Chapter 10.A: File-System Interface

Chapter 10.A: Outline

- 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*

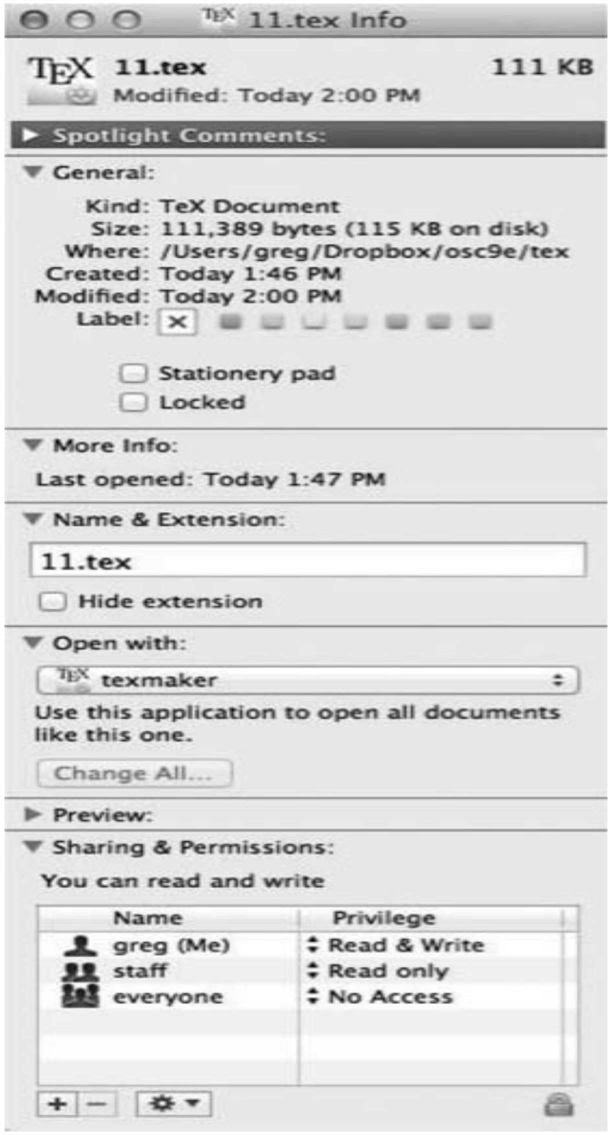
File Concept

- Contiguous *logical address space*
- Types:
  - Data
    - complex
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - 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

A window of file info on Mac OS X



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

▶ Spotlight Comments:

▼ General:

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

▼ More Info:

Last opened: Today 1:47 PM

▼ Name & Extension:

11.tex

☐ Hide extension

▼ Open with:

texmaker

Use this application to open all documents like this one.

▶ Preview:

▼ Sharing & Permissions:

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 (or 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

```
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();
```

File Locking Example – Java API (Cont.)

```
            // 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();
       }
   }
}
```

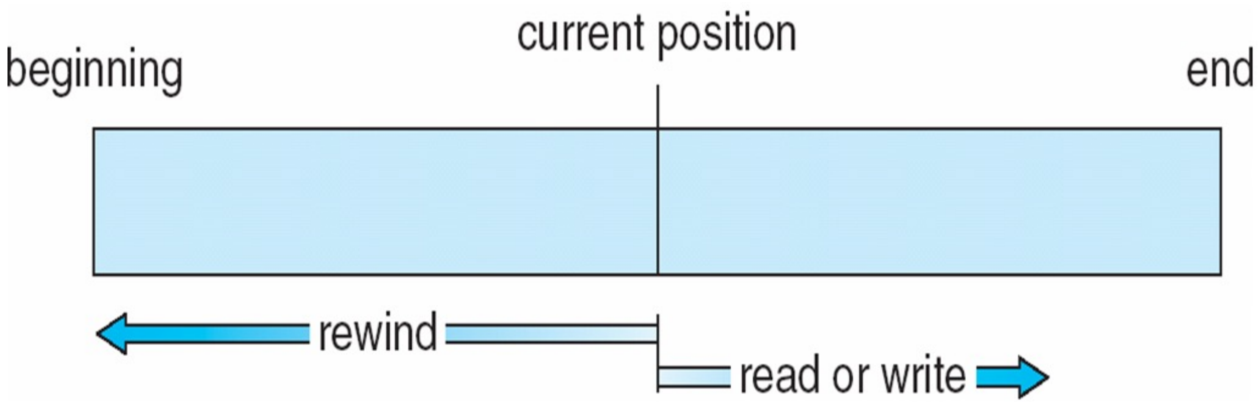
File Types – Name, Extension

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 compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- *None* - sequence of words or bytes
- *Simple record structures*
  - Lines
  - Fixed length
  - Variable length
- *Complex structures*
  - Formatted document
  - Relocatable load file (i.e., executable 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 Chapter 12

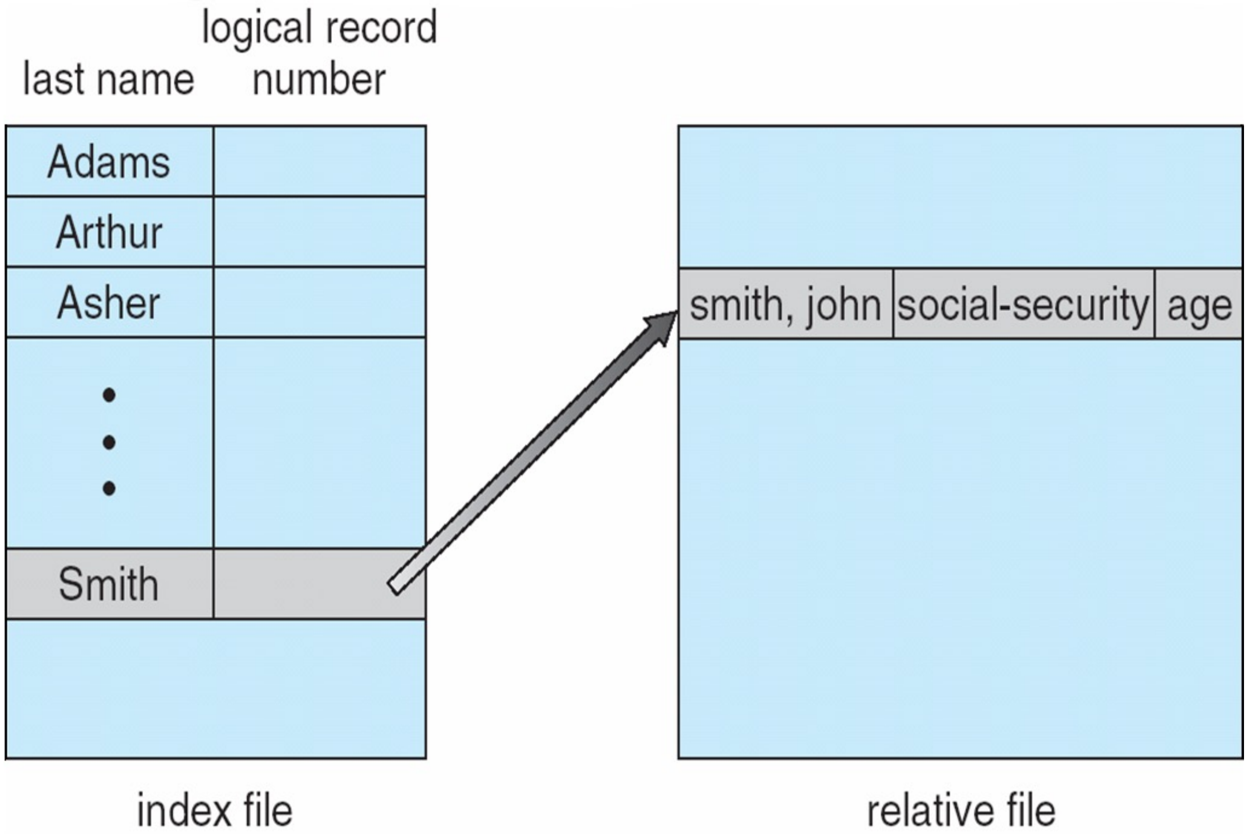
Simulation of Sequential-Access on Direct-Access File

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

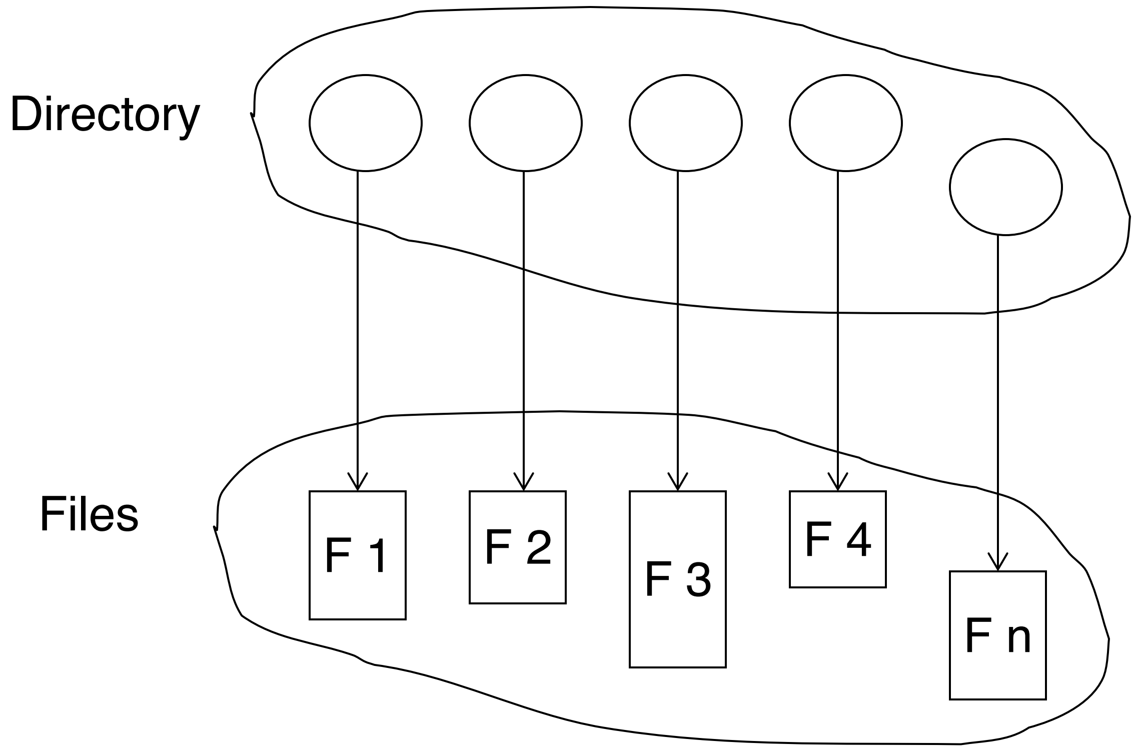
- Can be built on top of base methods
- General involve creation of an *index* for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- E.g., **IBM** *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

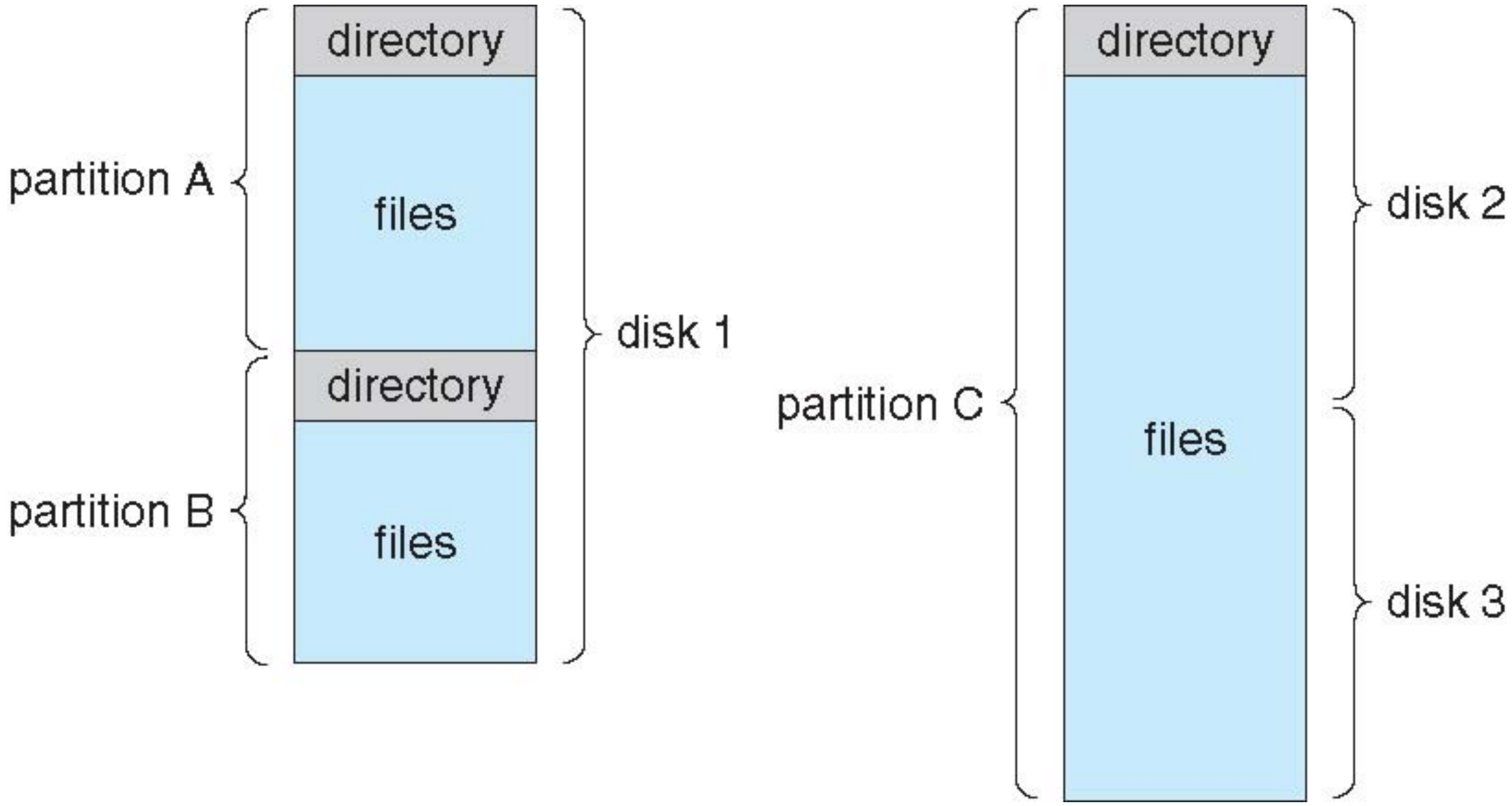


Both the directory structure and the files reside on disk

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 many file systems, some general- and some special- purpose
- E.g., **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

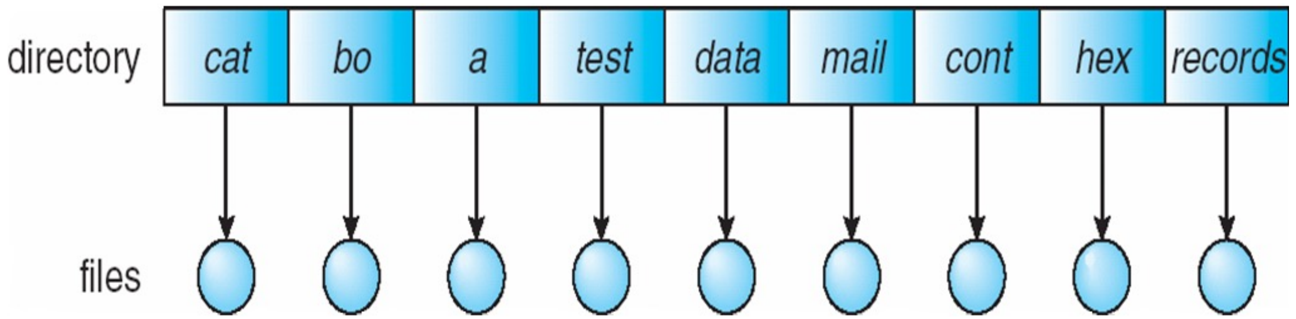
- *Search* for a file
- *Create* a file
- *Delete* a file
- *List* a directory
- *Rename* a file
- *Traverse* the file system

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

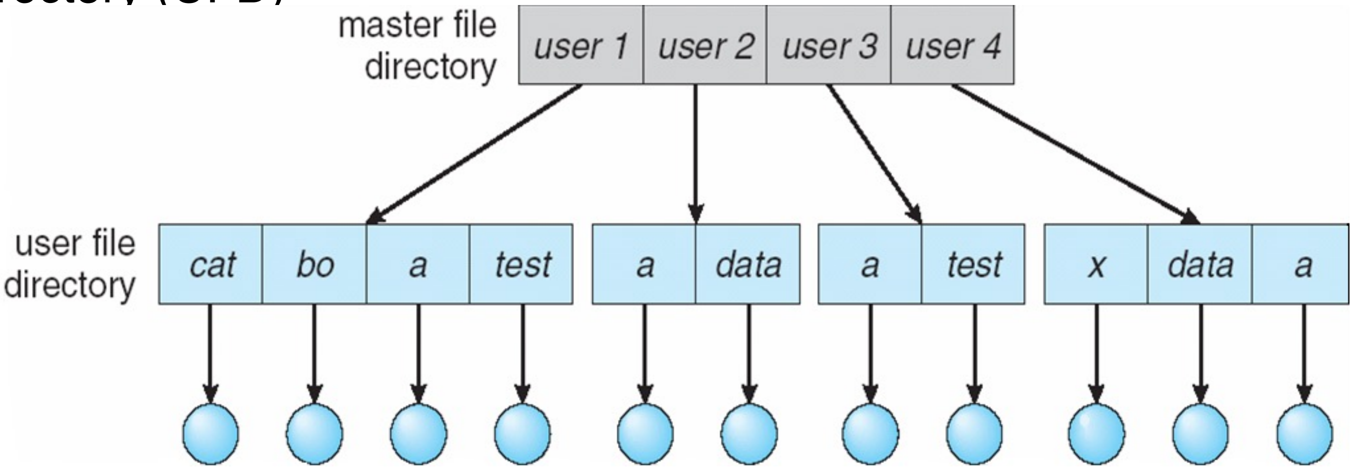
- *A single directory for all users*



- Naming problem
- Grouping problem

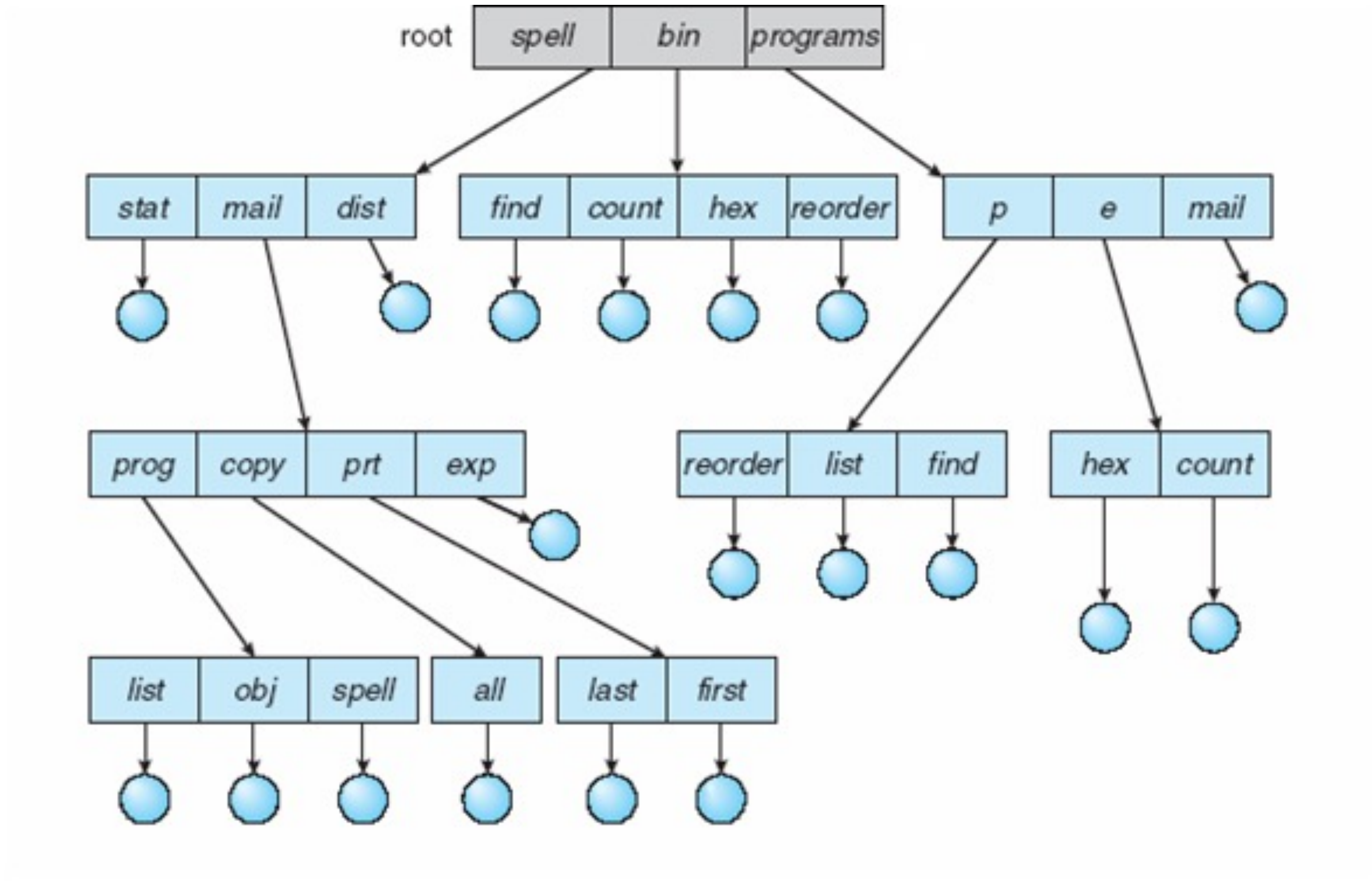
Two-Level Directory

- *Separate directory for each user*
  - Master file directory (MFD)
  - User file directory (UFD)



- 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 (or working directory)
  - E.g., For Linux OS,

```
cd /spell/mail/prog
type list
```

Tree-Structured Directories (Cont.)

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

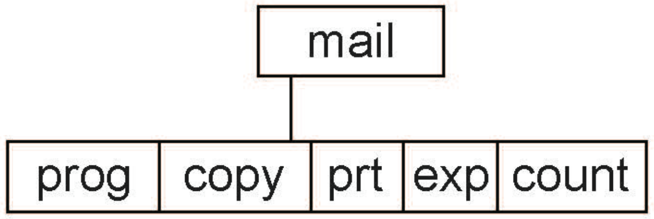
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

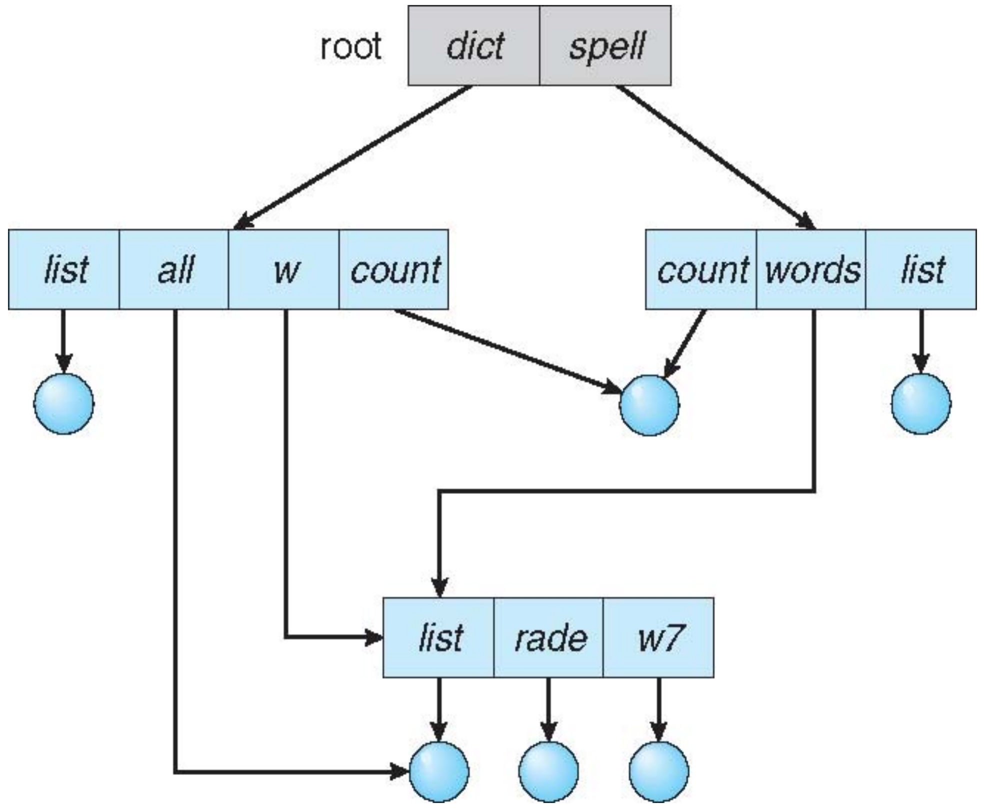
```
mkdir <dir-name>
```

  - Example: if in current directory `/mail`

```
mkdir count
```



Acyclic-Graph Directories

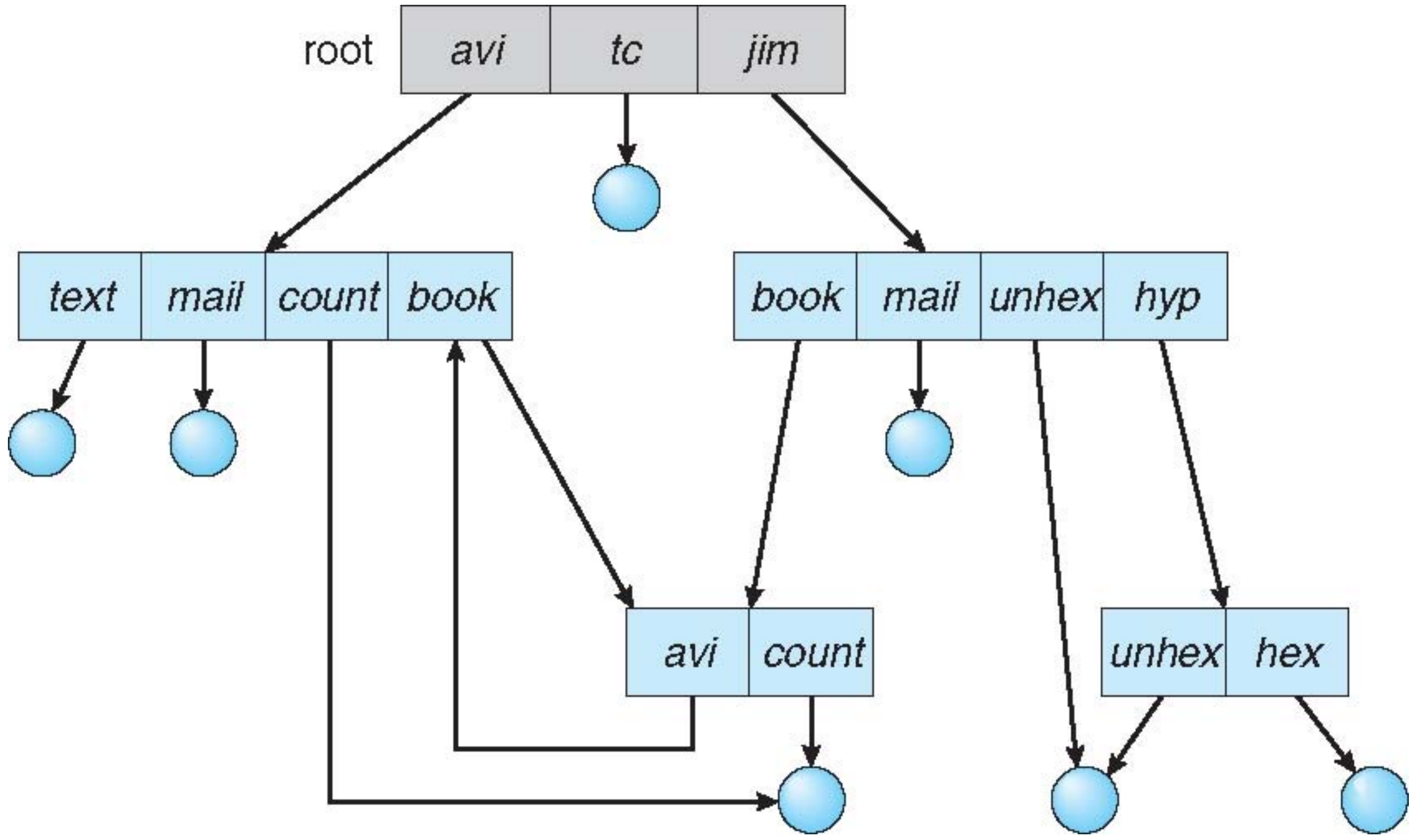


- Have shared subdirectories and files
  - Two different names (*aliasing*)
    - If *dict* deletes *list* ⇒ dangling pointer.
  - Solutions:
    - *Backpointers*, so we can delete all pointers
      Variable size records a problem
      - Backpointers using a daisy chain organization
    - *Entry-hold-count* solution

Acyclic-Graph Directories (Cont.)

- New directory entry type
  - *Link* – another name (pointer) to an existing file
  - *Resolve the link* – follow pointer to locate the file

General Graph Directory

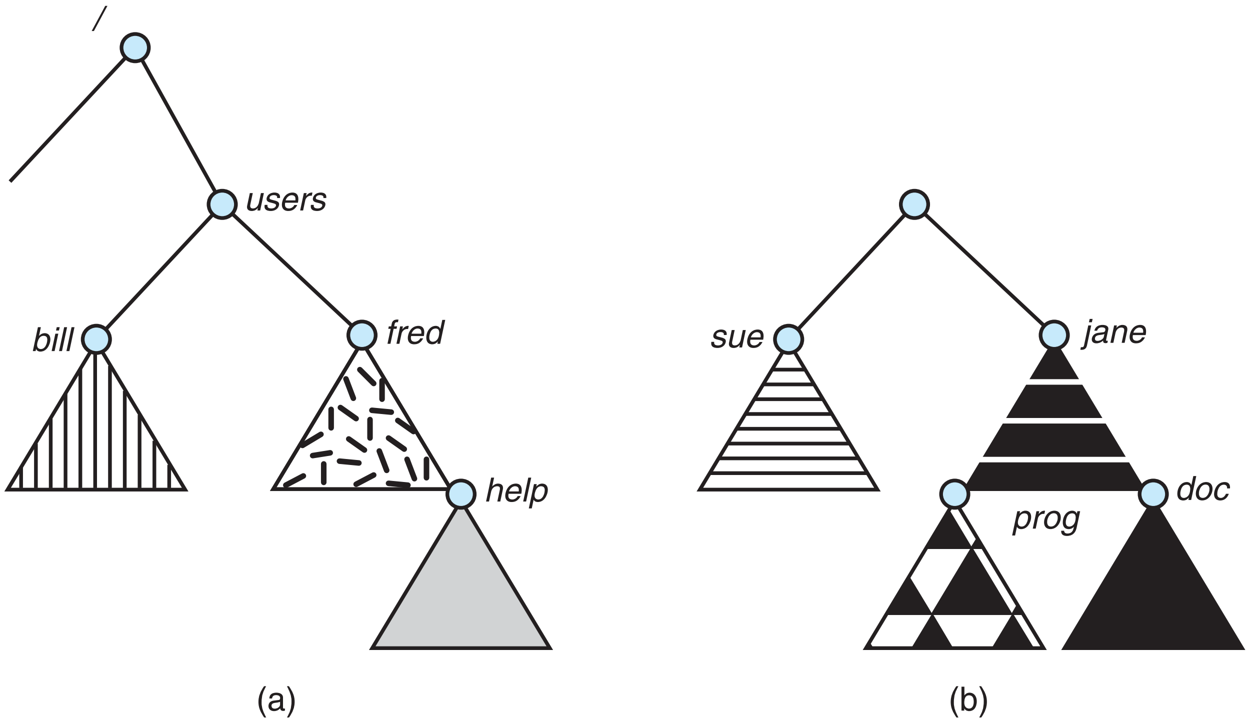


General Graph Directory (Cont.)

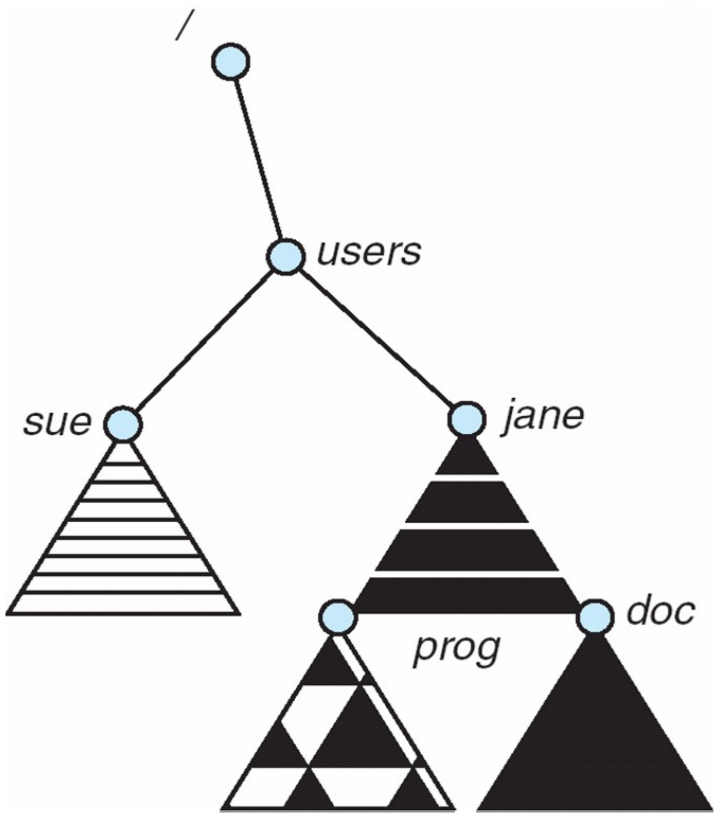
- How do we guarantee no cycles?
  - Allow *only links to file* not subdirectories
  - *Garbage collection*
  - Every time a new link is added use a *cycle detection algorithm* to determine whether it is OK

File System Mounting

- A file system must be *mounted* before it can be *accessed*
- A unmounted file system (i.e., Fig. (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
  - Owner of a file / directory
    - *User IDs* identify users, allowing permissions and protections to be per-user
  - Group of a file / directory
    - *Group IDs* allow users to be in groups, permitting group access rights

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 v.3** include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

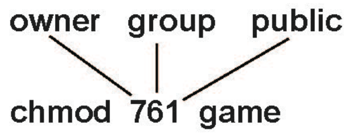
- Specify how multiple users are to access a shared file simultaneously
  - Similar to Ch. 5 *process synchronization algorithms*
    - Tend to be less complex due to disk I/O and network latency (for remote file systems
  - *Andrew File System* (**AFS**) implemented complex remote file sharing semantics
    - AFS has session semantics
      - Writes only visible to sessions starting after the file is closed
  - *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

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* (R), *write* (W), *execute* (X)
- Three classes of users on Unix / Linux



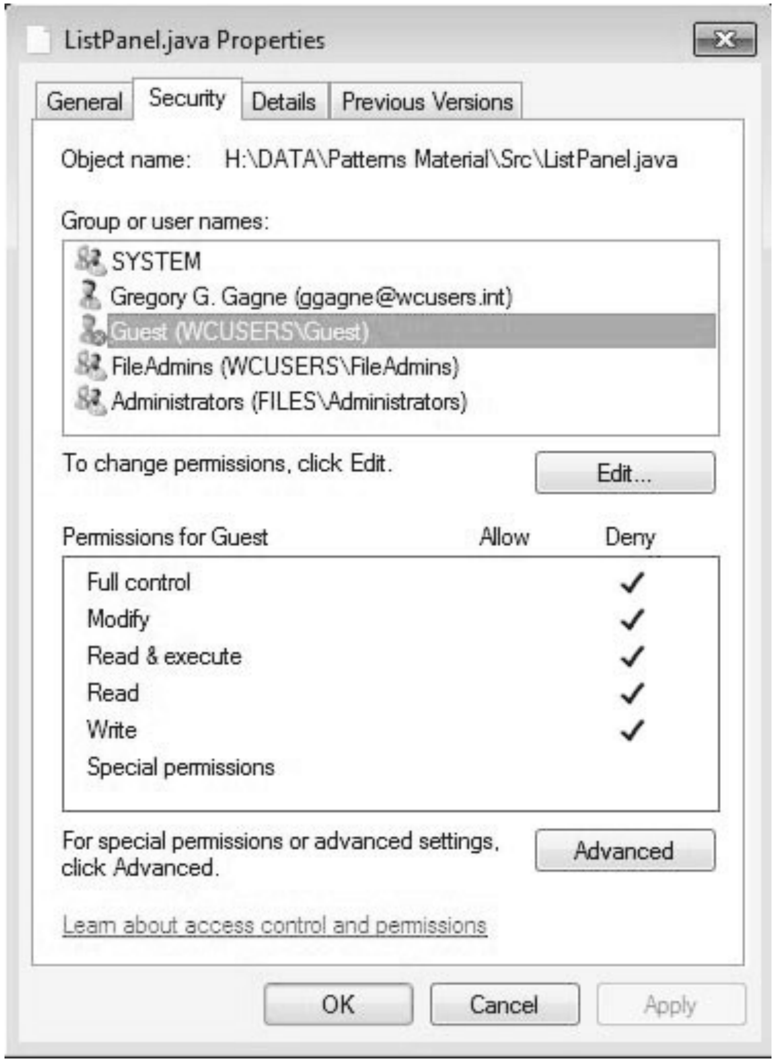
			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- 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.

Attach a group to a file

```
chgrp      G      game
```

Windows 7 Access-Control List Management



ListPanel.java Properties

General | Security | Details | Previous Versions

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. [Edit...]

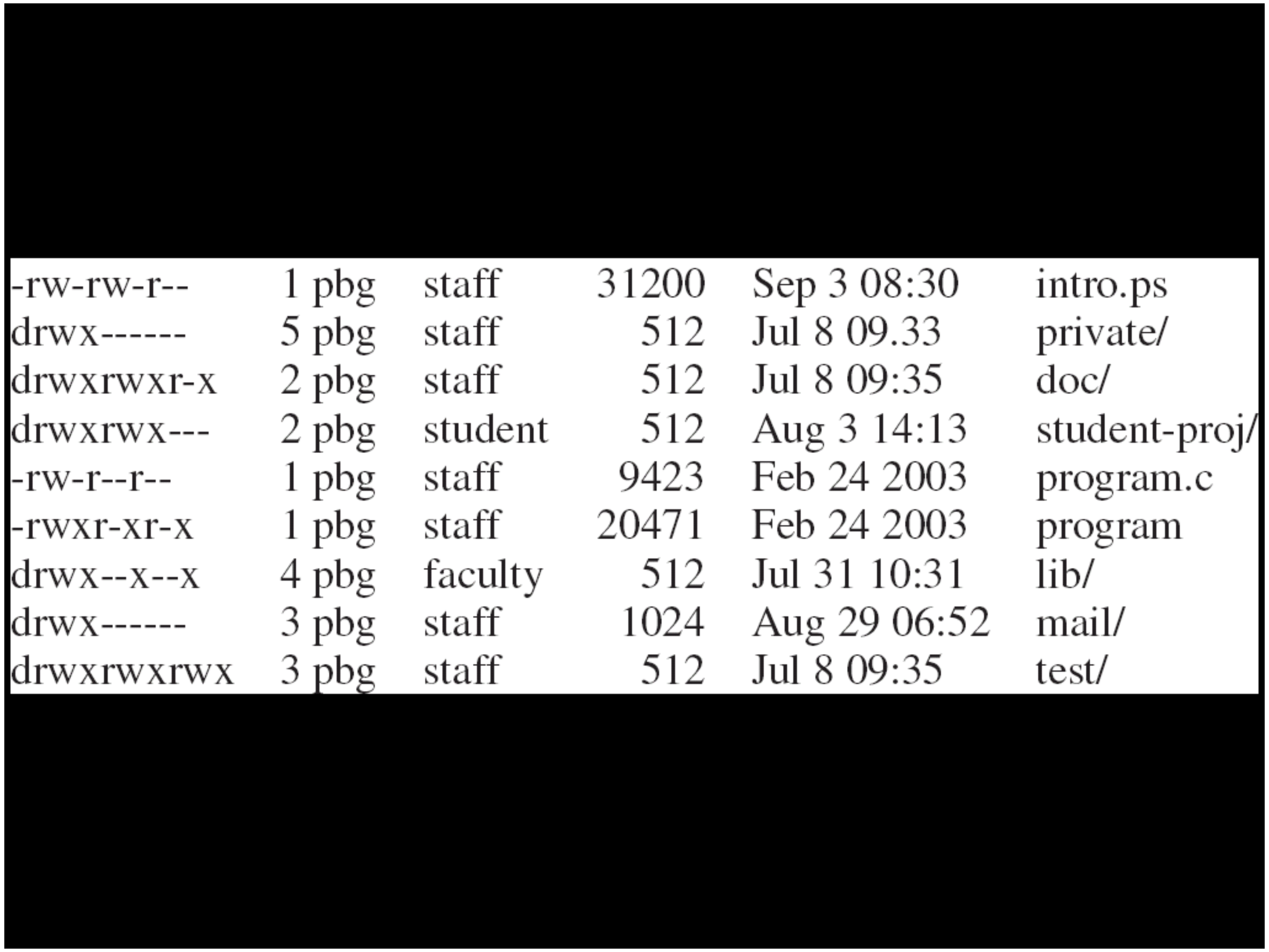
Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced. [Advanced]

Learn about access control and permissions

[OK] [Cancel] [Apply]

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/
```

Summary

- A *file* is an abstract data type defined and implemented by the operating system. It is a *sequence of logical records*. A logical record may be a byte, a line (of fixed or variable length), or a more complex data item. The operating system may specifically support various record types or may leave that support to the application program.
- A major task for the operating system is to *map the logical file concept onto physical storage devices* such as hard disk or NVM device. Since the physical record size of the device may not be the same as the logical record size, it may be necessary to order logical records into physical records. Again, this task may be supported by the operating system or left for the application program.

Summary (Cont.)

- Within a *file system*, it is useful to create *directories* to allow files to be organized. A *single-level directory* in a multiuser system causes naming problems, since each file must have a unique name. A *two-level directory* solves this problem by creating a separate directory for each user's files. The directory lists the files by name and includes the file's location on the disk, length, type, owner, time of creation, time of last use, ...
- The natural generalization of a two-level directory is a *tree-structured directory*. A tree-structured directory allows a user to create subdirectories to organize files. *Acyclic-graph directory structures* enable users to share subdirectories and files but complicate searching and deletion. A general graph structure allows complete flexibility in the sharing of files and direc- tories but sometimes requires garbage collection to recover unused disk space.

Summary (Cont.)

- *Remote file systems* present challenges in reliability, performance, and security. Distributed information systems maintain user, host, and access information so that clients and servers can share state information to man- age use and access.
- Since files are the main information-storage mechanism in most computer systems, *file protection* is needed on multiuser systems. Access to files can be controlled separately for each type of access — read, write, execute, append, delete, list directory, and so on. File protection can be provided by access lists, passwords, or other techniques.

End of Chapter 10.A.