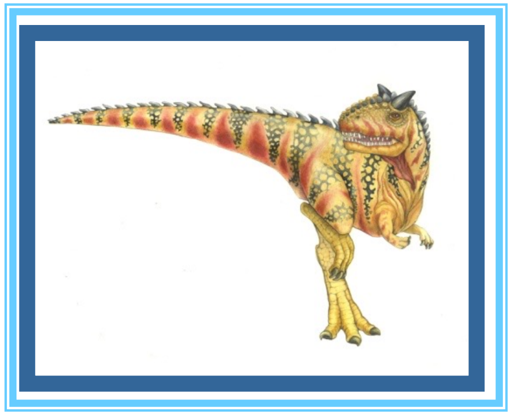
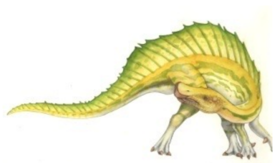
Chapter 10: File-System Interface



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

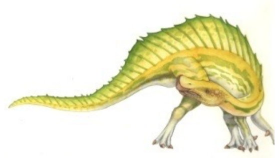
- 3 criteria for long-term information storage:
  - Should be able to store very large amount of information
  - Information must survive the processes using it
  - Should provide concurrent access to multiple processes
- Solution:
  - Store information on disks in units called **files**
  - Files are persistent, and only owner can explicitly delete it
  - Files are managed by the OS
- File Systems: How the OS manages files!

File System

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.

- File System Components
  - Disk Management: collecting disk blocks into files
  - Naming: Interface to find files by name, not by blocks
  - Protection: Layers to keep data secure
  - Reliability/Durability（持久）: Keeping of files durable despite crashes, media failures, attacks, etc

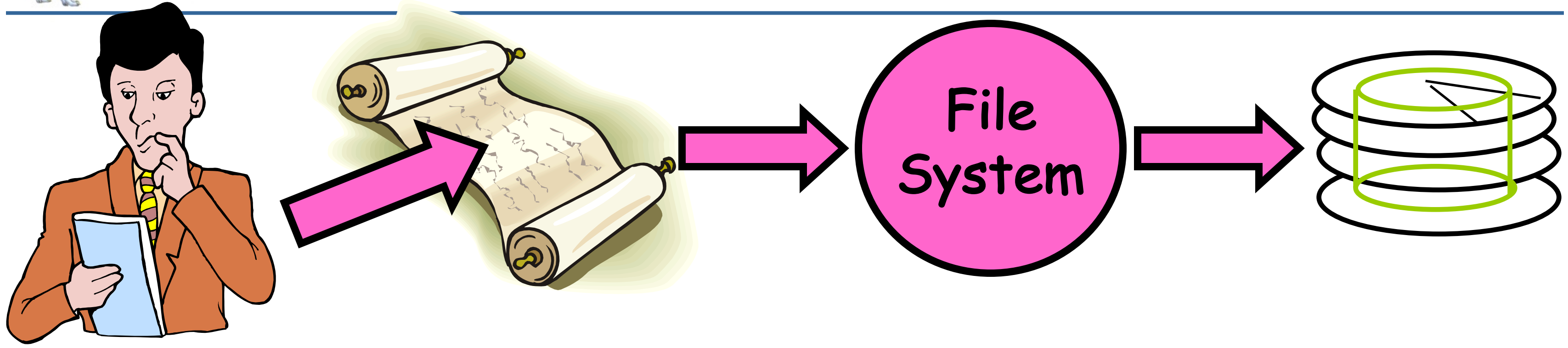
5



File System

- User vs. System View of a File
  - User's view:
    - Durable Data Structures
  - System's view (system call interface):
    - Collection of Bytes (UNIX)
    - Doesn't matter to system what kind of data structures you want to store on disk!
  - System's view (inside OS):
    - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
    - Block size ≥ sector size; in UNIX, block size is 4KB

Translating from User to System View

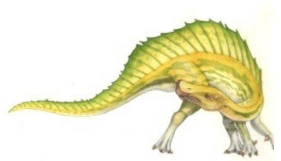


- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block
- Everything inside File System is in whole size blocks（块为单位）
  - For example, `getc()`, `putc()` ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks (i.e. systems view inside OS)

7

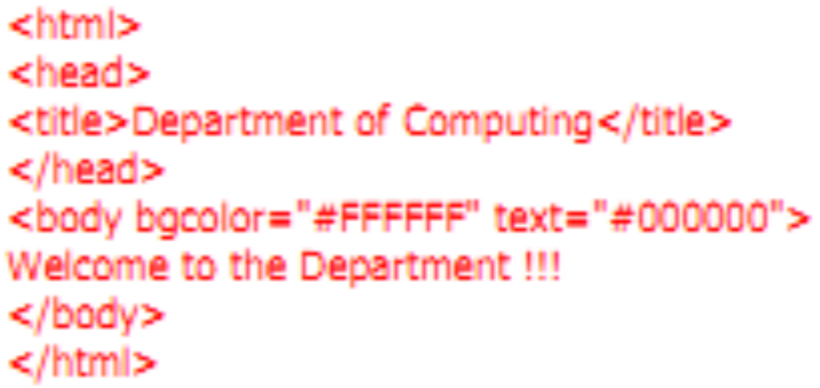
File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program



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（可以对方法1增加控制符实现后两种,如下表）
- Who decides:
  - Operating system
  - Program

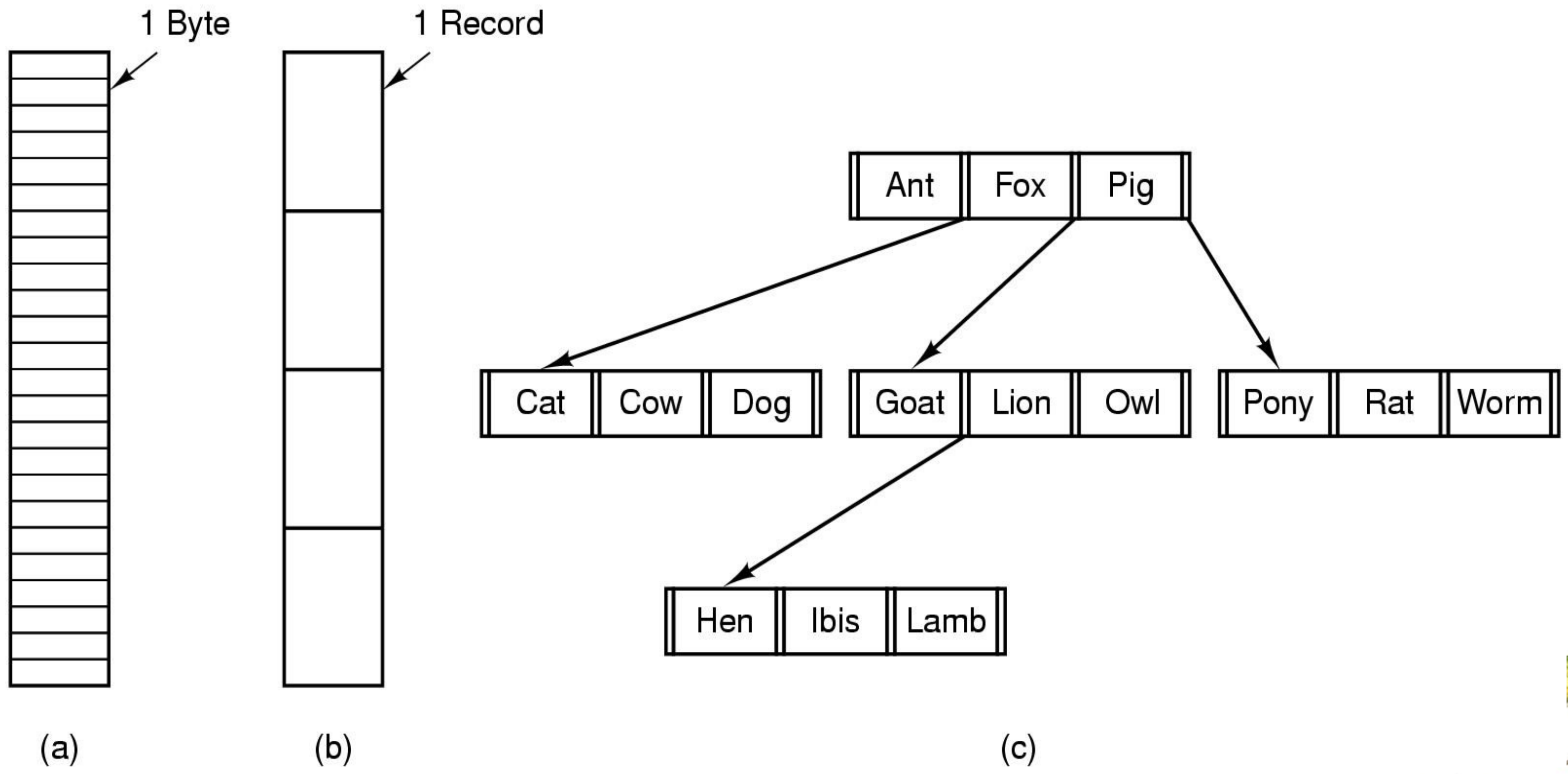


```
<html>
<head>
<title>Department of Computing</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
Welcome to the Department !!!
</body>
</html>
```

File Structure

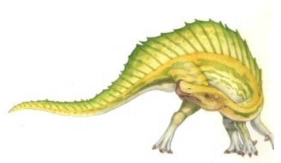
- Byte Sequence: unstructured, most commonly used(a)
- Record sequence: r/w in records, used earlier(b)
- Complex structures, e.g. tree(c)

  - Data stored in variable length records; location decided by OS



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

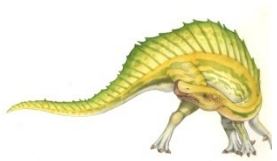
- File is an **abstract data type**
- Basic file operations
  - **Create**
  - **Write**
  - **Read**
  - **Reposition within file**
  - **Delete**
  - **Truncate** (删除内容,保留文件目录)

File Operations

- **Create**
  - Allocates disk space
  - Makes an entry in the file directory
- **Delete**
  - Searches the directory
  - Releases all file space
  - Erases the directory entry for the file

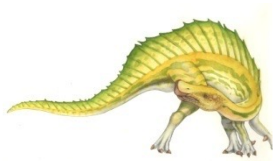
File Operations

- **Read**
  - requires specifying file name and the memory location where the next block of the file to be put
  - Searches the directory to find the file's location
  - Reads the block of the file into memory
  - Updates the read pointer



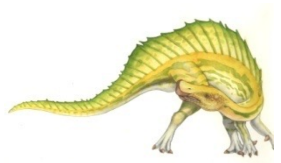
File Operations

- **Write**
  - requires specifying file name and the information to be written to the file
  - Searches the directory to find the file's location
  - Writes to the file according to the write pointer to the location in the file
  - Updates the writer pointer



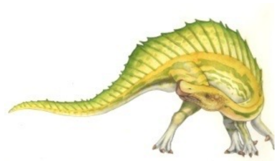
File Operations

- **Repositioning within a file**
  - Known as a file seek
  - Searches the directory to find the file's entry
  - Updates the current-file-position pointer to a given value
- **Turncate**
  - Erases the contents of a file but keeps its attributes
    - ✓ Only the file length be reset to 0
  - Releases its file space



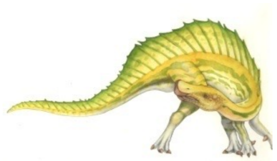
File Operations

- ***Open(F_i)*** **:**make a file ready for reading/writing.
  - searches the directory structure on disk for entry F_i
  - moves the content of entry to memory
- ***Close (F_i)*** **:** mark the completion of operation on file.
  - moves the content of entry F_i in memory to directory structure on disk



Open Files

- Most of the file operations involve searching the directory for the entry of the named file. To avoid this constant searching, many OS require that an open() system call be made before a file first used actively.
- The OS keeps a small table, called the open-file table, containing information about all open files.
- The open() takes a file name and searches the directory, copying the directory entry into the open-file table.
- The open() system call typically returns a pointer to the entry in the open-file table.

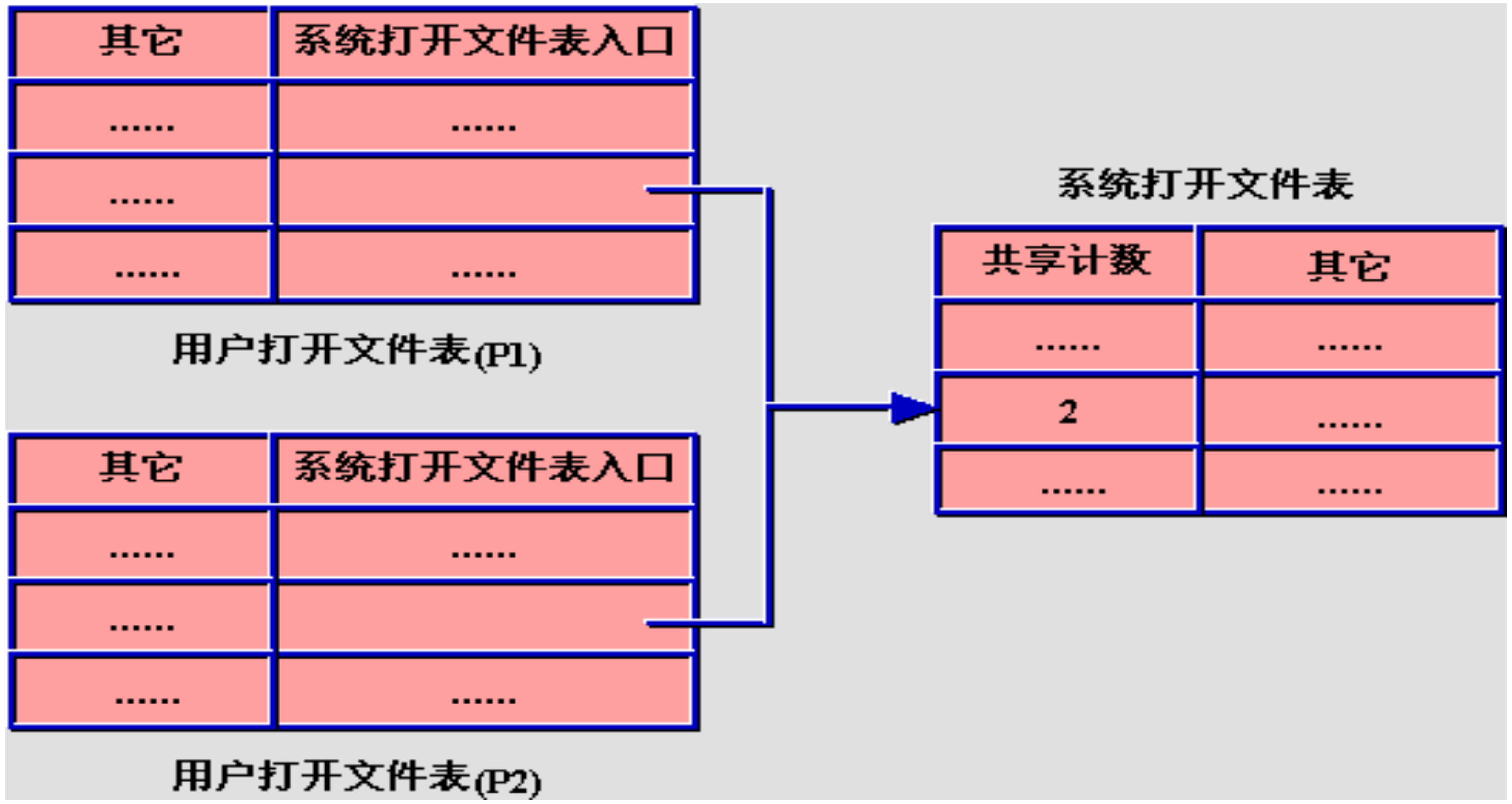


Open Files

- There may be several processes opening the same file at the same time.
- OS uses two levels of internal tables:
  - Process(or User) open-file table
    - Tracks all files opened by a process
    - Stores information about the use of the file, e.g. current file pointer, access rights and accounting information.
    - Each entry in process open-file table in turn points to a system-wide open-file table.
  - System open-file table
    - Contains process-independent information, such as the disk location of the file, access date, file size and open count.

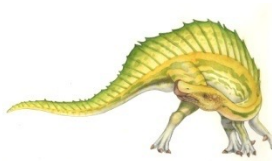
Open Files

- Process open-file table/System open-file table



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;
  - Access rights: per-process access mode information
  - 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: the information to locate the file on the disk;

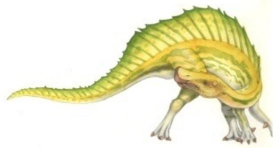


Open File Locking

- File locks are useful for files that are shared by several processes which provided by some operating systems and file systems
- File locks allow one process to lock a file and prevent other processes from gaining access to it.
- File locks provide functionality similar to reader-writer locks.
  - Shared lock: like reader lock, allowing reading currently
  - Exclusive lock: like writer lock, only one process can acquire at a time.

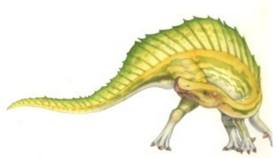
Open File Locking

- Further more ,OS may provide either mandatory or advisory file-locking mechanisms.
  - **Mandatory(强制锁)** – access is denied depending on locks held and requested
  - **Advisory(建议锁)** – 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)

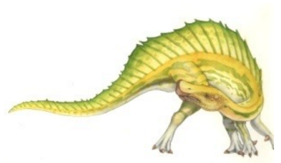
```
                // this locks the second half of the file - shared
                sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                SHARED);
                /** Now read the data . . . */
                // release the lock
                sharedLock.release();
           } catch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

File Naming

- Motivation: Files abstract information stored on disk
  - You do not need to remember block, sector, …
  - We have human readable names
- How does it work?
  - Process creates a file, and gives it a name
    - Other processes can access the file by that name
  - Naming conventions are OS dependent
    - Usually names as long as 255 characters is allowed
    - Digits and special characters are sometimes allowed
    - MS-DOS and Windows are not case sensitive, UNIX family is

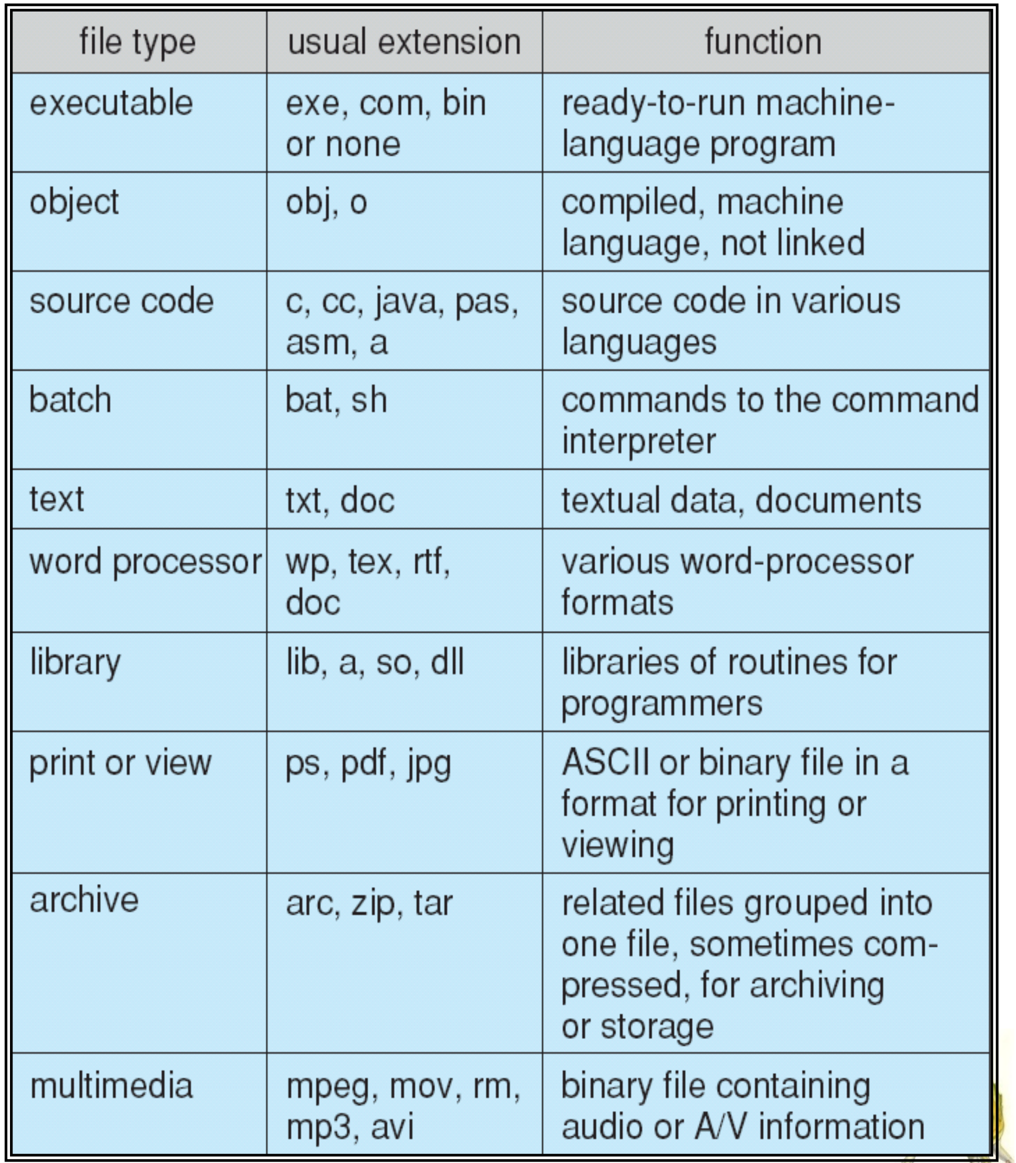
File Extensions

- Name divided into 2 parts, second part is the extension
- On UNIX, extensions are not enforced by OS
  - However C compiler might insist on its extensions
    - These extensions are very useful for C
- Windows attaches meaning to extensions
  - Tries to associate applications to file extensions



File Types – Name, Extension

- Two major types of files:
  - Program files
    - Source code
    - Object code
    - Executable program
  - Data files
    - Character or ASCII file
    - Binary file
    - Free-formatted text file
    - Formatted or structured file
- There are different subtypes of files.
  - They are often indicated by the file 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

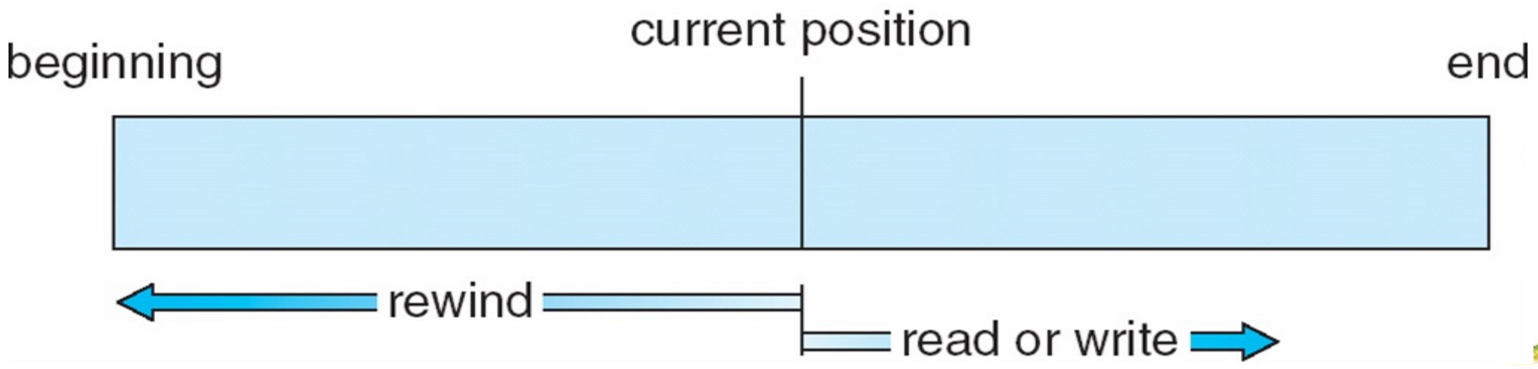
File System Patterns

- How do users access files?
  - Sequential Access
    - bytes read in order ("give me the next X bytes, then give me next, etc")
  - Random Access
    - read/write element out of middle of array ("give me bytes i—j")

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

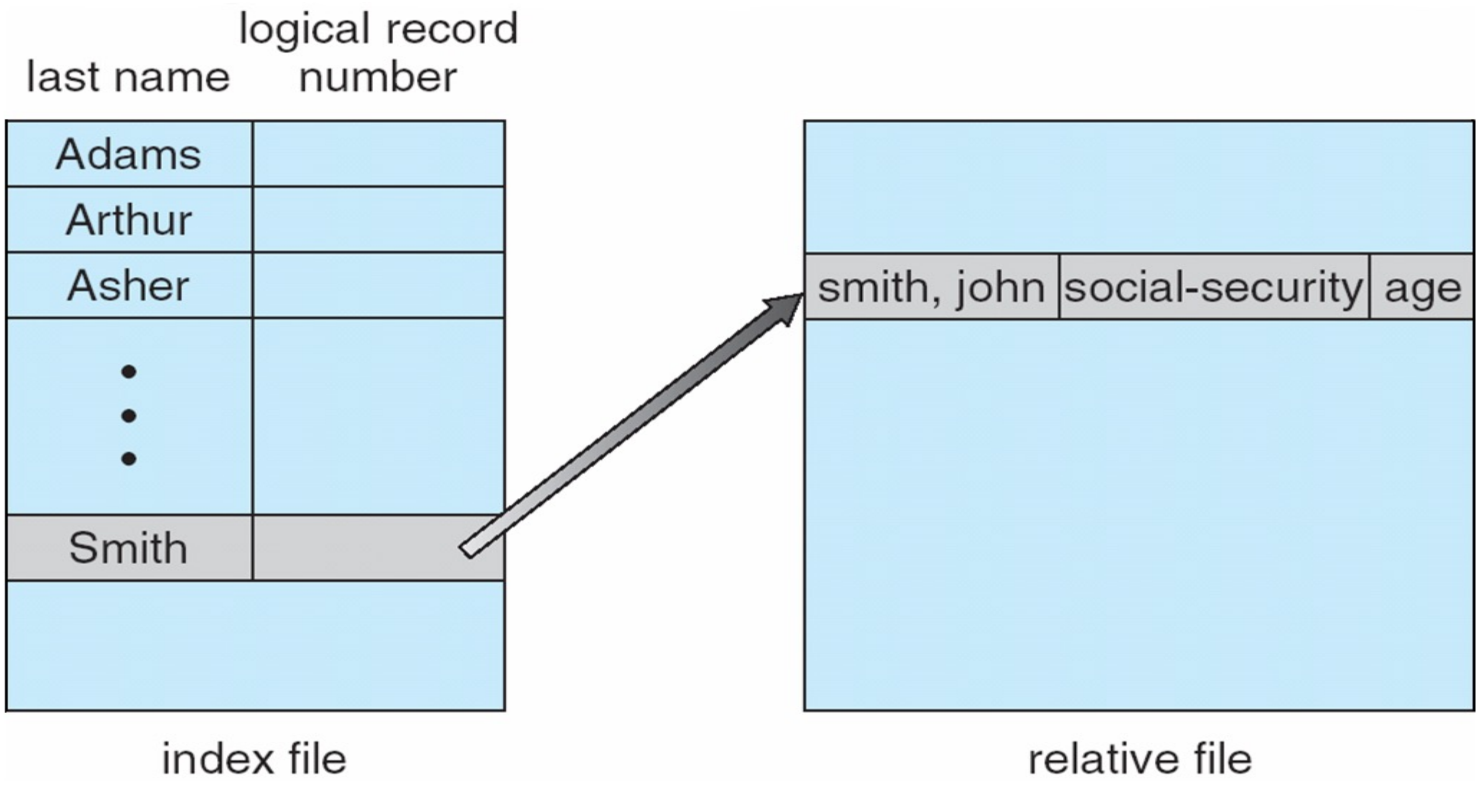


Simulation of Sequential Access on Direct-access File

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

Example of Index and Relative Files

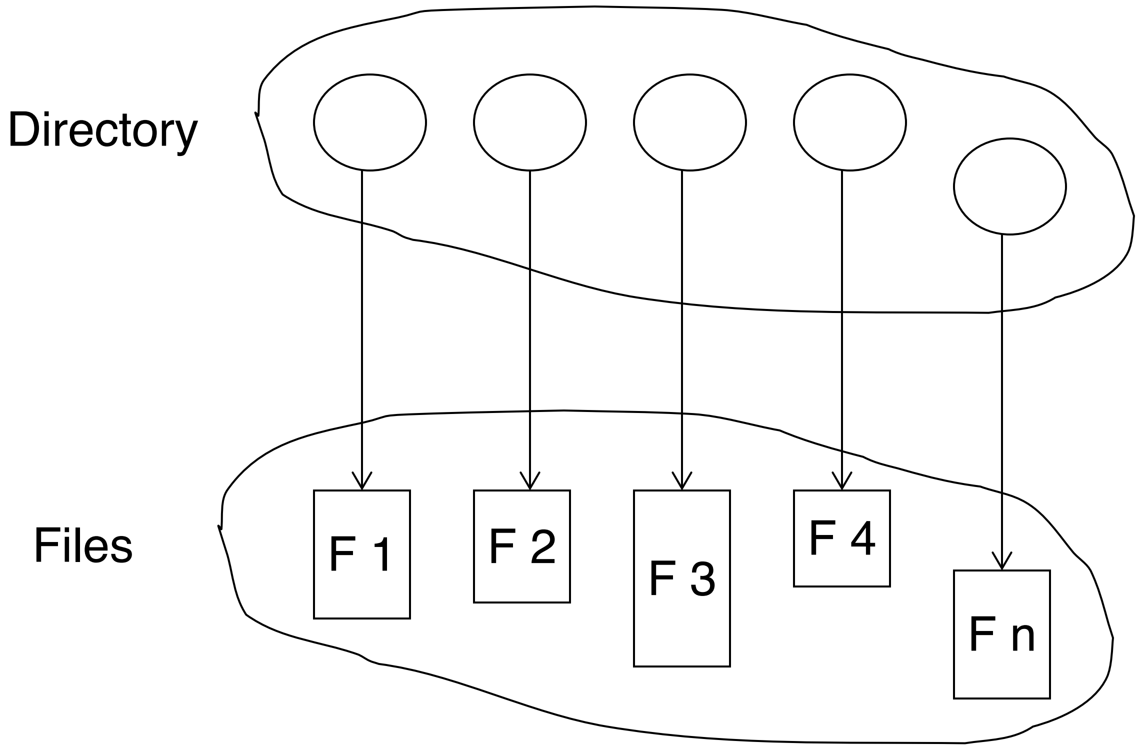
- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.



- 存取方法与索引结构有直接关系

Directory Structure

- Some systems store millions of file on disk, to manage all these data, we need to organize them. This organization involves the use of directory. A directory is a collection of nodes containing information about all files.

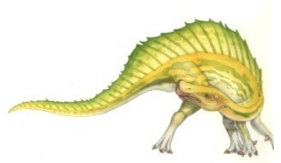


- Both the directory structure and the files reside on disk
- Backups of these two structures are kept on tapes

Disk Structure

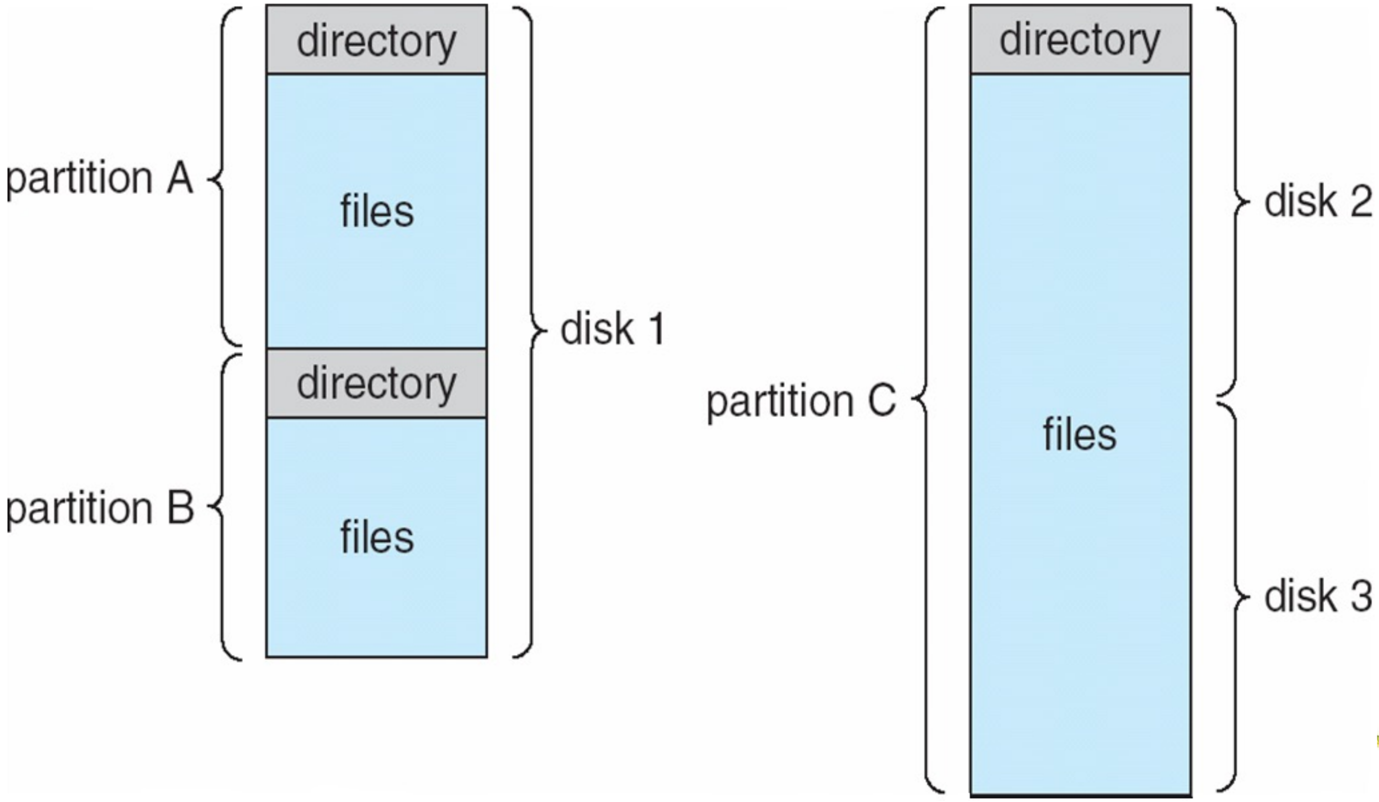
- Disk can be subdivided into partitions， Partitions also known as minidisks, slices;
- Disks or partitions can be RAID protected against failure;
- Disk or partition can be used raw – without a file system, or formatted with a file system;
- Entity containing file system known as a volume;
- Each volume containing file system also tracks that file system's info in device directory or volume table of contents;

*RAID(Redundant Array of Independent Disks)



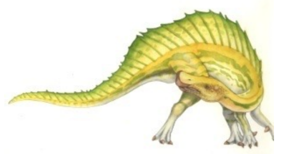
A Typical File-system Organization

- A disk may be divided into many parts.
  - A part may hold an individual file system. This is called a partition.
- Sometimes, several disks are combined together to hold one large file system.
  - This collection of disks is also called a partition



Operations Performed on Directory

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

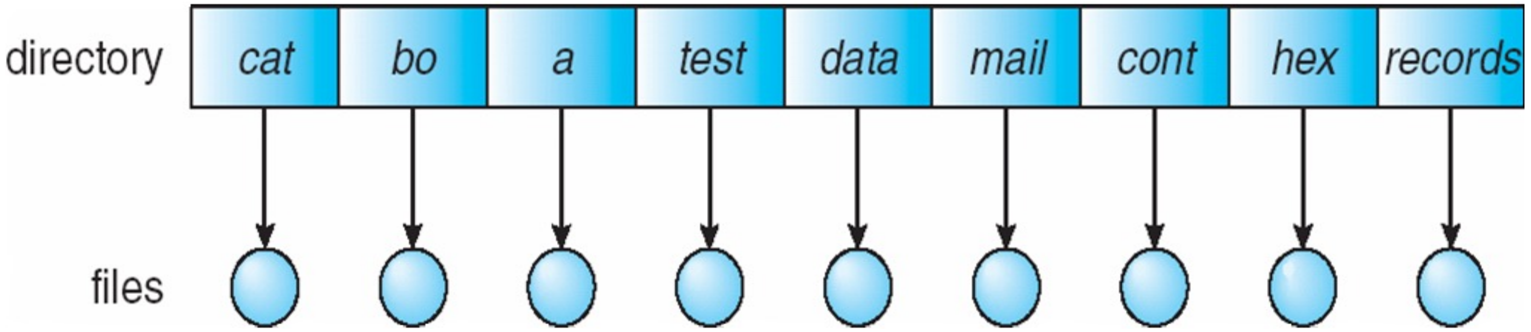


Organize the Directory (Logically) to Obtain

- 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

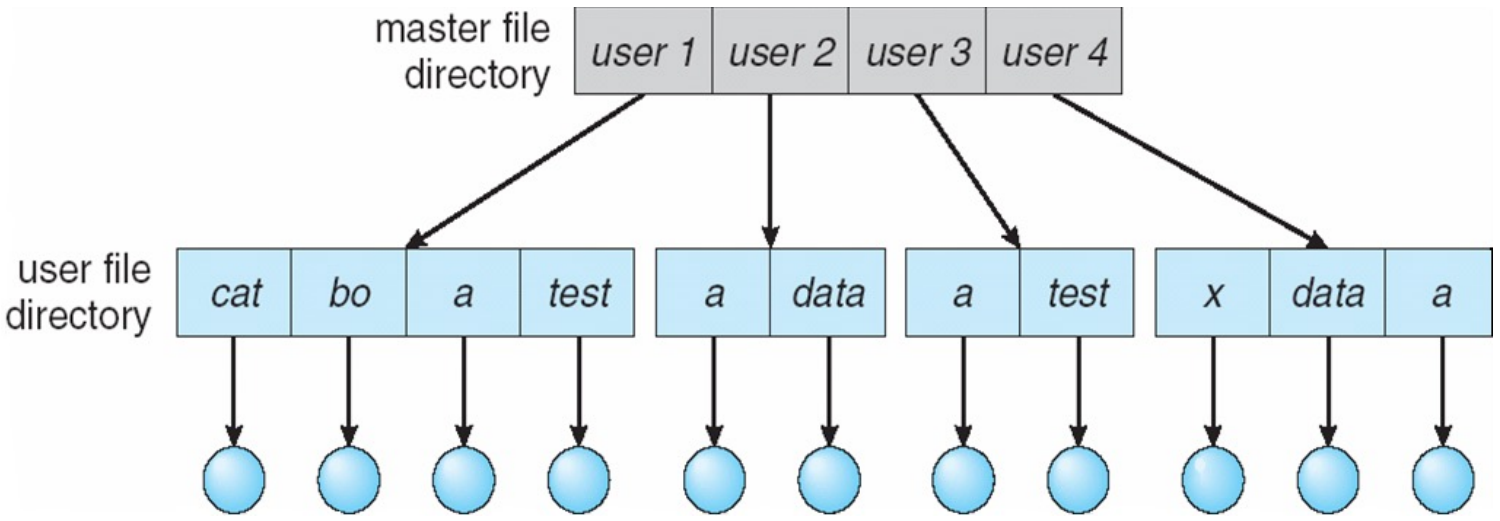
- A single directory for all users



- Naming problem
- Grouping problem

Two-Level Directory

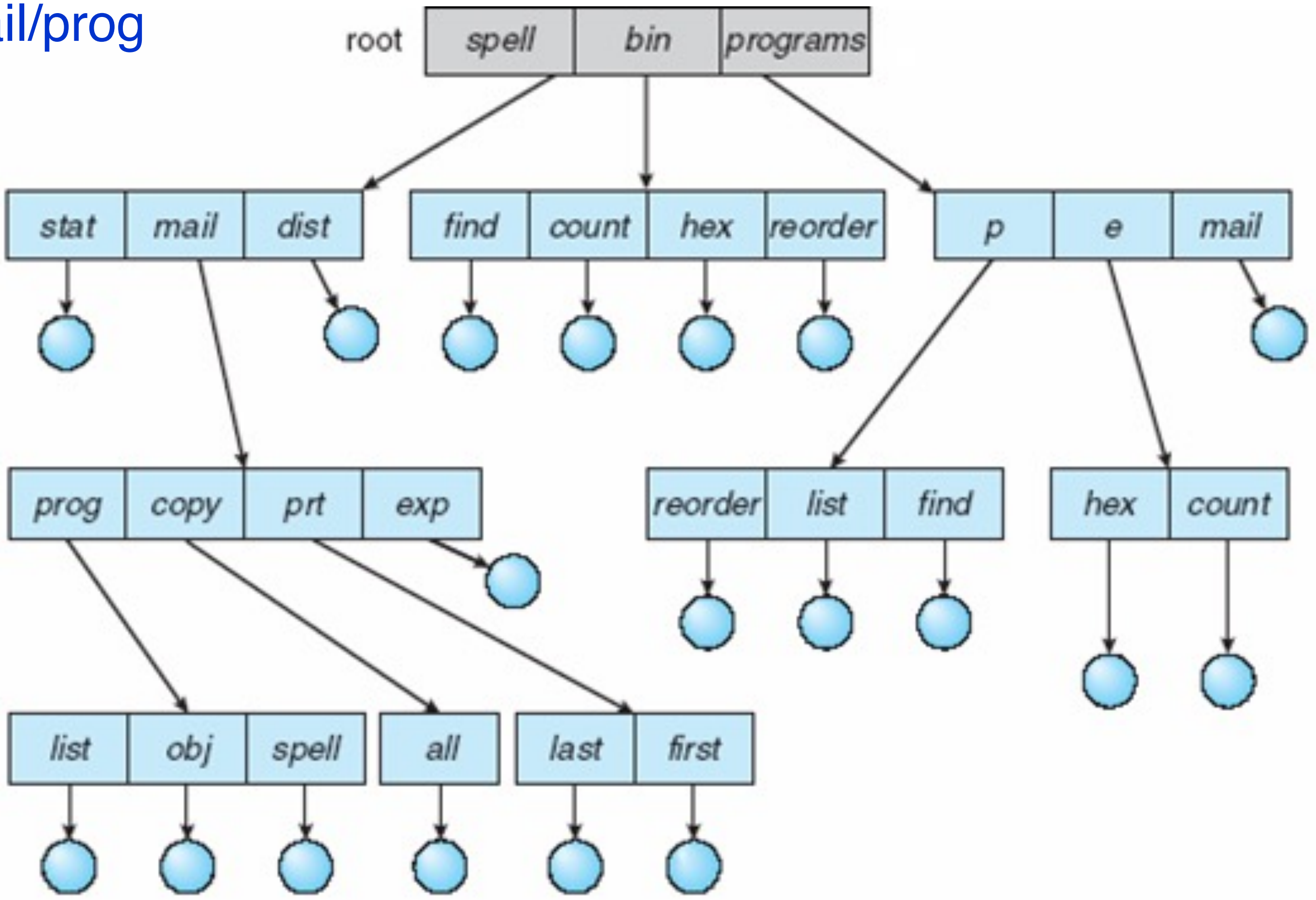
- Separate directory for each user



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

Tree-Structured Directories

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



Tree-Structured Directories (Cont)

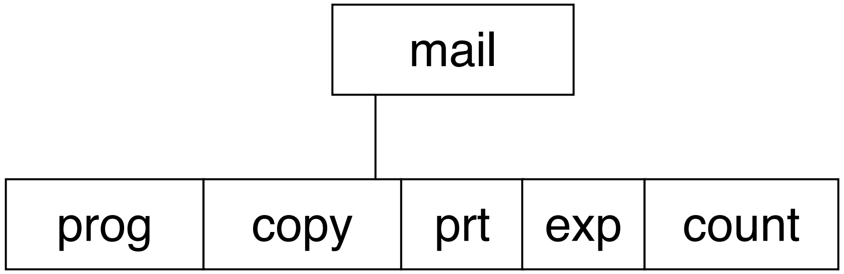
Absolute or **relative** path name

- 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

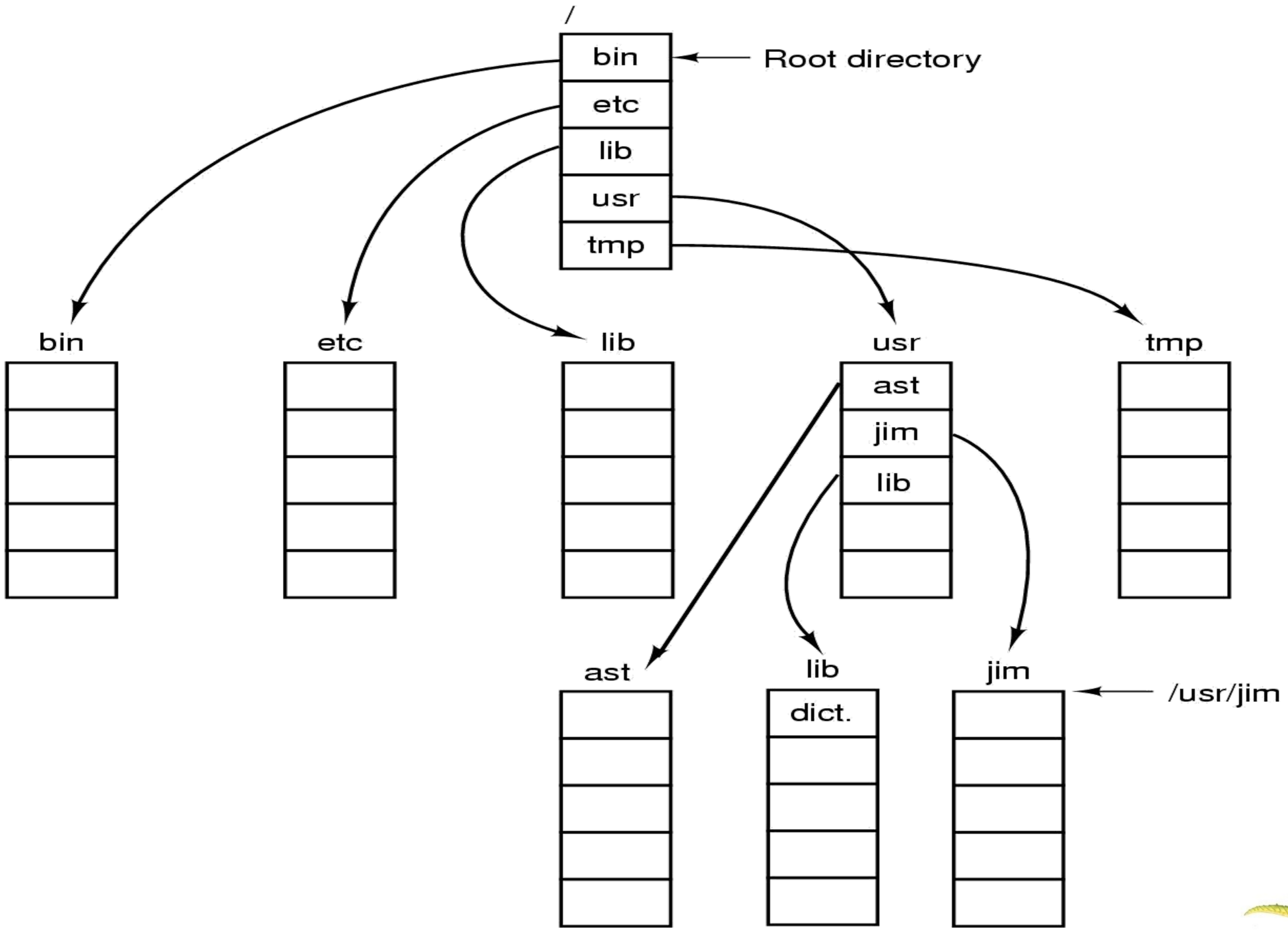
  mkdir count



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

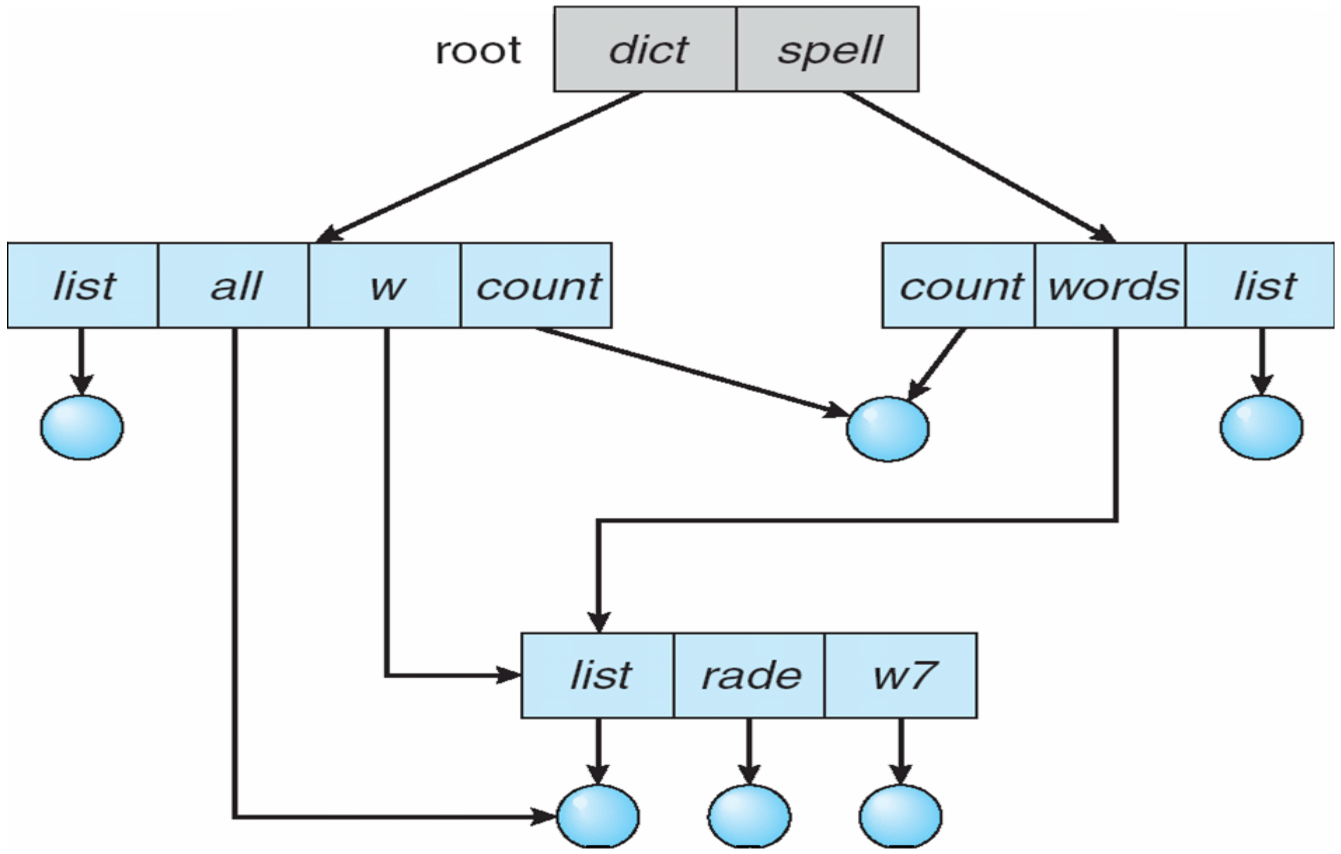
Path Names

A UNIX directory tree



Acyclic-Graph Directories (无循环)

- Have shared subdirectories and files



Acyclic-Graph Directories (Cont.)

- An acyclic-graph directory structure is more flexible than a simple tree structure, but it is also more complex
- Two different names (aliasing)
- Several problems:
  - A file may have multiple absolute path names.
  - File deletion: when can the space allocated to a shared file be deallocated and reused?

Acyclic-Graph Directories (Cont.)

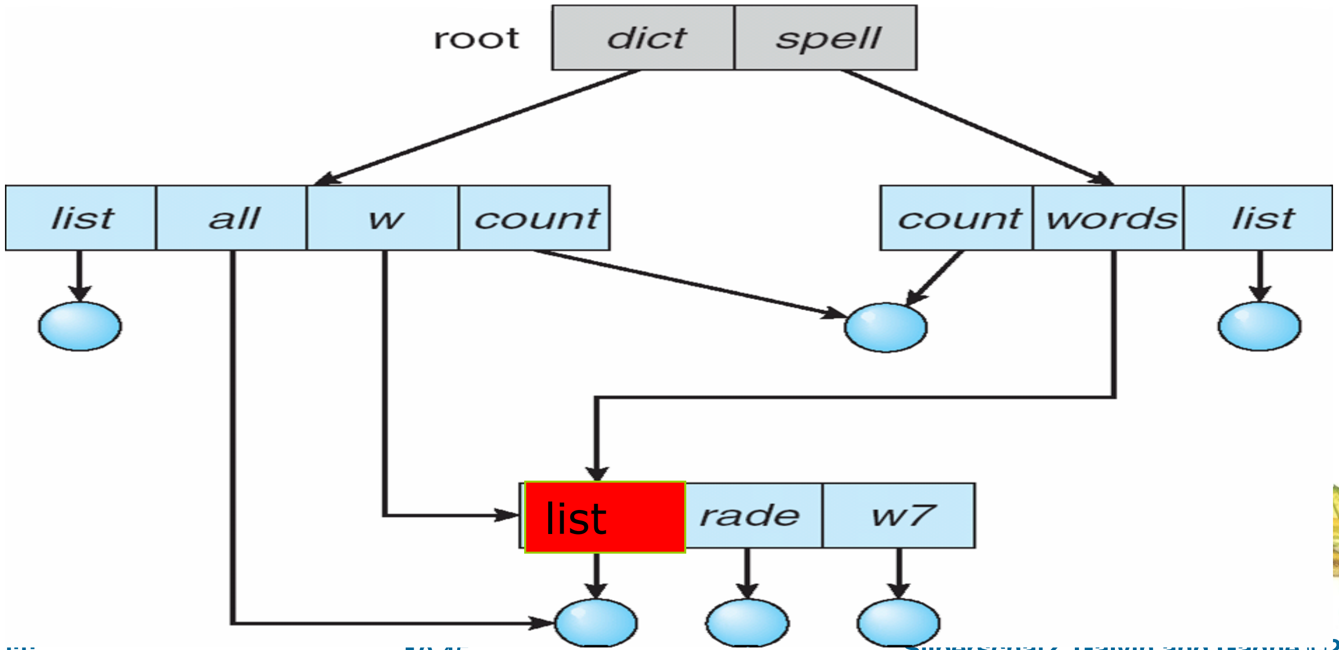
- If *dict* deletes *list* ⇒ dangling pointer

Solutions: 1.Keep backpointers of links for each file(保持后备指针);

2.Leave the link, and delete only when accessed later(断开链接);

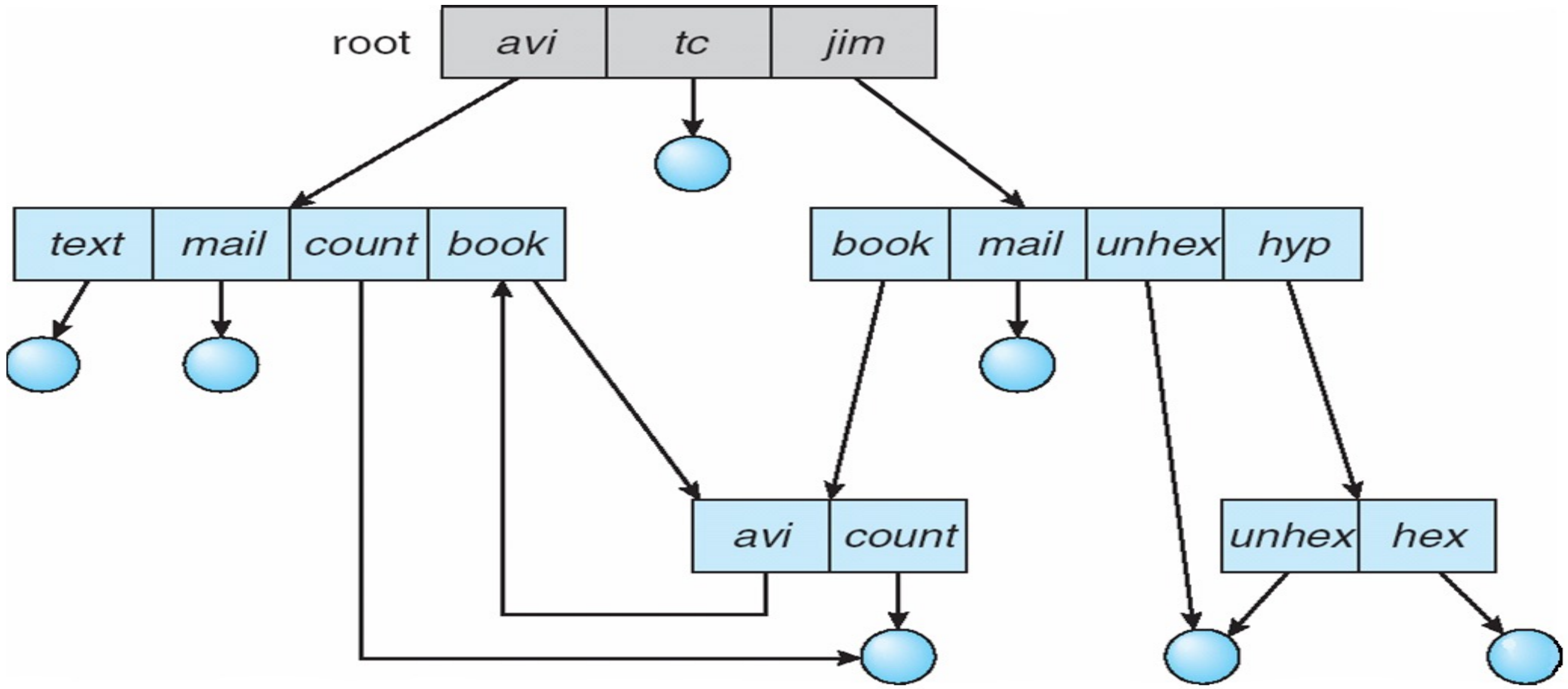
3.Keep reference count of each file(引用计数);

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



General Graph Directory

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK
  - Garbage collection is necessary only because of possible cycles in graph. extremely time consuming



File System Mounting

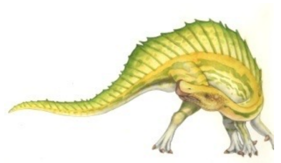
- A file system must be **mounted** before it can be accessed
- How to mount?

① The OS is given the name of the device and the mount point—the location within the directory structure where the file system is to be attached.

  - Typically, a mount point is an empty directory.

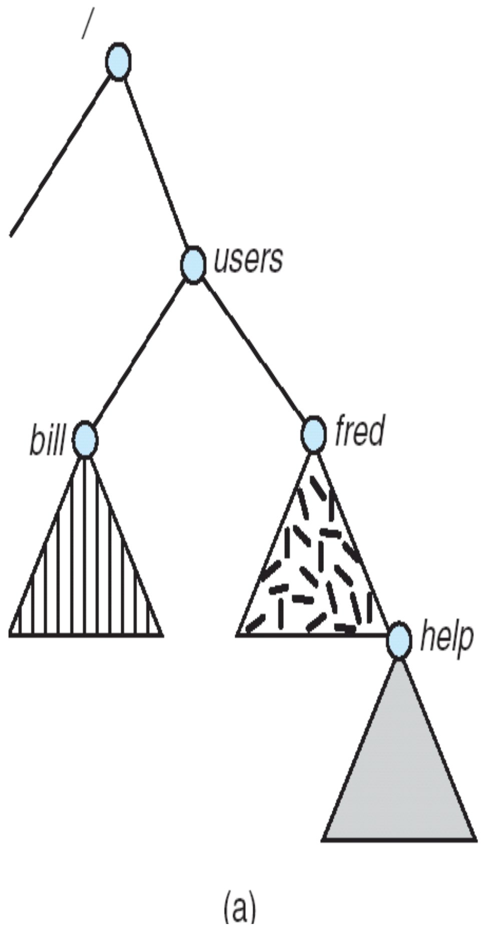
② Then, the OS verifies that the device contains a valid file system;

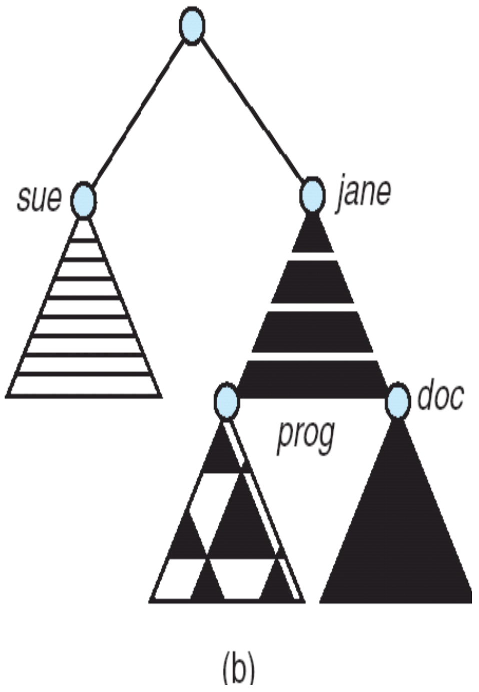
③ Finally, the OS notes in its directory structure that a file system is mounted at the specified mount point.

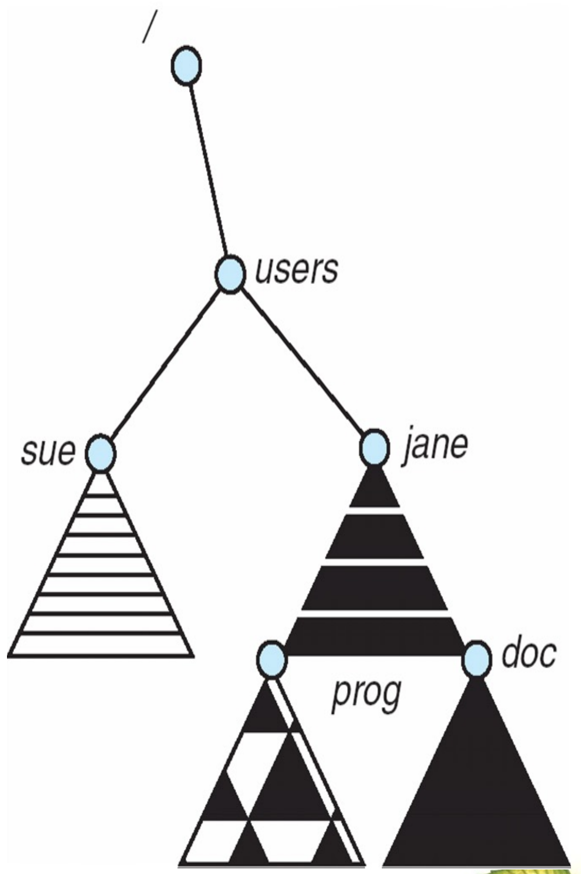


(a) Existing. (b) Unmounted Partition

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





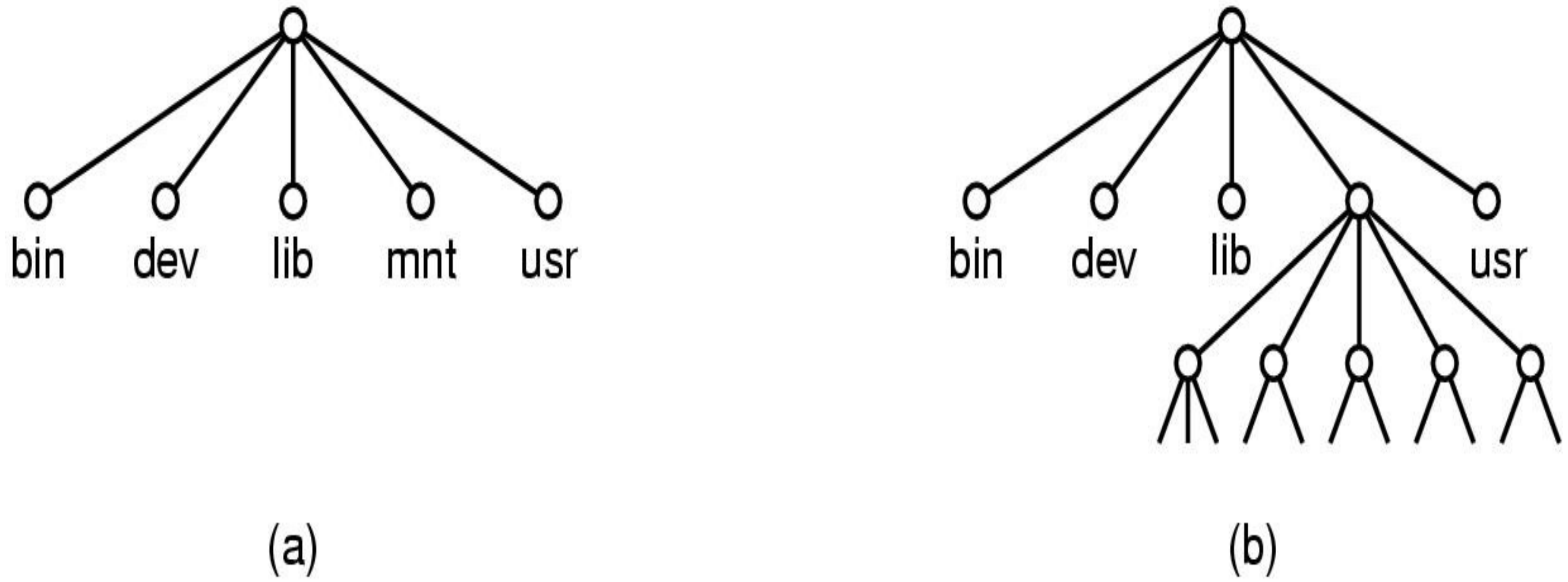


File System Mounting

- Mount allows two FSes to be merged into one

  For example you insert your floppy into the root FS

```
mount("/dev/fd0", "/mnt", 0)
```



49

File Sharing

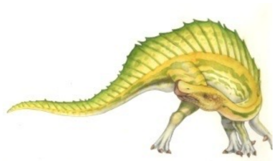
- Sharing of files on multi-user systems is desirable；
- Sharing may be done through a **protection** scheme；
- On distributed systems, files may be shared across a network；
- Network File System (NFS) is a common distributed file-sharing method；

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；

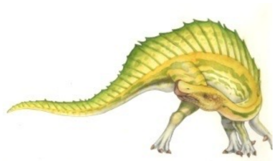
File Sharing – Remote File Systems

- 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**

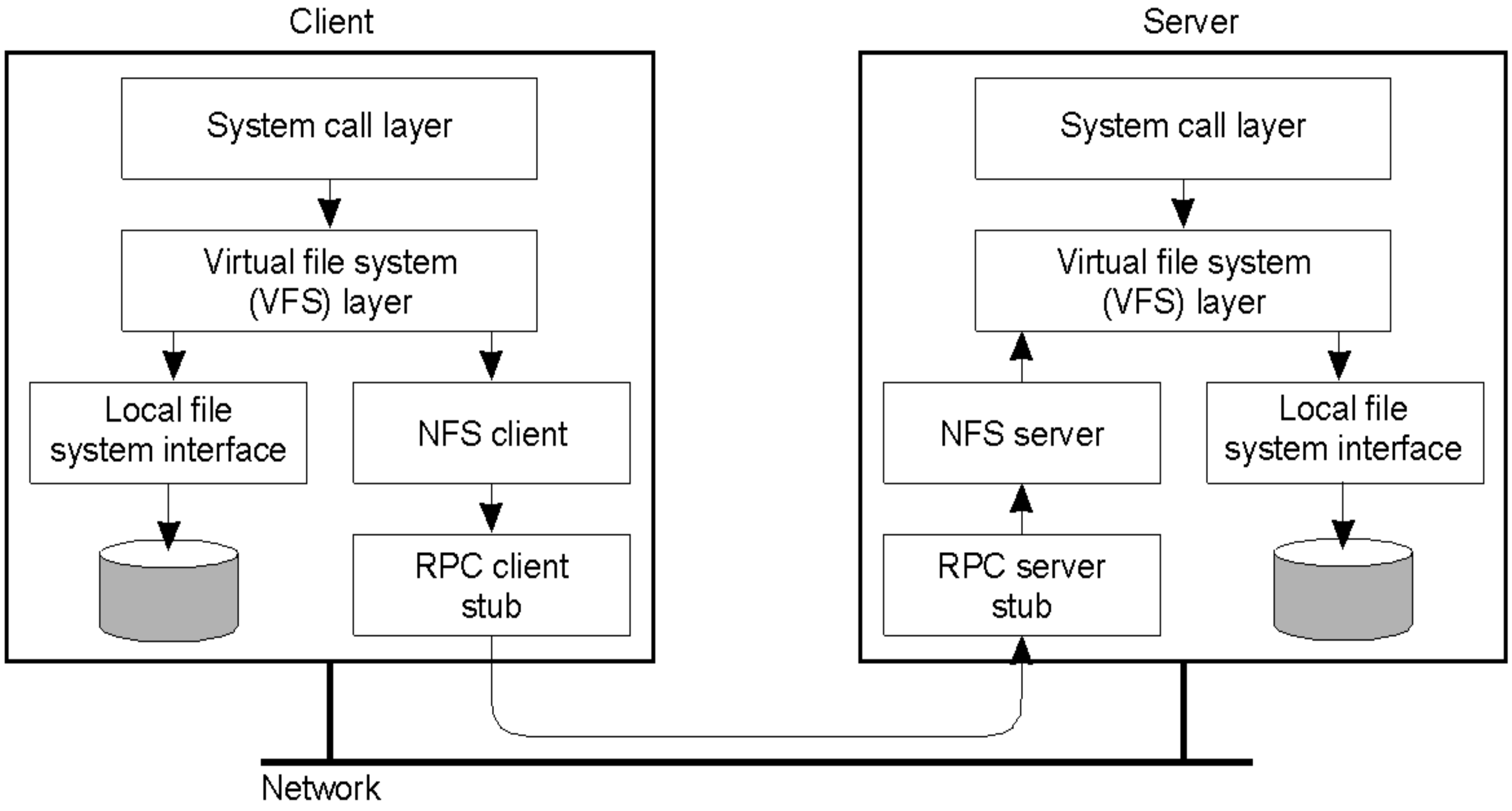


File Sharing – Remote File Systems

- **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** (Common Internet File System) 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 – Remote File Systems

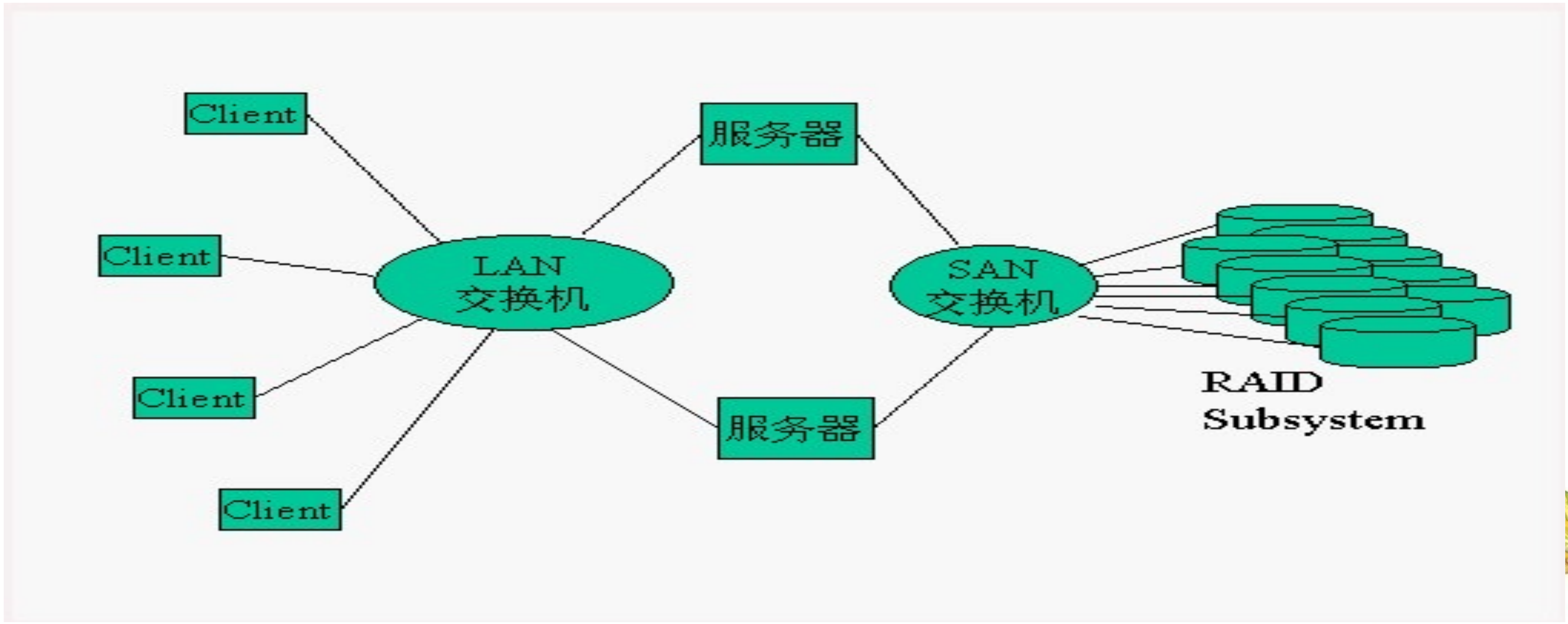


Client
Server
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

The basic NFS architecture for UNIX systems.

File Sharing

- Storage Area Network (SAN) is a current technology to provide large storage capacities to a large user population.
  - It is a high-speed network dedicated to the task of transporting data for storage and retrieval.
  - It connects together computers and storage devices to allow sharing of the pool of storage devices.
  - It is adopted by large institutions.



File Sharing

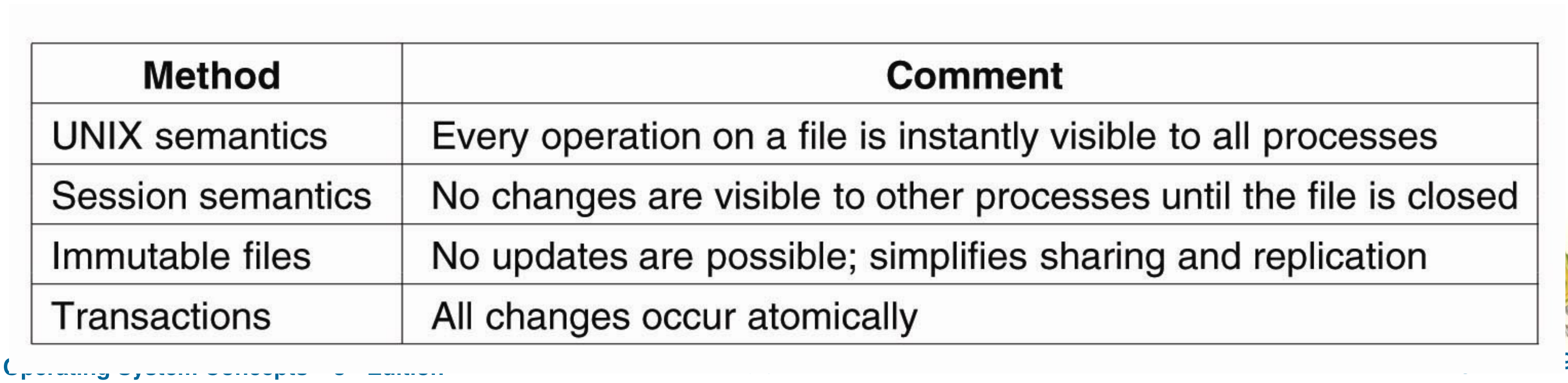
- Cloud storage is the newest technology to host data in a collection of nodes over the cloud.
  - Service is usually provided by third party, usually data center.
  - Common storage: iCloud, dropbox, Google drive.

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

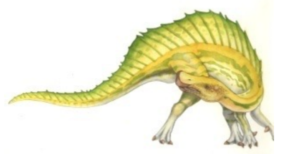
- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 7 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems）
  - Andrew File System (AFS) implemented complex remote file sharing semantics
    - 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



Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transactions	All changes occur atomically

Protection

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

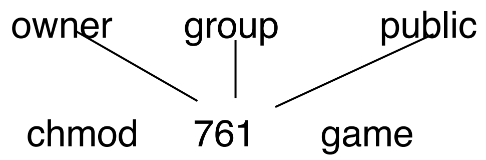


Access Lists and Groups

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

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

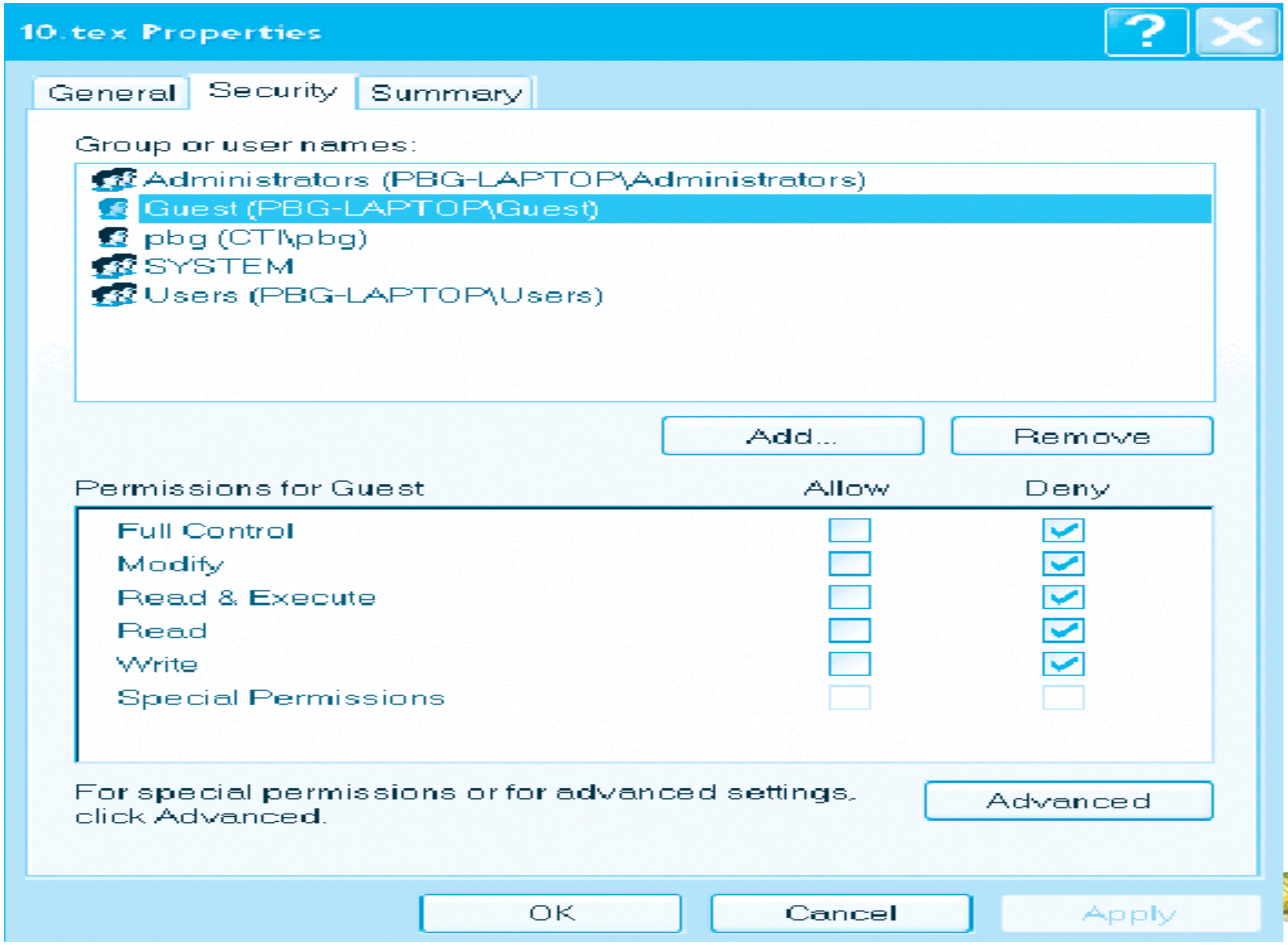
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

```
chgrp    G    game
```

Windows XP Access-control List Management



10.tex Properties

General | Security | Summary

Group or user names:

- Administrators (PBG-LAPTOP\Administrators)
- Guest (PBG-LAPTOP\Guest)
- pbg (CTI\pbg)
- SYSTEM
- Users (PBG-LAPTOP\Users)

Add... | Remove

Permissions for Guest	Allow	Deny
Full Control	☐	✓
Modify	☐	✓
Read & Execute	☐	✓
Read	☐	✓
Write	☐	✓
Special Permissions	☐	☐

For special permissions or for advanced settings, click Advanced.

Advanced

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/
```

assignment

1. 10.9
2. 10.11

End of Chapter 10