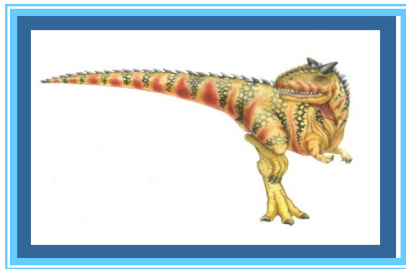




# Chapter 10: File-System Interface

---





# Chapter 10: File-System Interface

---

- 10.1 File Concept
- 10.2 Access Methods
- 10.3 Directory Structure
- 10.4 File-System Mounting
- 10.5 File Sharing
- 10.6 Protection





# Objectives

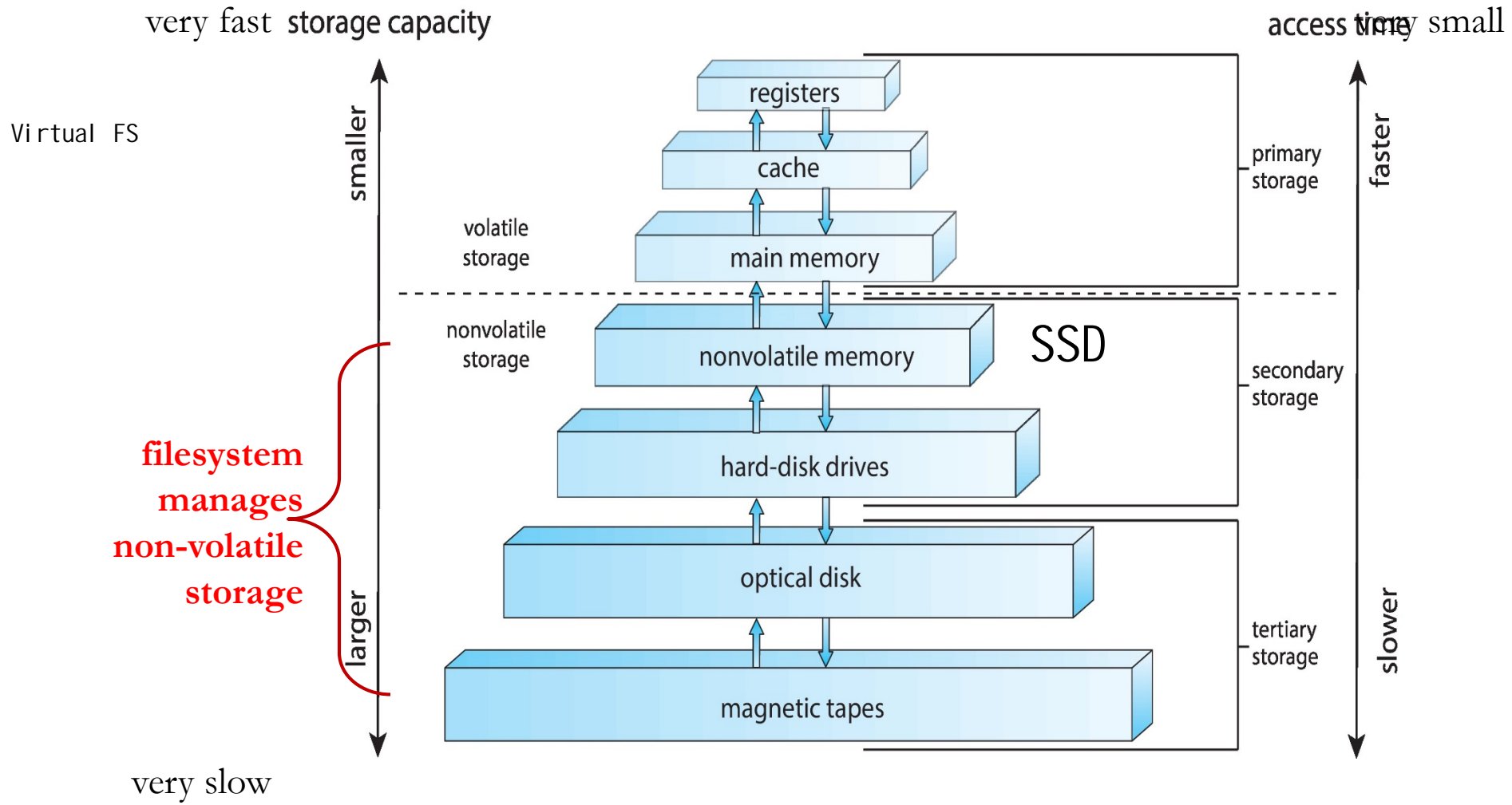
---

- 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





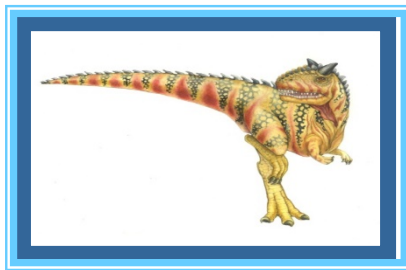
# Hierarchical Storage Architecture





# 10.1 File Concept

---





# File Concept (文件概念)

- **文件**是存储某种介质上的（如磁盘、光盘、SSD等）并具有文件名的一组相关信息的集合
- A file is a sequence of bytes stored on some device



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





# File Attributes (文件属性)

- **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 (文件操作)

- File is an **abstract data type** (抽象数据类型)
  - **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 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 closes it
  - **Disk location of the file**: cache of data access information
  - **Access rights**: per-process access mode information





# File Types (文件类型)

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 (文件内部结构)

## ■ **None** - sequence of words, bytes (流文件结构)

内容组织

## ■ **Simple record structure** (记录文件结构)

- Lines
- Fixed length
- Variable length

<https://www.csdn.net/tags/Mtjagg5sNjE10DYtYmxvZw0000000000.html>

## ■ **Complex Structures**

- Formatted document icon, location, ...
- Relocatable load file

## ■ Can simulate last two with first method by inserting appropriate control characters

## ■ **Who decides:**

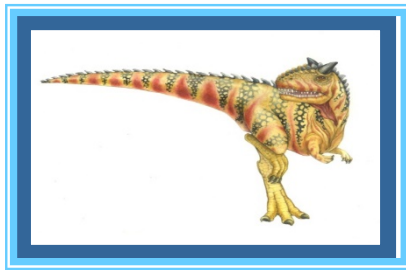
- Operating system
- Program





## 10.2 Access Methods ( 文件的访问方法 )

---





# Access Methods

## ■ Sequential Access (顺序存取) **Fig 10.2**

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

## ■ Direct Access (直接存取) **Fig 10.3**

read  $n$   
write  $n$   
position to  $n$   
    read next  
    write next  
rewrite  $n$

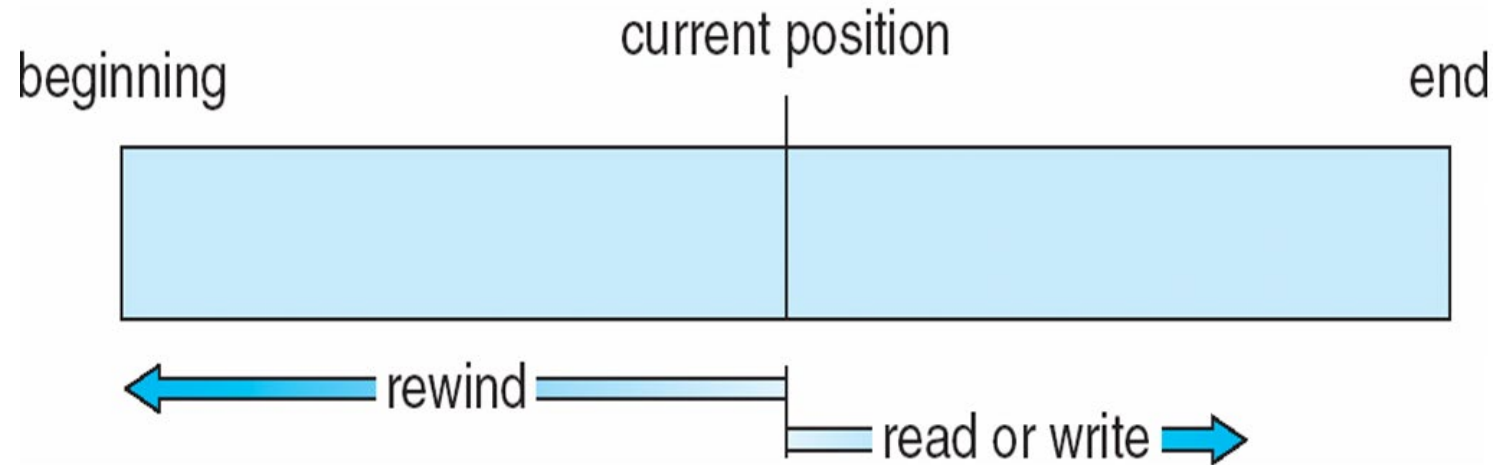
$n$  = relative block number **Fig 10.4**

## ■ Indexed sequential-access (索引顺序)





## Fig 10.2 Sequential-access File





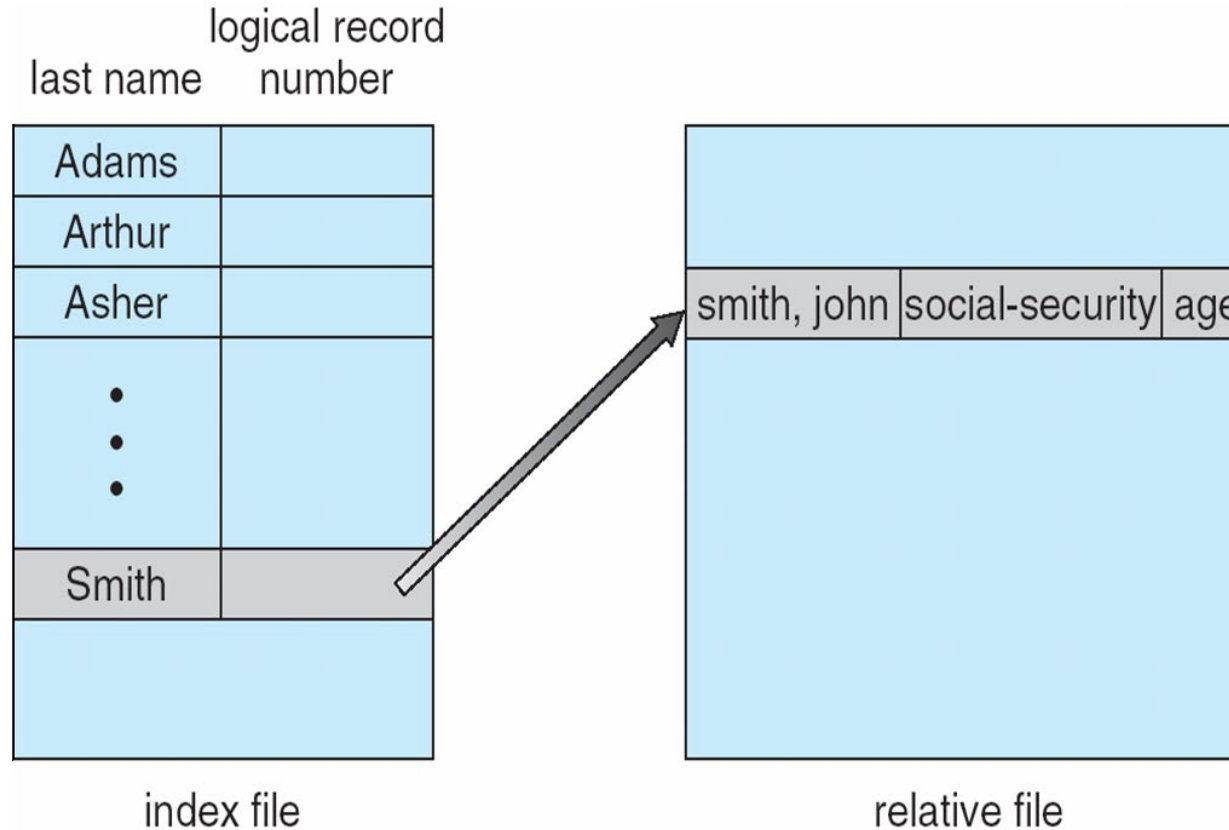
## Fig 10.3 Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp + 1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp + 1;</i>





## Fig 10.4 Example of Index and Relative Files

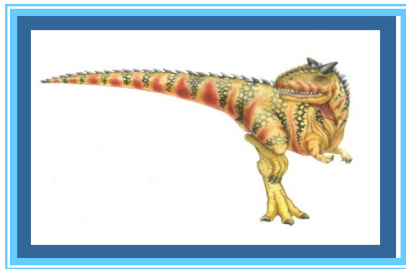






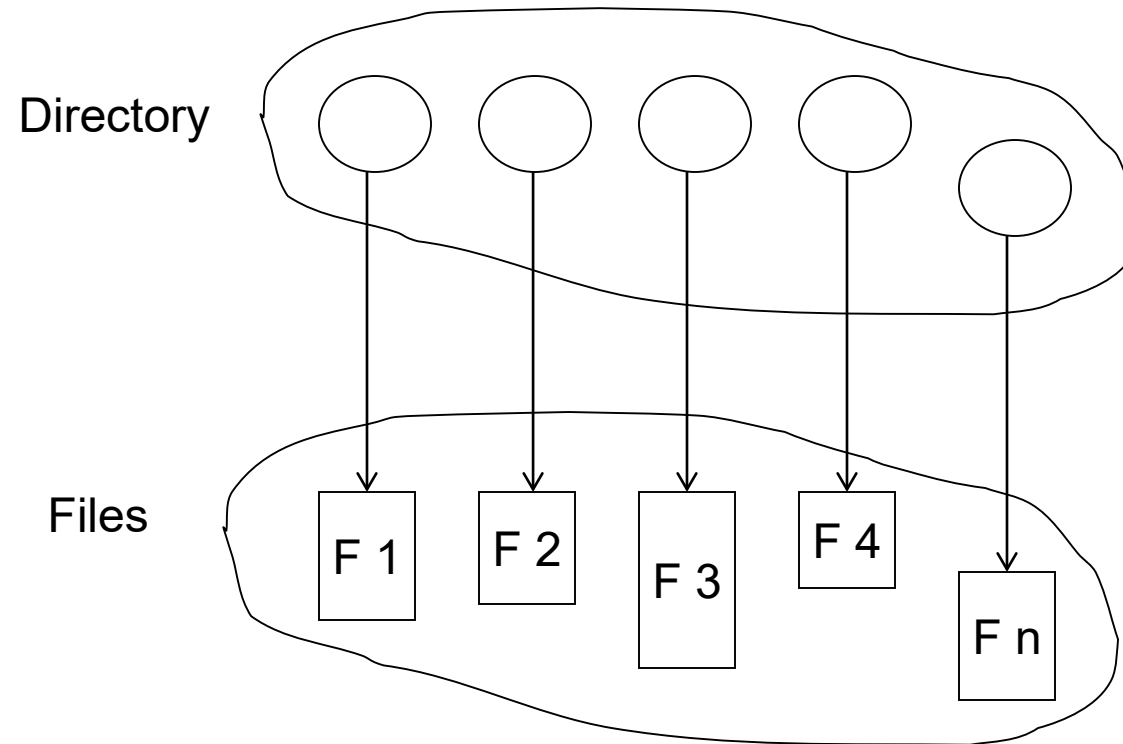
## 10.3 Directory Structure (目录结构)

---



# 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





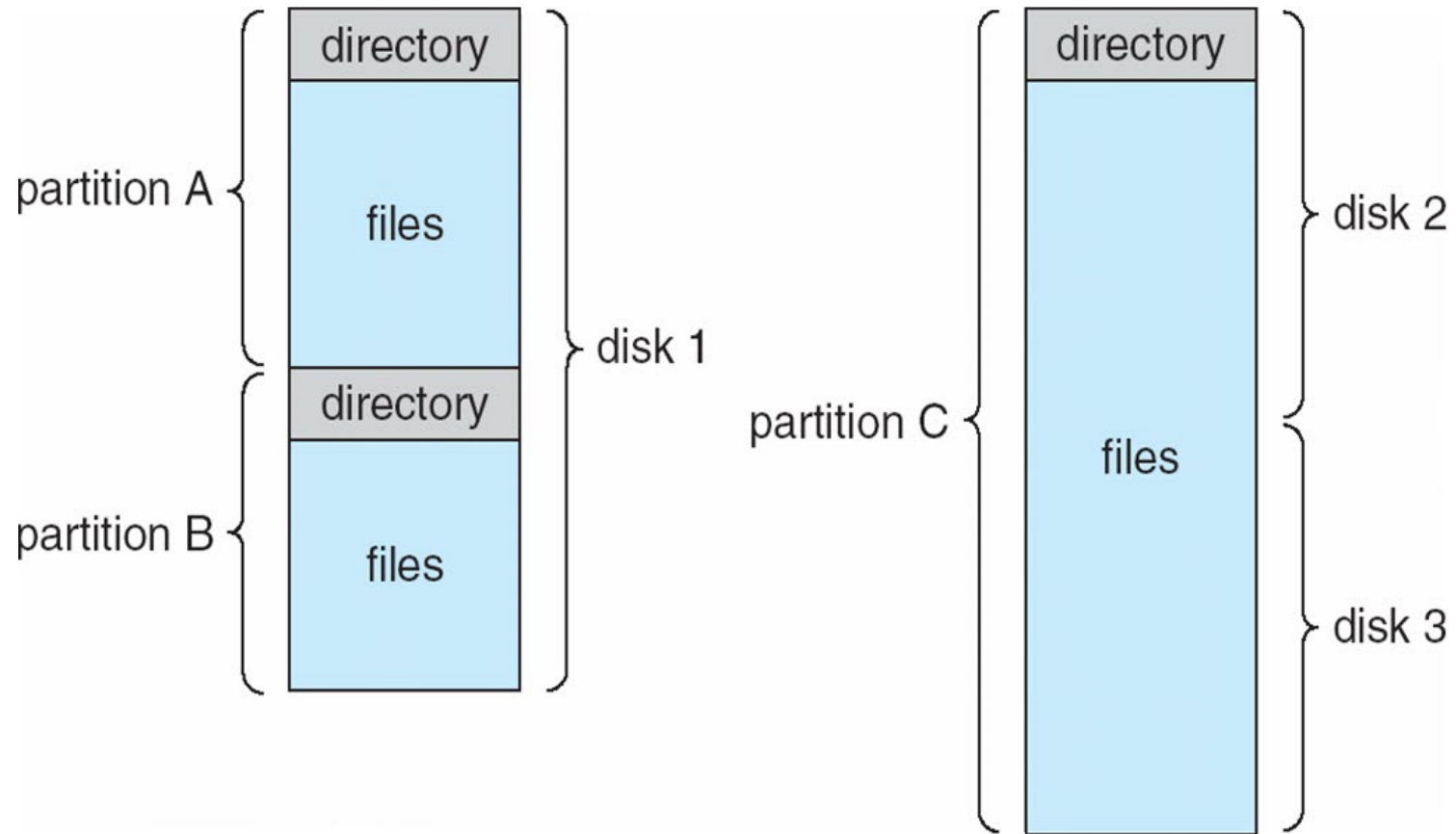
# Disk Structure

- Disk can be subdivided into **partitions** (分区)
- 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





# Operations Performed on Directory

---

- ✓ Search for a file
- ✓ Create a file
- ✓ Delete a file
- ✓ List a directory
- ✓ Rename a file
- ✓ Traverse the file system (遍历文件系统)





# Organize the Directory (Logically) to Obtain

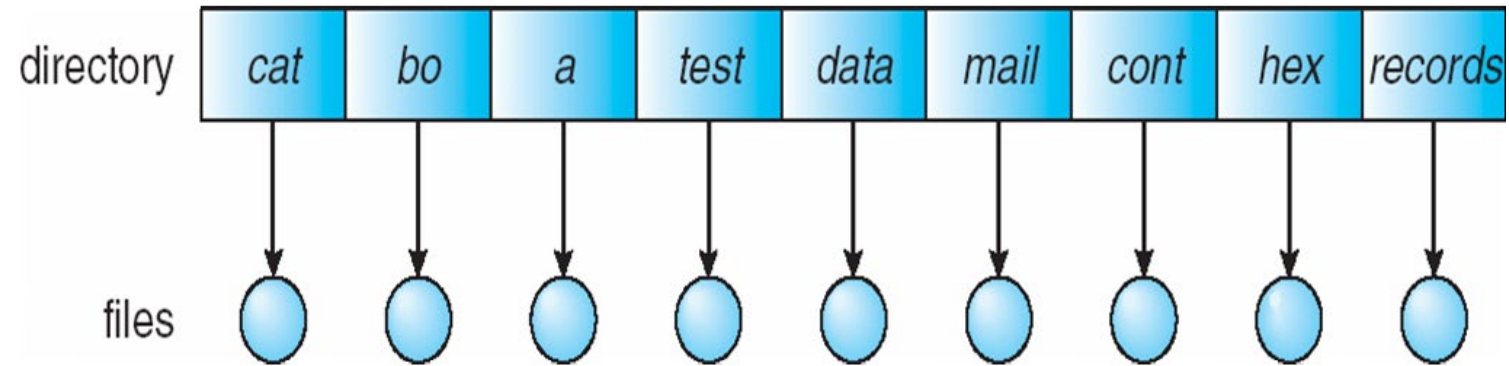
- **Efficiency** – locating a file quickly
- **Naming (重名)** – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- **Grouping (分组)** – logical grouping of files by properties, (e.g., all Java programs, all games, ...)





# Single-Level Directory (单级目录)

- A single directory for all users



Naming problem

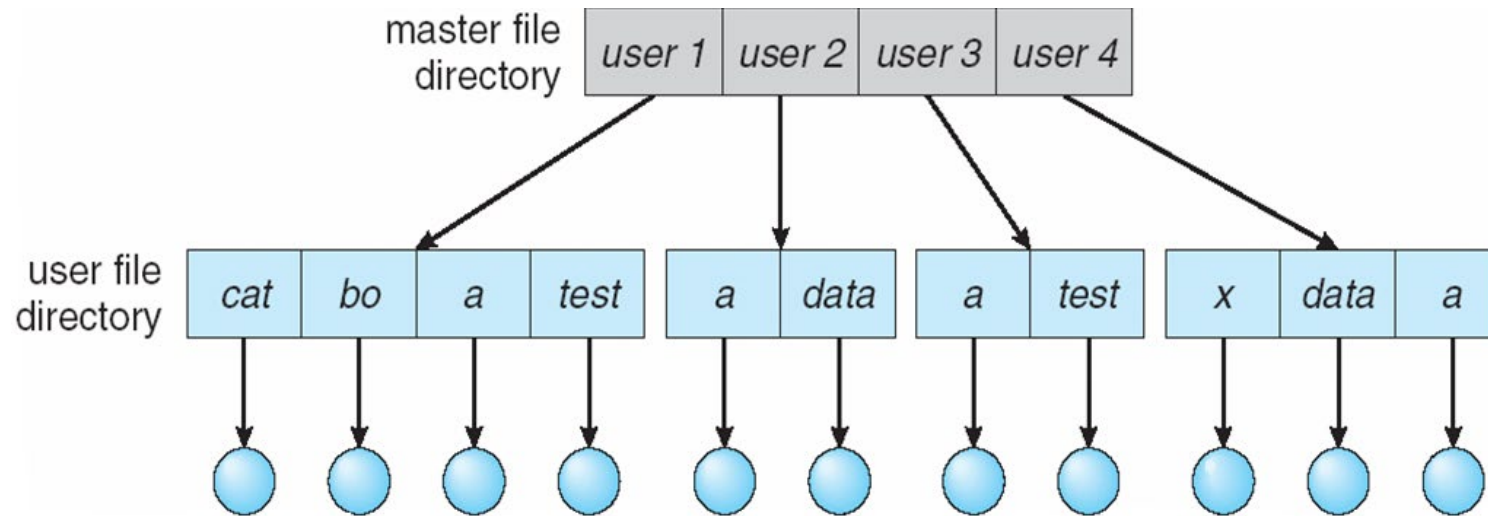
Grouping problem





# Two-Level Directory (二级目录)

## ■ Separate directory for each user



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









# 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 绝对路径、相对路径

- Creating a new file is done in current directory

- Delete a file

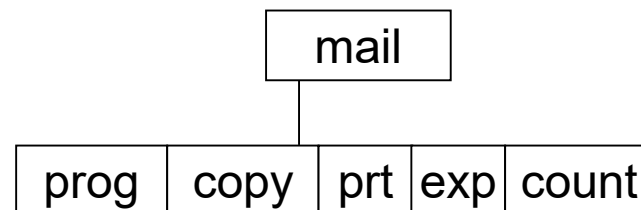
`rm <file-name>`

- Creating a new subdirectory is done in current directory

`mkdir <dir-name>`

Example: if in current directory `/mail`

`mkdir count`



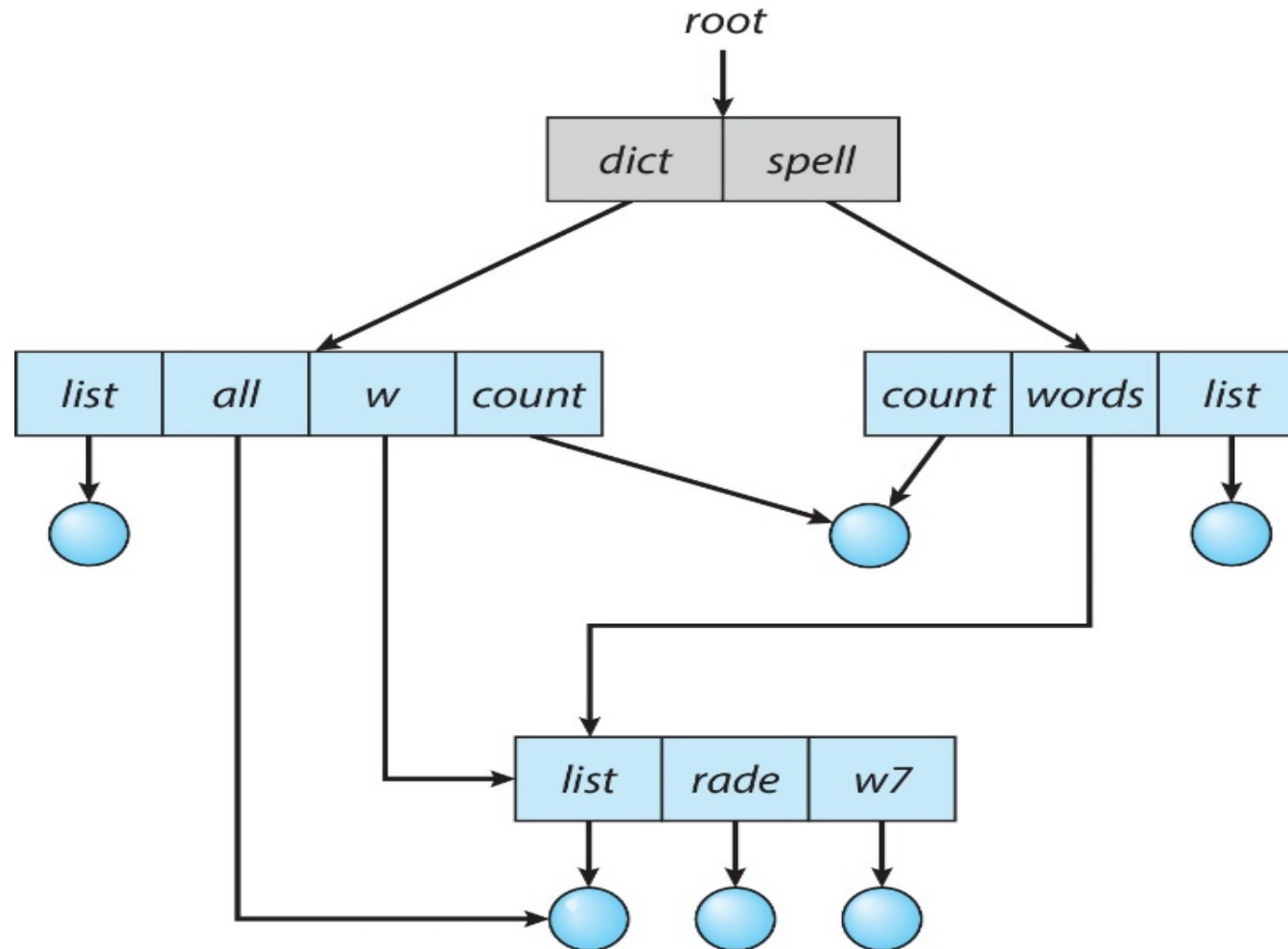
Deleting “mail”  $\Rightarrow$  deleting the entire subtree rooted by “mail”





# Acyclic-Graph Directories 无环图结构目录

- Have **shared** subdirectories and files





# Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *all*  $\Rightarrow$  dangling pointer

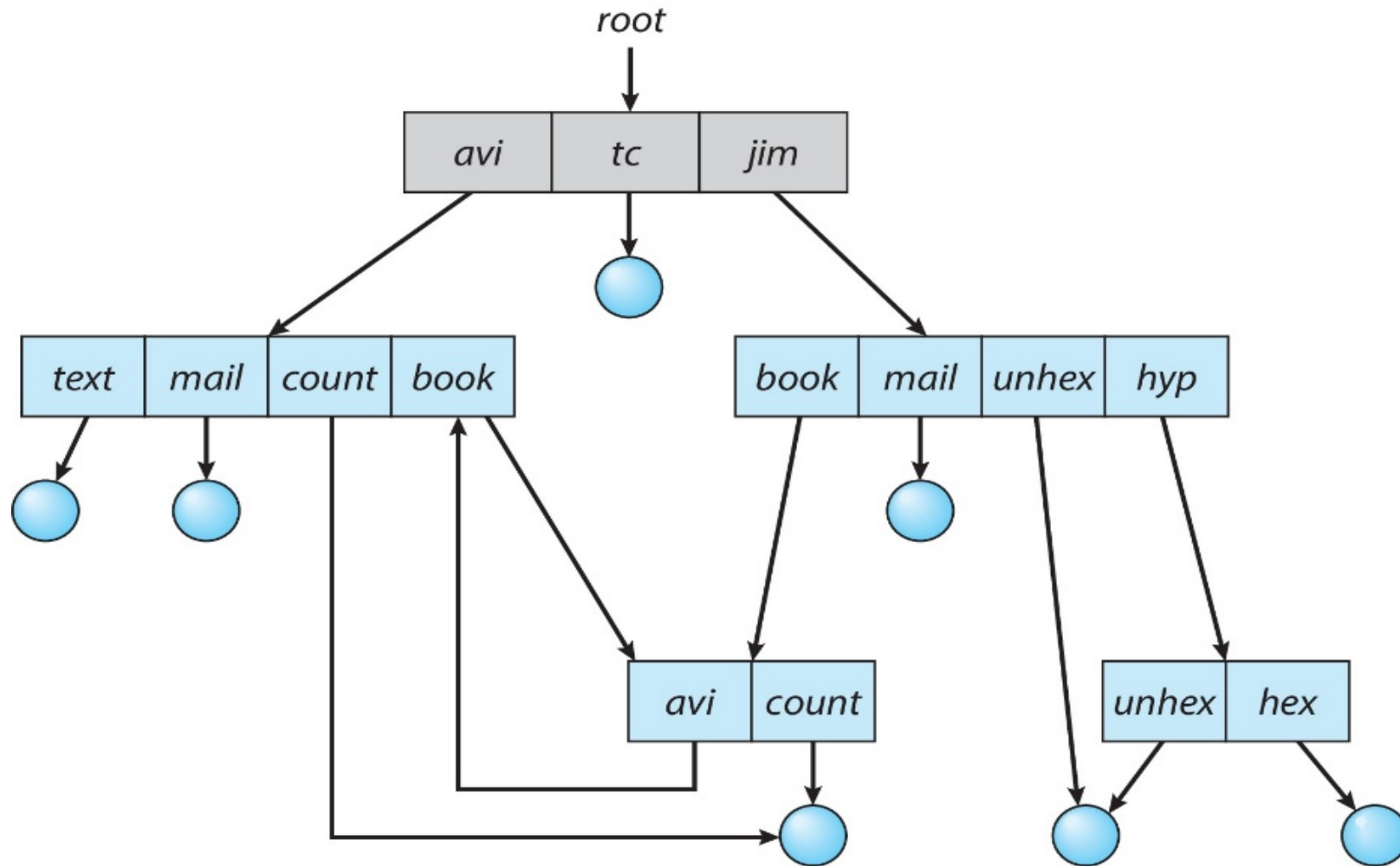
Solutions:

- Backpointers (逆向指针), so we can delete all pointers  
Variable size records a problem
- Backpointers using a daisy chain organization
- Entry-hold-count solution (表项保留计数)
  - ▶ **unix、linux:hard links**
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file





# General Graph Directory (普通图结构目录)





# General Graph Directory (Cont.)

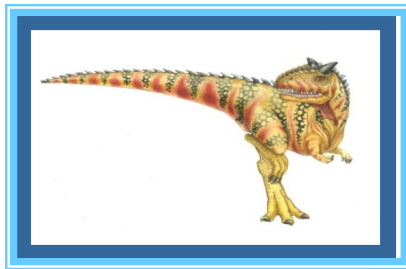
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm (环检测算法) to determine whether it is OK





# 10.4 File System Mounting

---



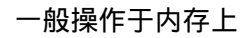




# File System Mounting

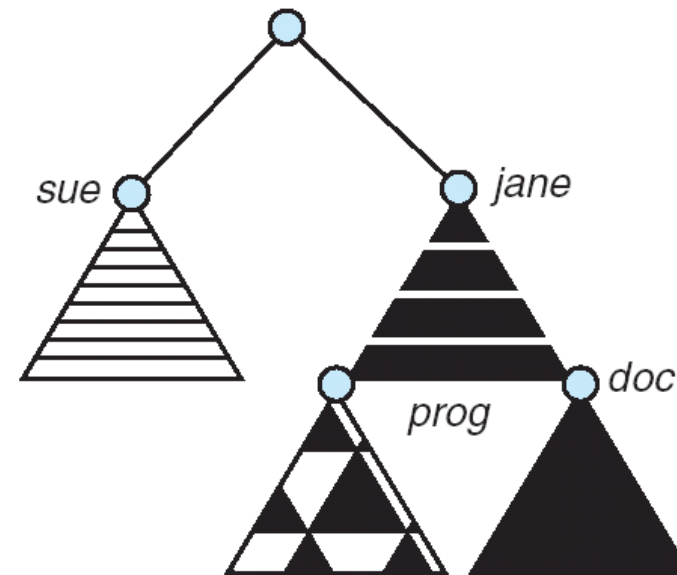
- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**





(a)

外存上没有就丢了

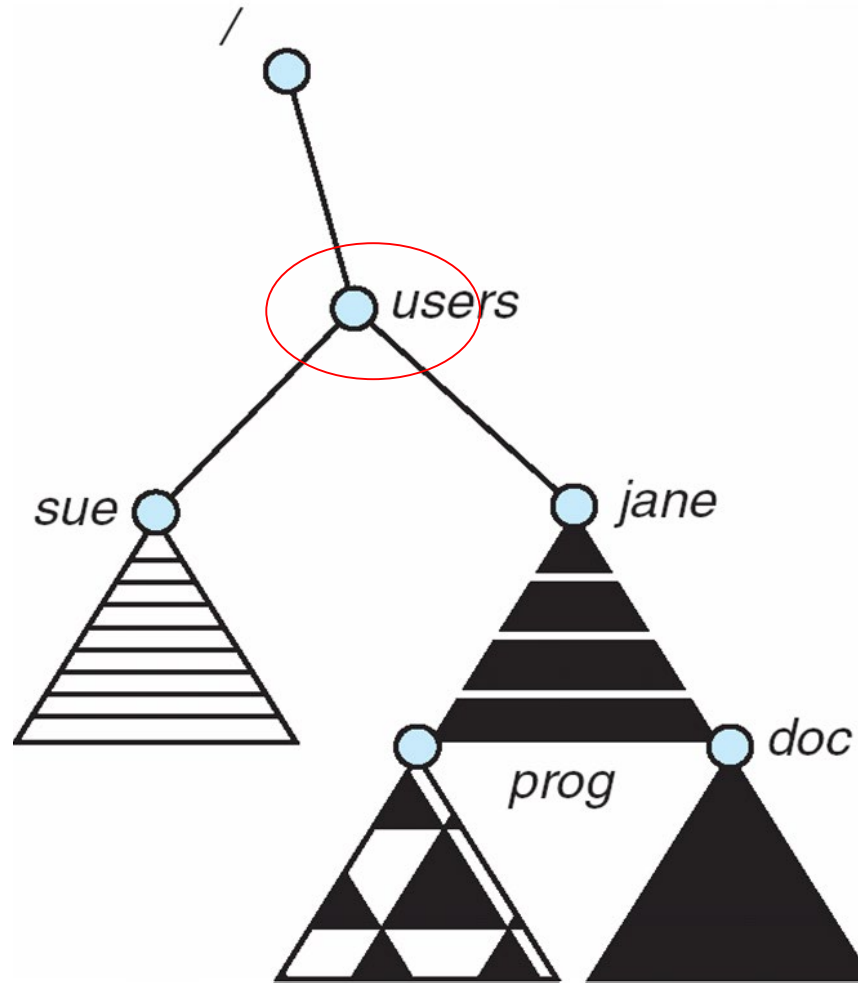


(b)





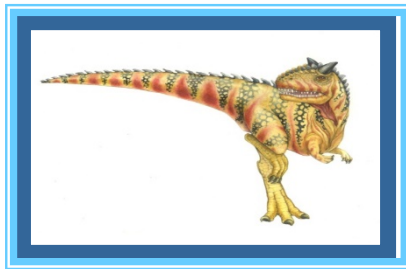
# Mount Point





# 10.5 File Sharing

---





# 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

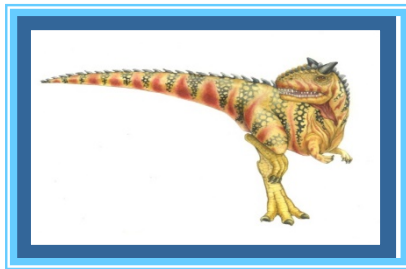
- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 7 process synchronization algorithms
    - ▶ Tend to be less complex due to disk I/O and network latency (for remote file systems)
  - Andrew File System (AFS) implemented complex remote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





# 10.6 Protection

---





# Protection

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





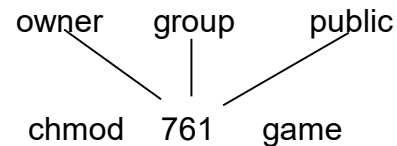
# Access Lists and Groups

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

Unix, Linux

a) owner access	7	⇒	RWX 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.



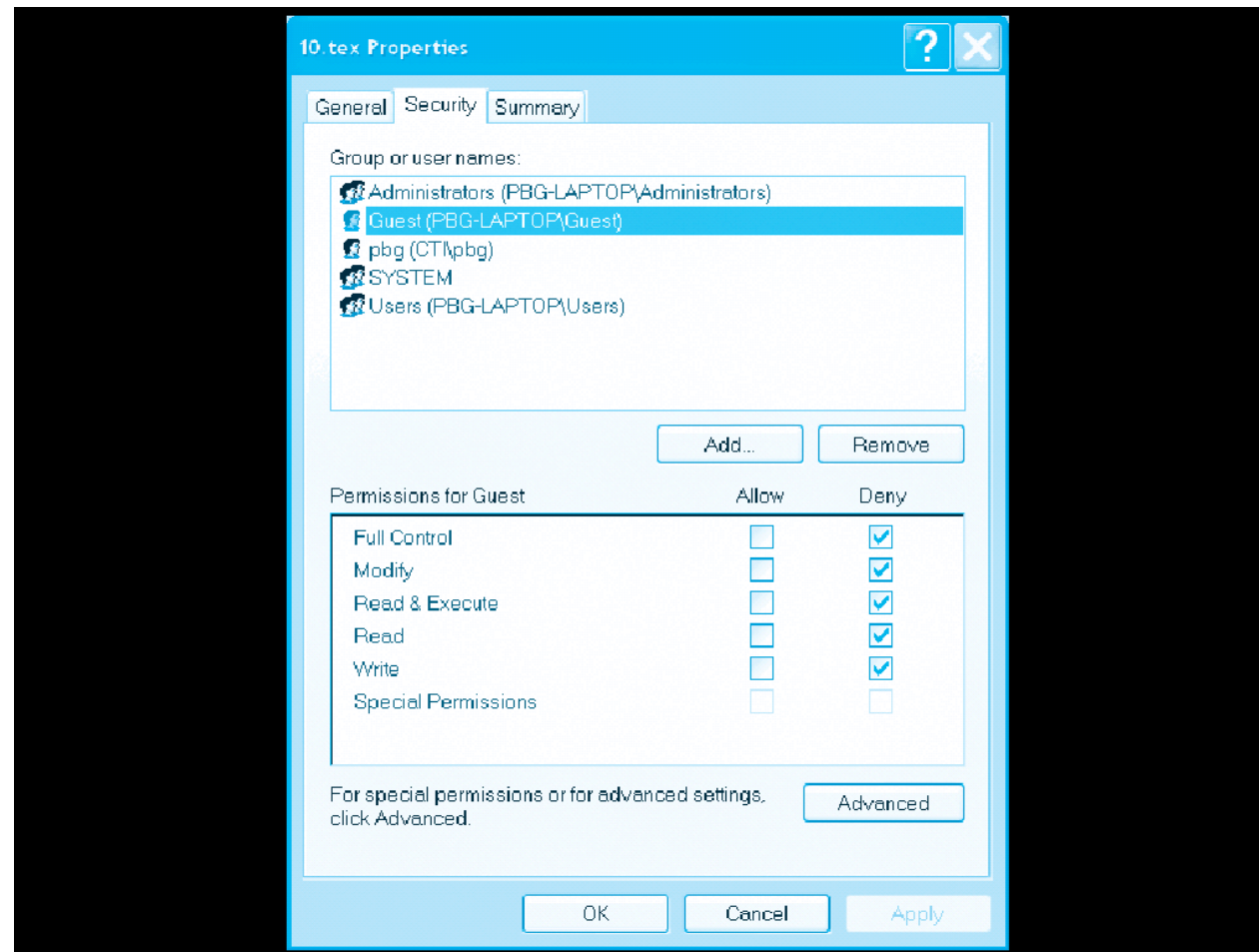
Attach a group to a file:

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/
```





# Homework

---

## ■ 学在浙大





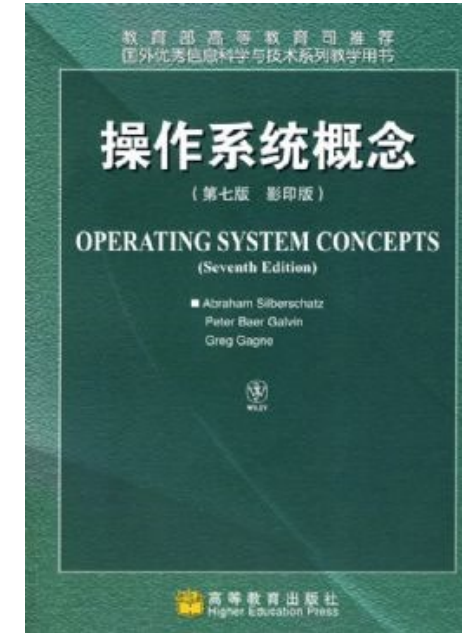
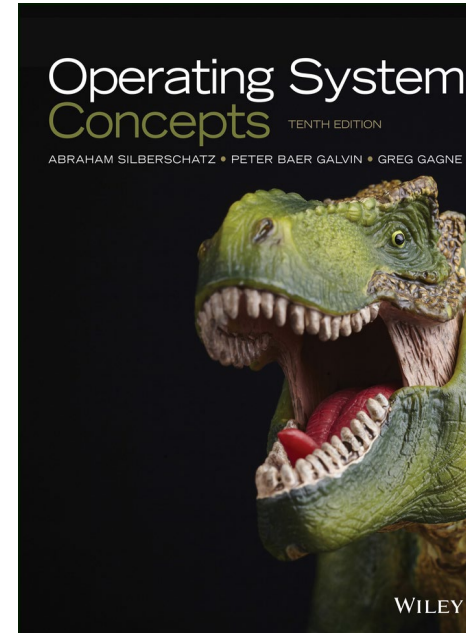
# Reading Assignments

## ■ Read for this week:

- **Chapters 10**  
of the text book:

## ■ Read for next week:

- **Chapters 11**  
of the text book:







# End of Chapter 10

---

