Chapter 10: File-System Interface

Chapter 10: File-System Interface

- File Concept
- Access Methods
- 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 Systems

- The file system consists of two distinct parts:
  - a **collection of *files***, each storing related data, and
  - a ***directory structure***, which organizes and provides information about all the files in the system.

File Concept

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes and optical disks.
- So that the computer system will be convenient to use, the operating system provides a **uniform logical view** of information storage.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit: ***the file.***
- Files are mapped by the operating system onto physical device.
- These storage devices are usually nonvolatile, so the contents are persistent through power failures and system reboots.

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- From the user's point of view the file is the **smallest block** of logical secondary storage.
  - Data cannot be written to secondary storage, unless they are within a file.

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

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

File Operations

- File is an **abstract data type**
- **Create**
- **Write**
- **Read**
- **Reposition within file**
- **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-file Table

- The operating system keeps a small table, called the **open-file table**, containing information about all open files.
- When a file operation is requested, the file is specified via an index into this table, so **no searching is required**.
- When the file is no longer being actively used, it is *closed* by the process, and the operating system removes its entry from the open-file table.
- Create and delete are system calls that work with closed rather than open files.

Open Files

- Several pieces of data are needed to manage 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 close it
  - **Disk location** of the file: cache of data access information (information needed to locate the file on disk)
  - **Access rights**: per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
- 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

Internal File Structure

- It is unlikely that the physical record size will exactly match the length of the desired logical record.
  - Logical records may even vary in length.
- **Packing** a number of logical records into physical blocks is a common solution to this problem.
  - For example, the UNIX operating system defines all files to be simply streams of bytes. Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks say, 512 bytes per block.
- Because disk space is always allocated in blocks, some portion of the last block of each file is generally wasted. The waste incurred to keep everything in units of blocks (instead of bytes) is **internal fragmentation**.
- All file systems suffer from internal fragmentation;
  - **the larger the block size the greater the internal fragmentation.**

Access Methods

- **Sequential Access**

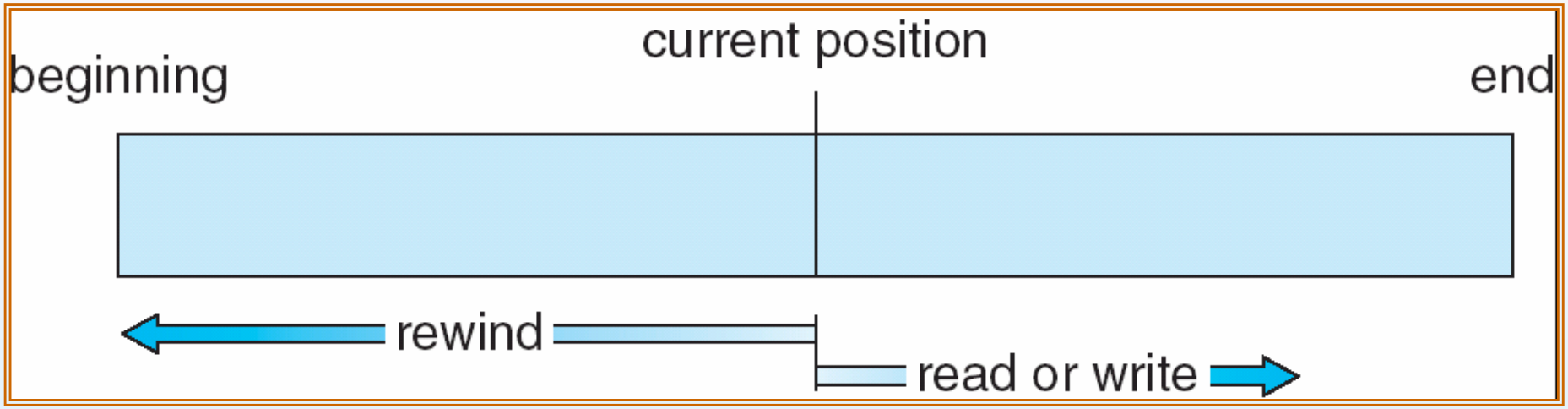
  read next
  write next
  reset
  no read after last write
  (rewrite)

- **Direct Access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  n = relative block number

Sequential-access File



The simplest access method is sequential access. Information in the file is processed in order, one record after the other.

This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

Direct Access

- For direct access, the file is viewed as a **numbered sequence of blocks** or records. Thus, we may read block 14, then read block 53, and then write block 7.
- There are no restrictions on the order of reading or writing for a direct-access file.
- Direct-access files are of great use for immediate access to **large amounts** of information.
- **Databases** are often of this type.
  - When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

Simulation of Sequential Access on a Direct-access File

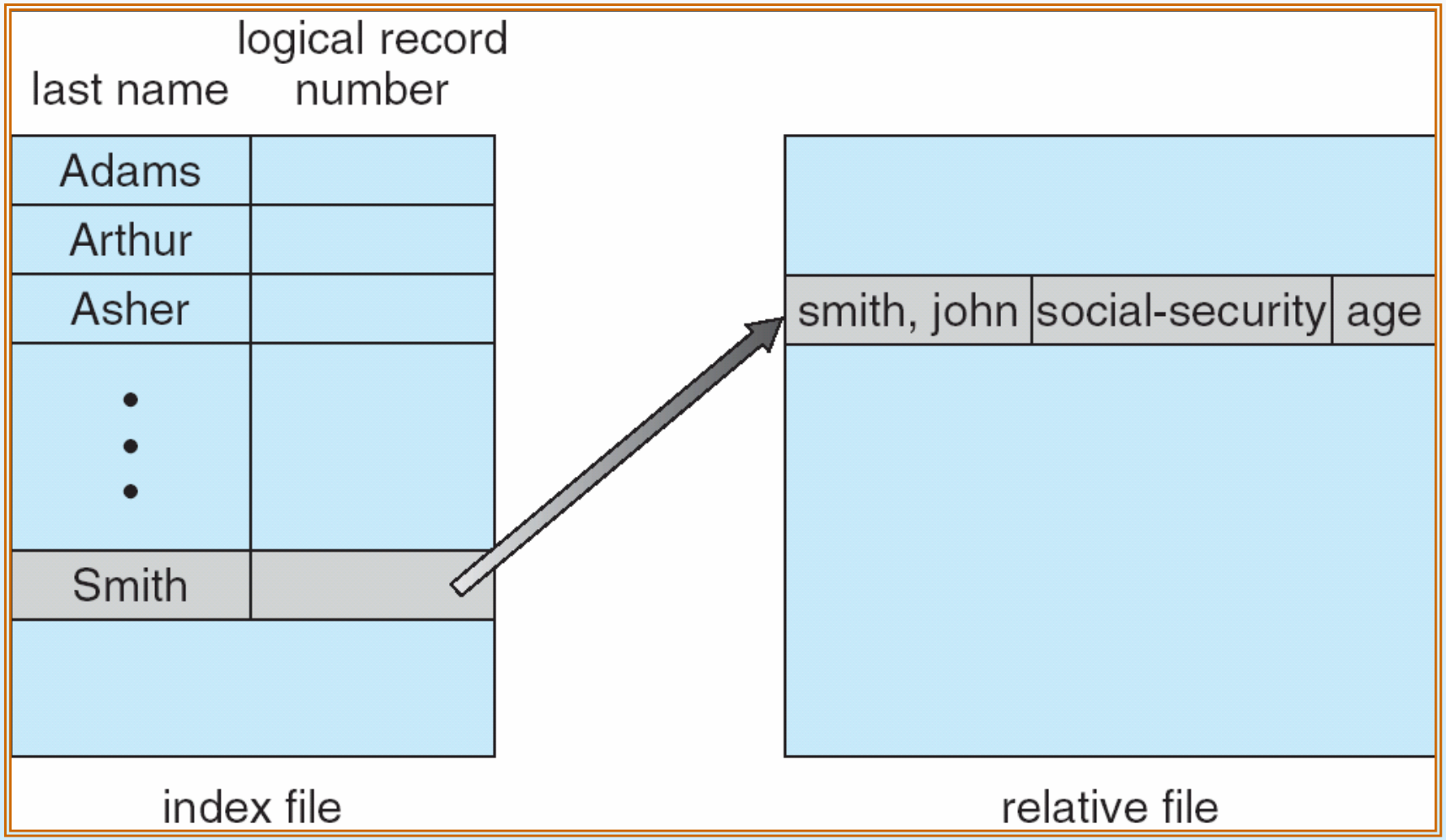
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

cp: current position

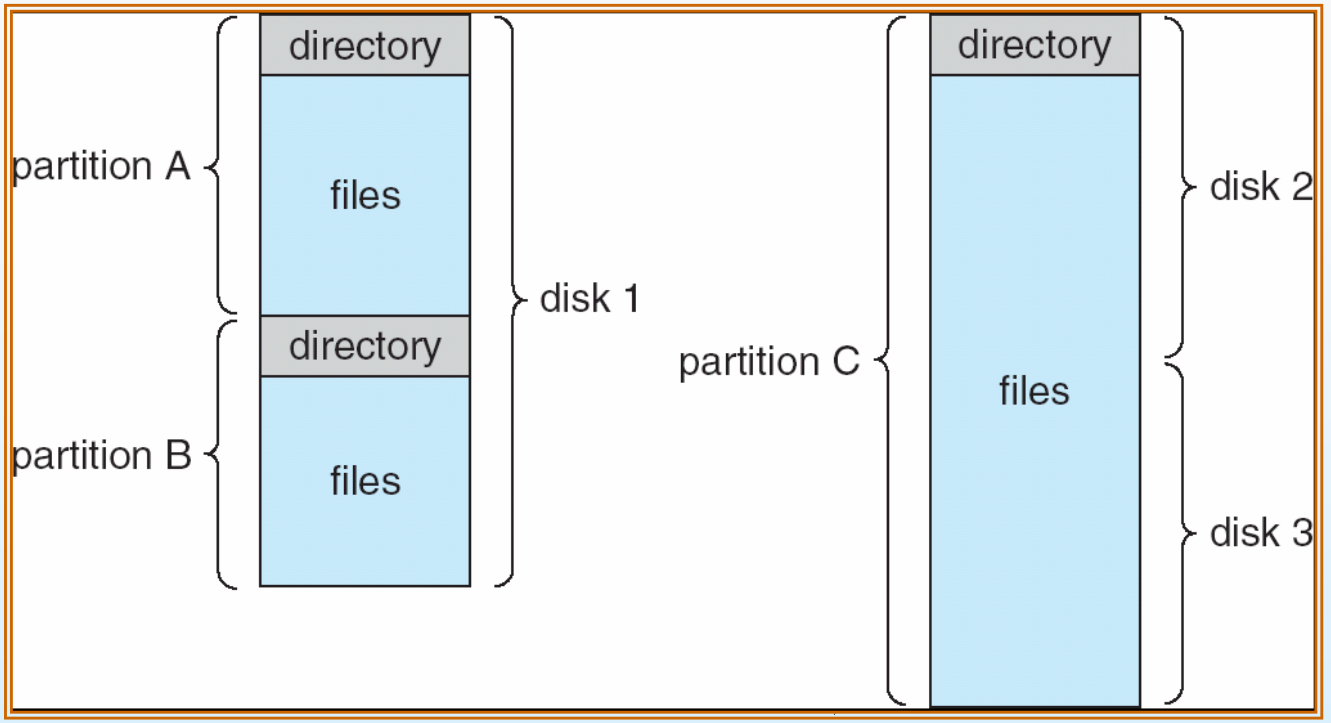
Other Access Methods

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an **index** for the file.
- The index, like an index in the back of a book, contains pointers to the various blocks.
- To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

Example of Index and Relative Files



A Typical File-system Organization

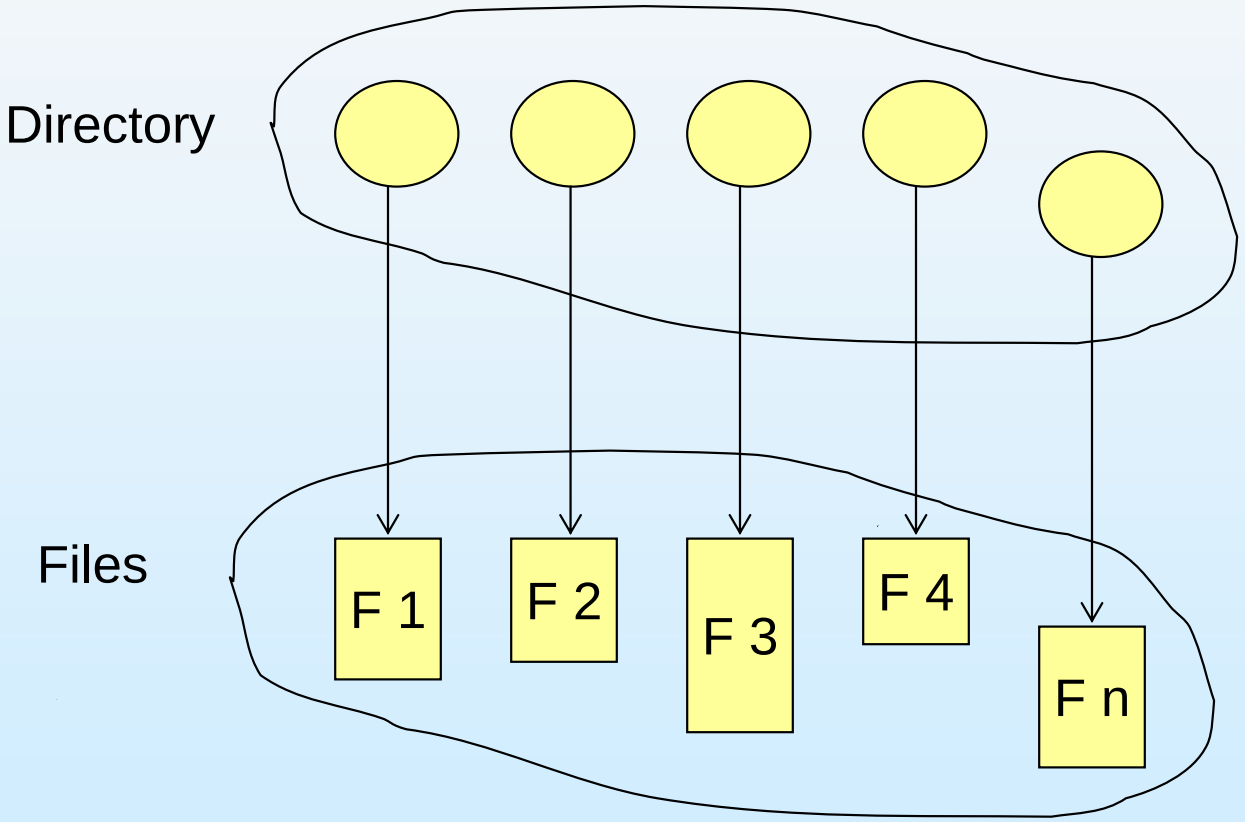


Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory** or **volume table of contents**.

The **device directory** (more commonly known simply as a **directory**) records information-such as name, location, size, and type-for all files on that volume.

Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

Operations Performed on Directory

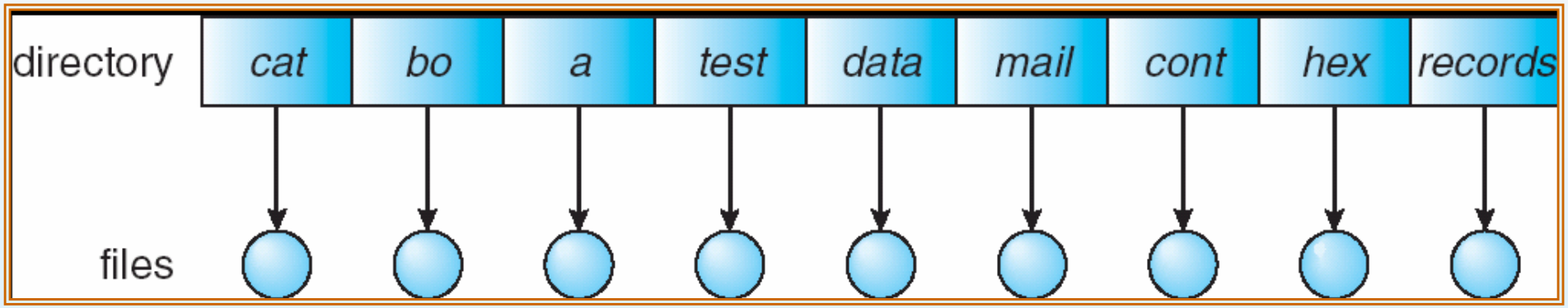
- The directory can be viewed as a symbol table that translates file names into their directory entries.
- Operations on a directory
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

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

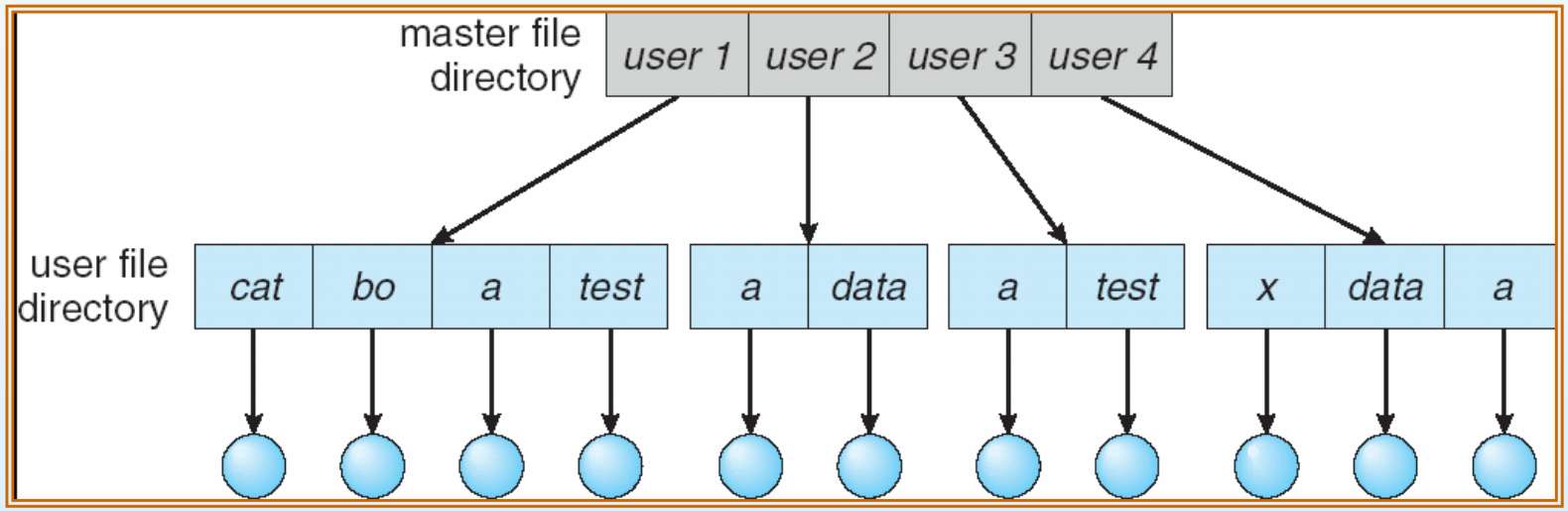


Naming problem

Grouping problem

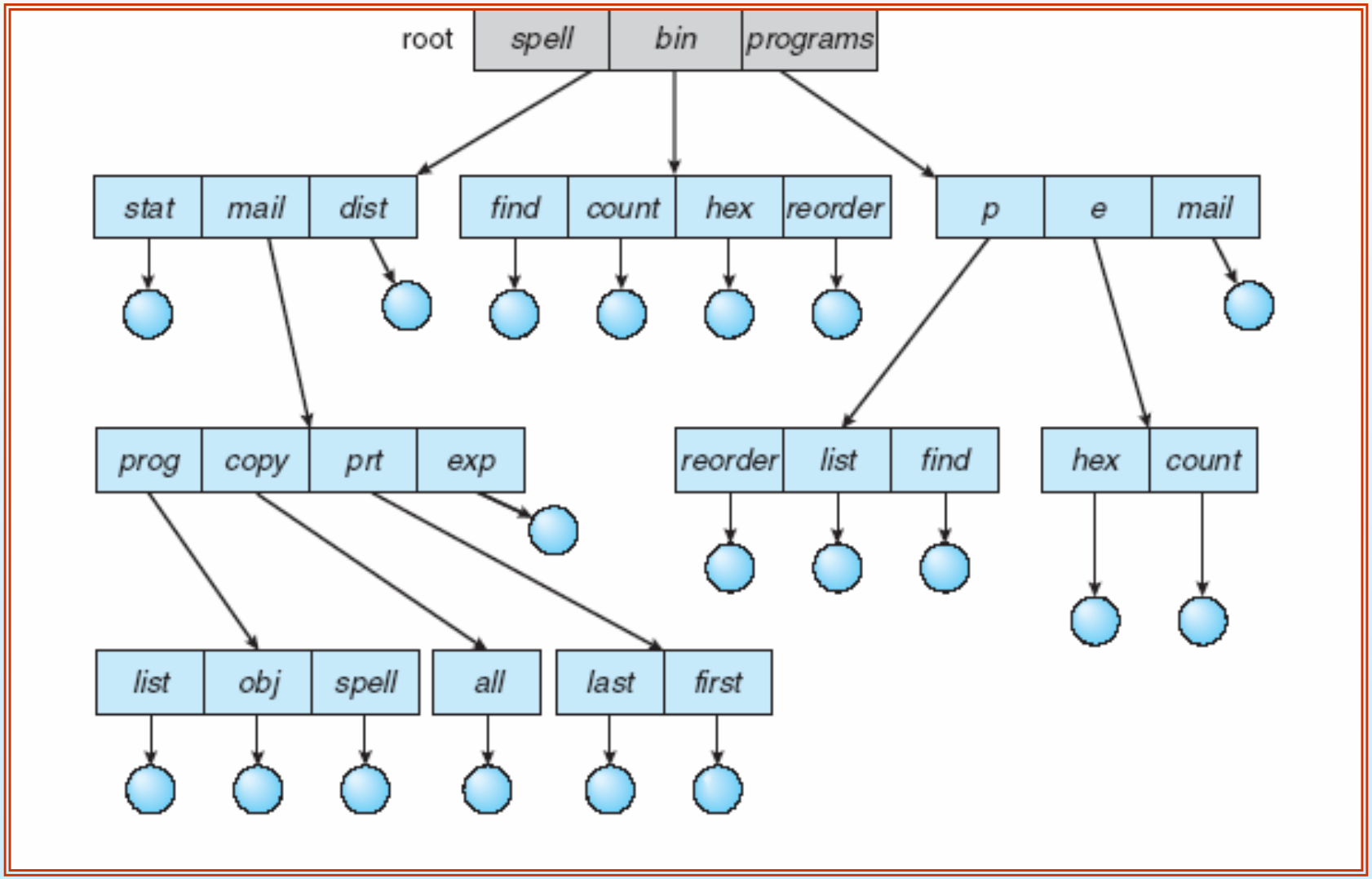
Two-Level Directory

- 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)

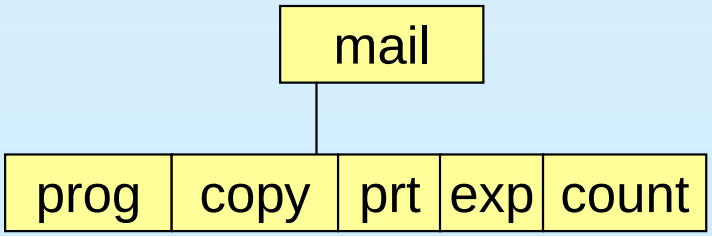
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /mail

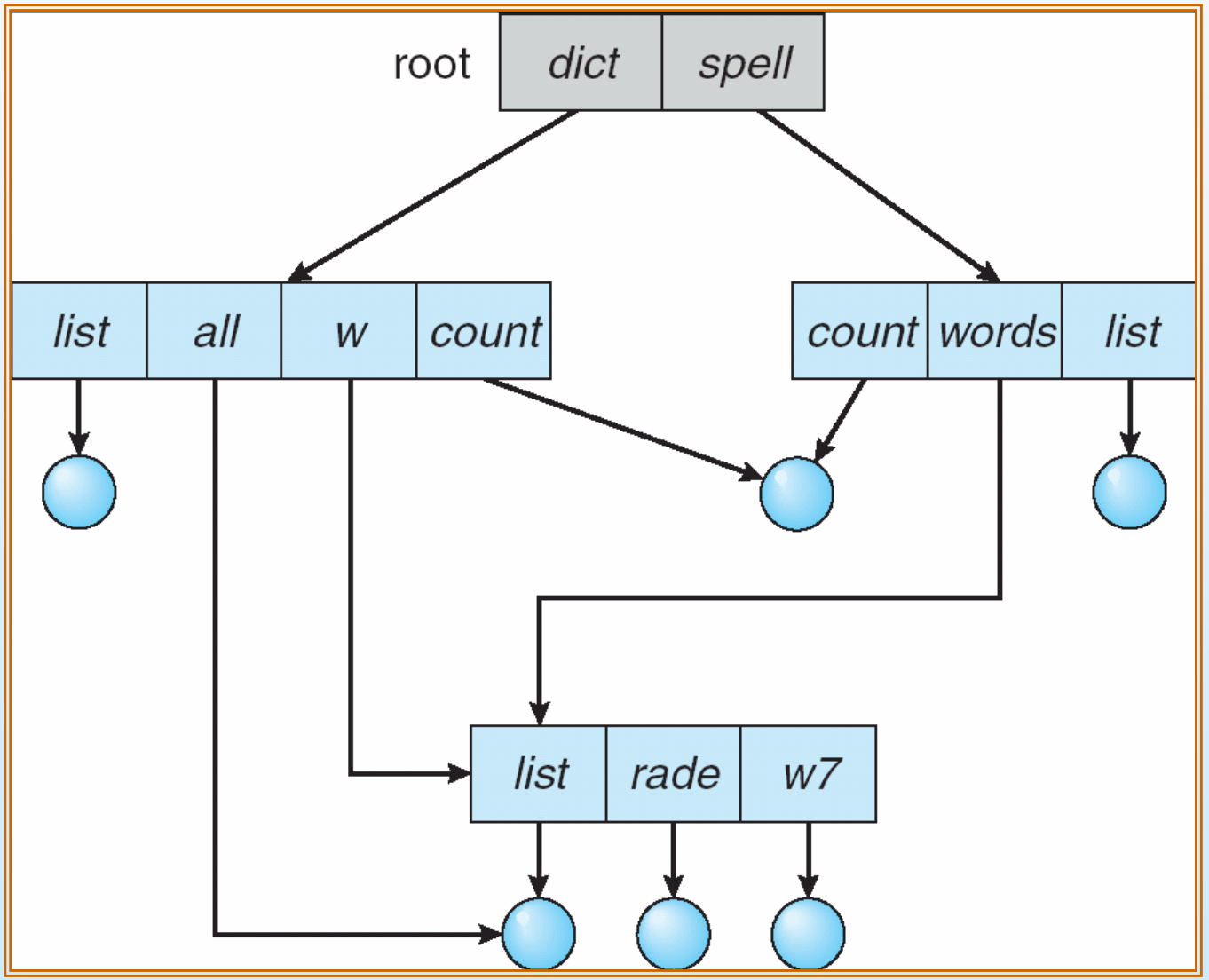
  mkdir count



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

Acyclic-Graph Directories

- Have shared subdirectories and files

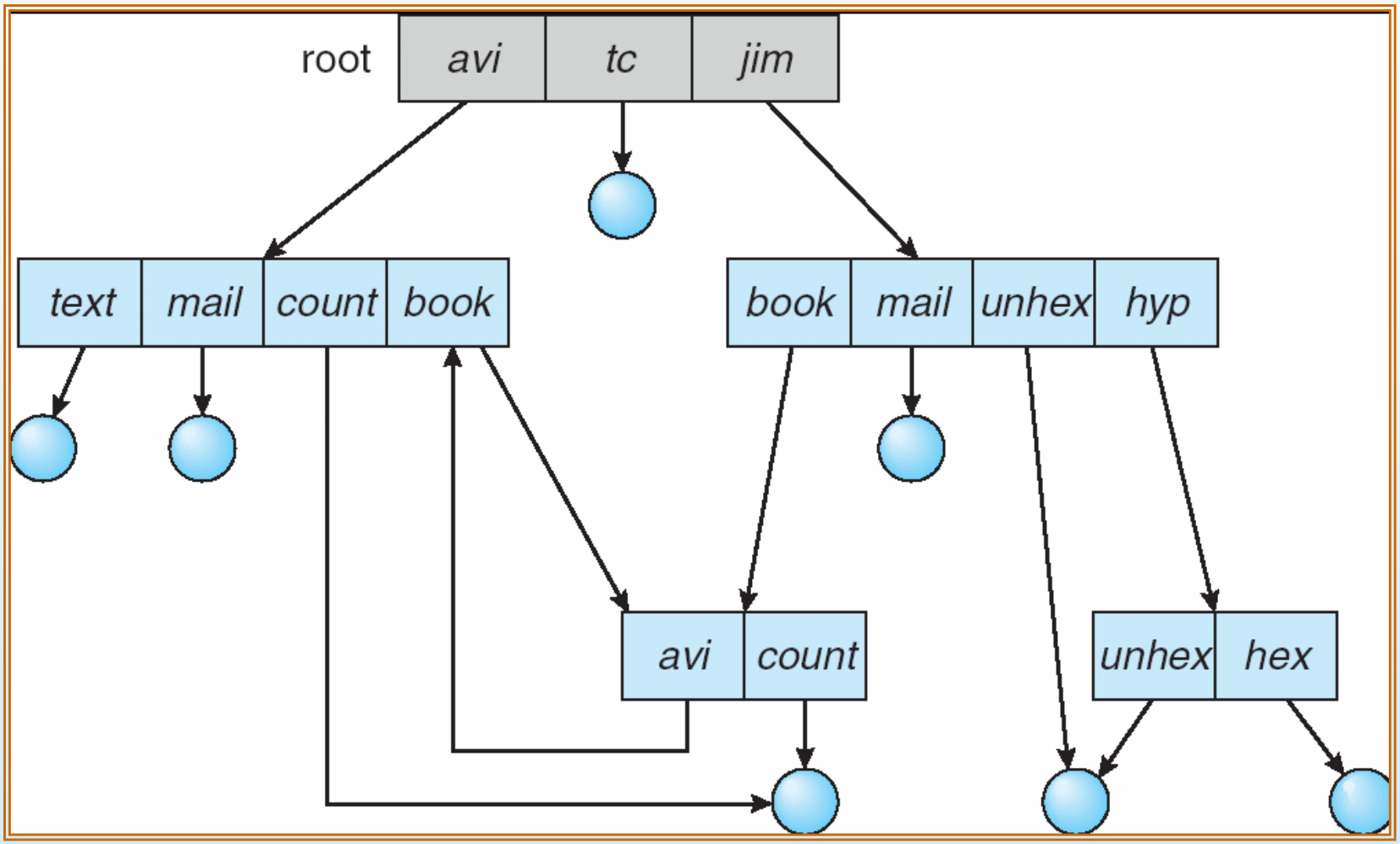


Acyclic-Graph Directories (Cont.)

Problems

- A file may now have multiple absolute path names.
  - Consequently distinct file names may refer to the same file.
- **Deletion**:
  - Removing the file whenever anyone deletes it, may leave dangling pointers to the now-nonexistent file
- Solutions to deletion:
  - Backpointers, so we can delete all pointers
  - Do no delete: preserve until all references to it are deleted
    - Keep list of all references to a file
    - Keep a Reference count
- New directory entry type
  - When new link enters: ReferenceCount++
  - When a link is deleted: ReferenceCount--

General Graph Directory



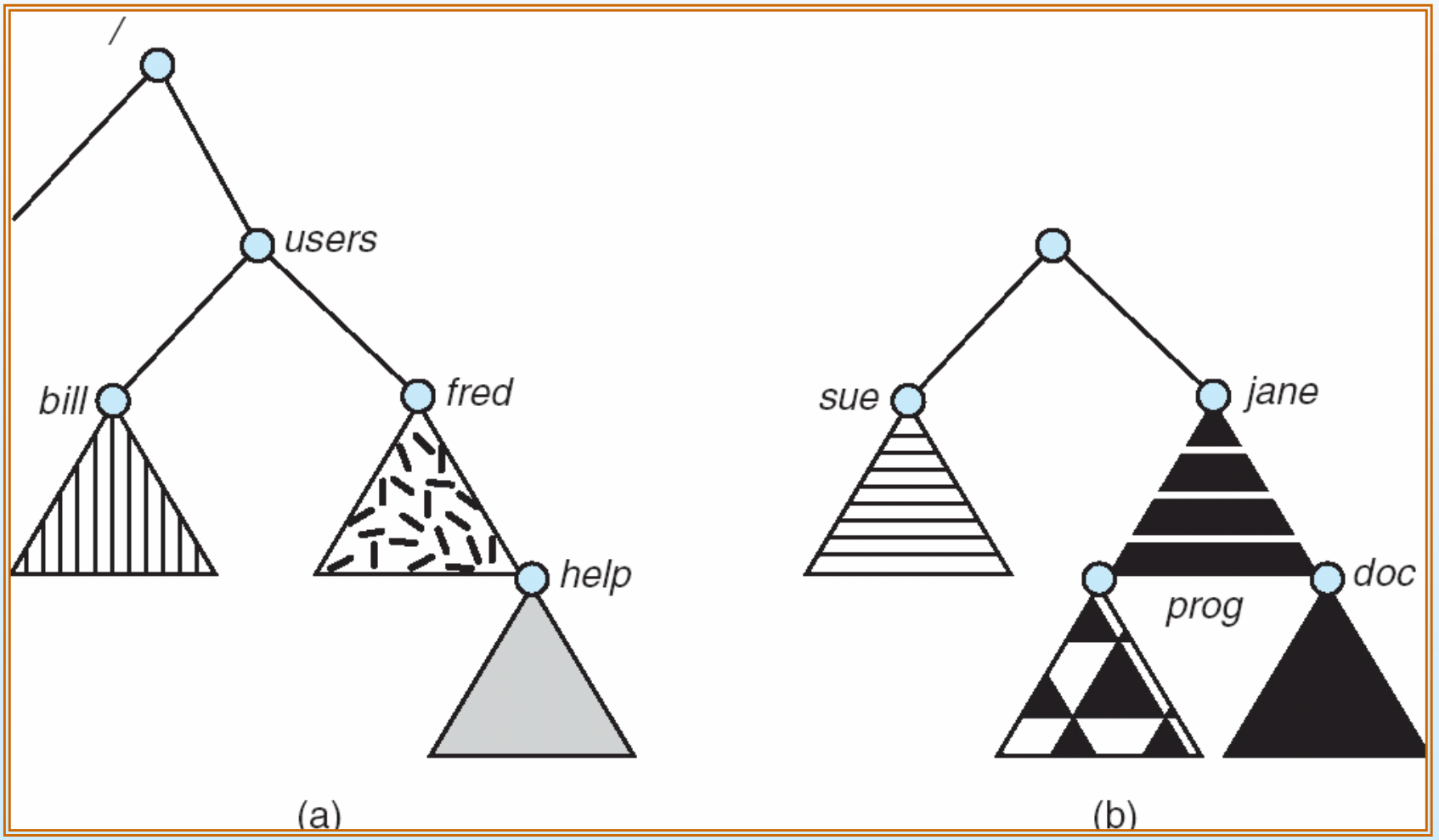
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Garbage collection
    - In order to determine when the last reference has been deleted and thus avoid cycles
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

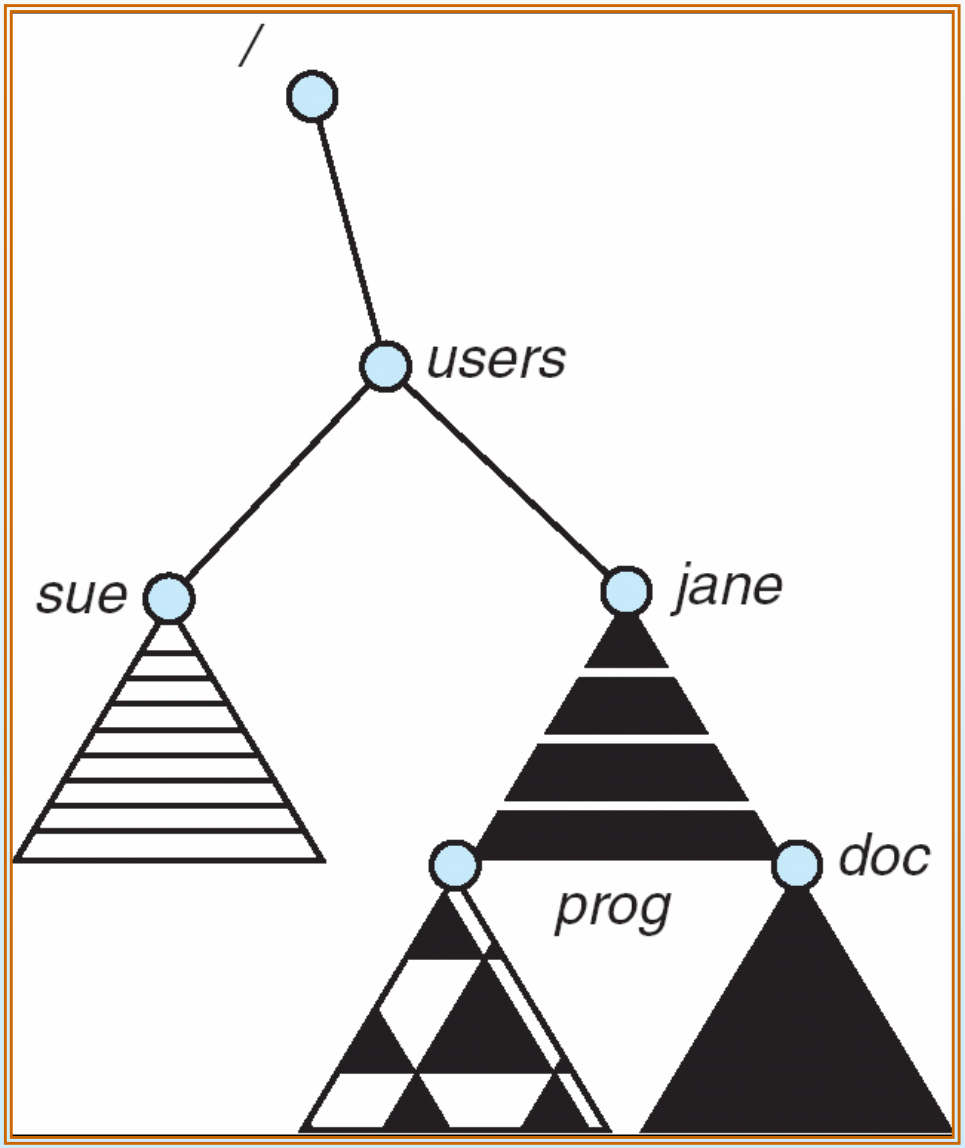
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**

(a) Existing. (b) Unmounted Partition



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

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **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

- 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 include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 7 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

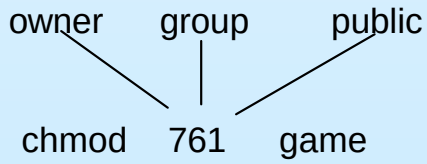
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access-Control Lists and Groups

- **Mode of access: read, write, execute**
- **Three classes of users**

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

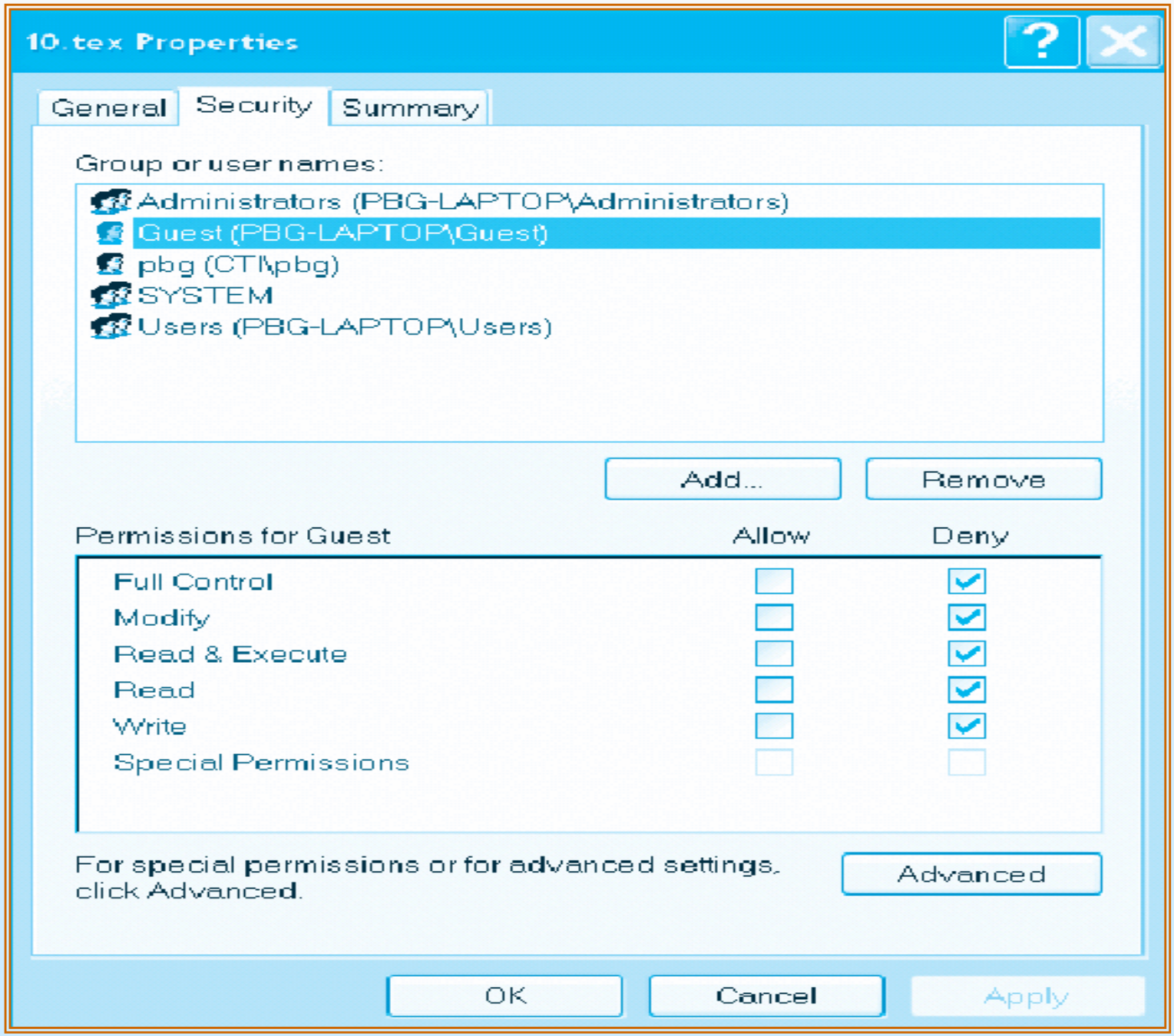
- **Ask manager to create a group (unique name), say G, and add some users to the group.**
- **For a particular file (say *game*) or subdirectory, define an appropriate access.**



Attach a group to a file

chgrp G game

Windows XP Access-control List Management



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/

End of Chapter 10

Readings

- Chapter 10.