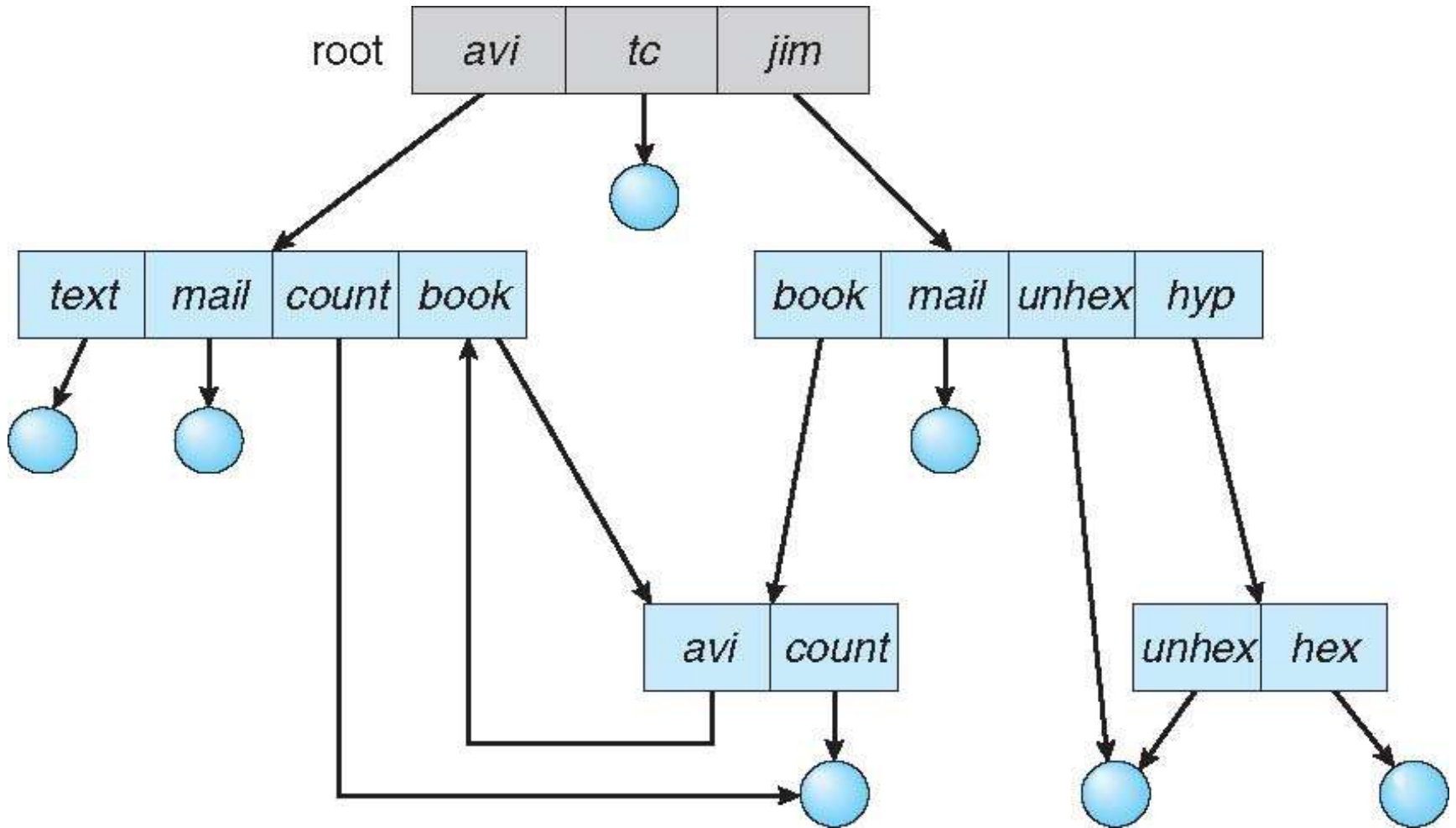
  Solutions:

  - 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
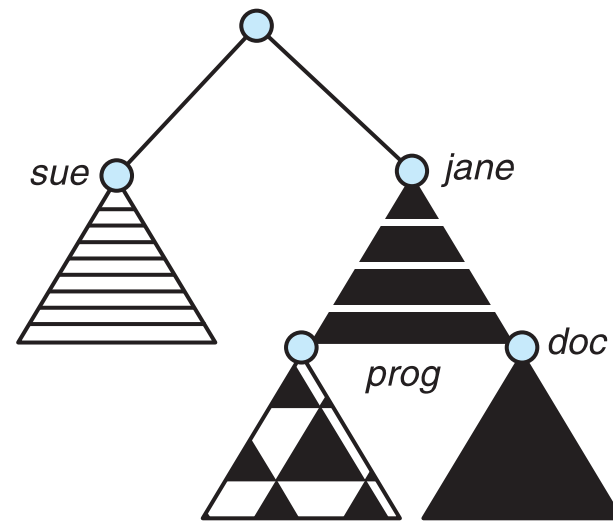
File System Mounting

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

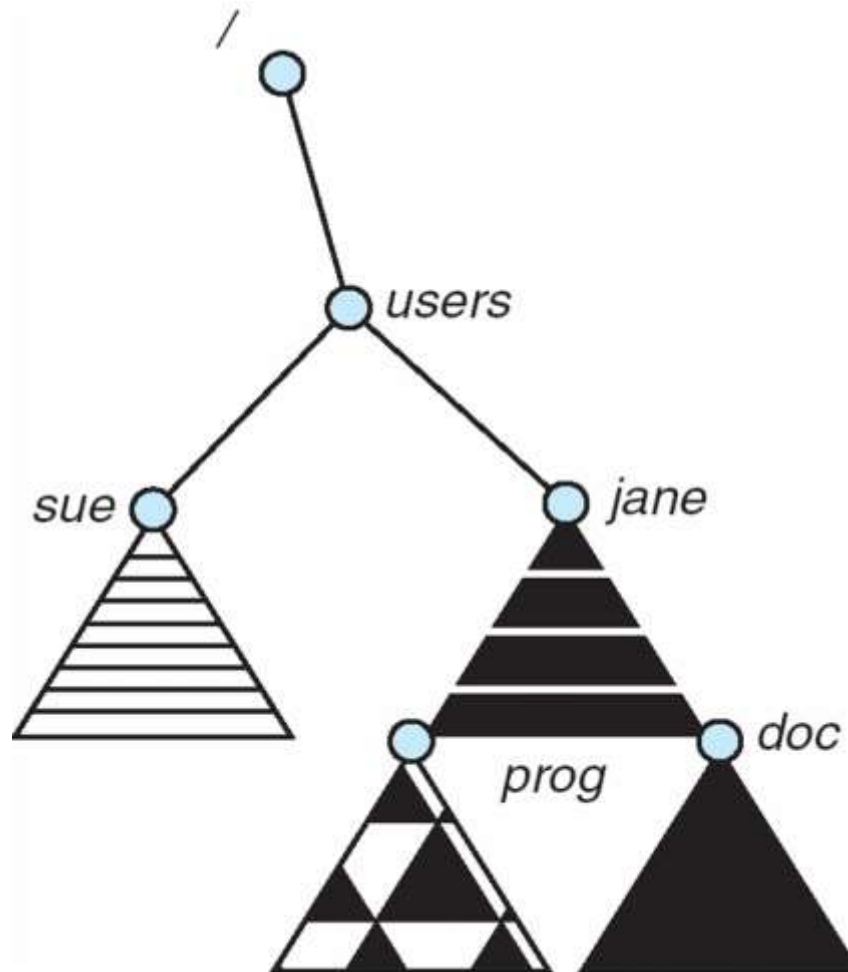- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**
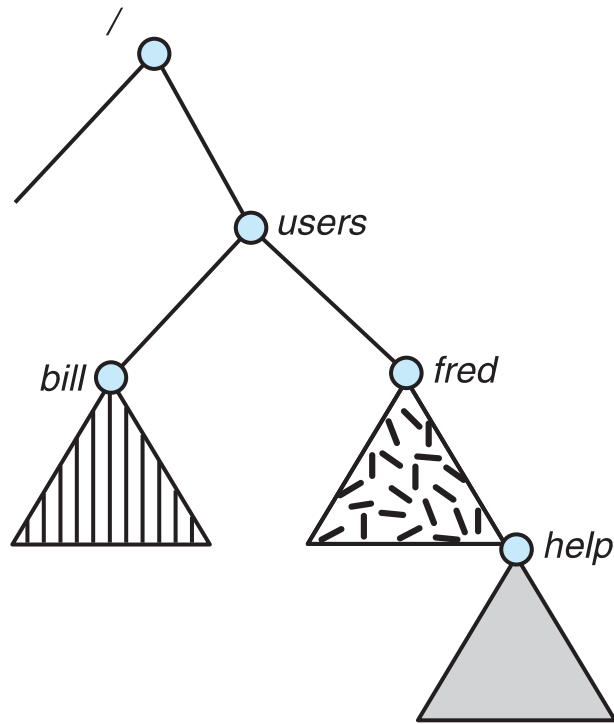
(a)   (b)

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**
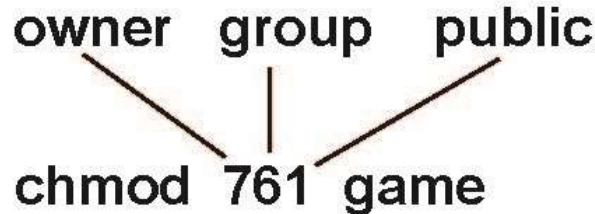 - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

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

			RWX
a) **owner access**	7	$\Rightarrow$	1 1 1
b) **group access**	6	$\Rightarrow$	1 1 0
c) **public access**	1	$\Rightarrow$	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

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/

Homework

- Exercises at the end of Chapter 13 (OS book)
  - 13.7

End of Chapter 13