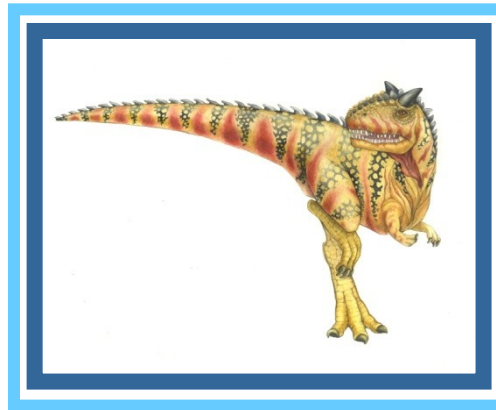


# Chapter 10:

# File-System Interface

---





# File System

---

- 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.
- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- Commonly, files represent programs (both source and object forms) and data.





# File Concept

---

- Contiguous logical address space
- Types:
  - Data
    - ▶ numeric
    - ▶ character
    - ▶ binary
  - Program
- In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user





- The information in a file is defined by its creator. Many different types of information may be stored in a file—source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.
- A **text file** is a sequence of characters organized into lines (and possibly pages).
- A **source file** is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An **object file** is a sequence of bytes organized into blocks understandable by the system's linker.
- An **executable file** is a series of code sections that the loader can bring into memory and execute.





# 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





- **Create** --- Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Write** --- we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Read** ---we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a *read* pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.







- **Reposition within file ---** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.
- **Deleting a file ----** We search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file ---** The user may want to erase the contents of a file but keep its attributes.





# Open Files

---

Several pieces of data are needed to manage open files:

- File pointer: pointer to last read/write location
- File-open count: counter of number of times a file is open
- Disk location of the file: cache of data access information. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
- Access rights: per-process access mode information. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.
- File locks allow one process to lock a file and prevent other processes from gaining access to it. File locks are useful for files that are shared by several processes.





# 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. If a lock is mandatory, then once a process acquires an exclusive lock, the operating system will prevent any other process from accessing the locked file.
- **Advisory** – processes can find status of locks and decide what to do.  
if the lock is advisory, then the operating system will not prevent any other process from accessing the locked file.





# 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





# Access Methods

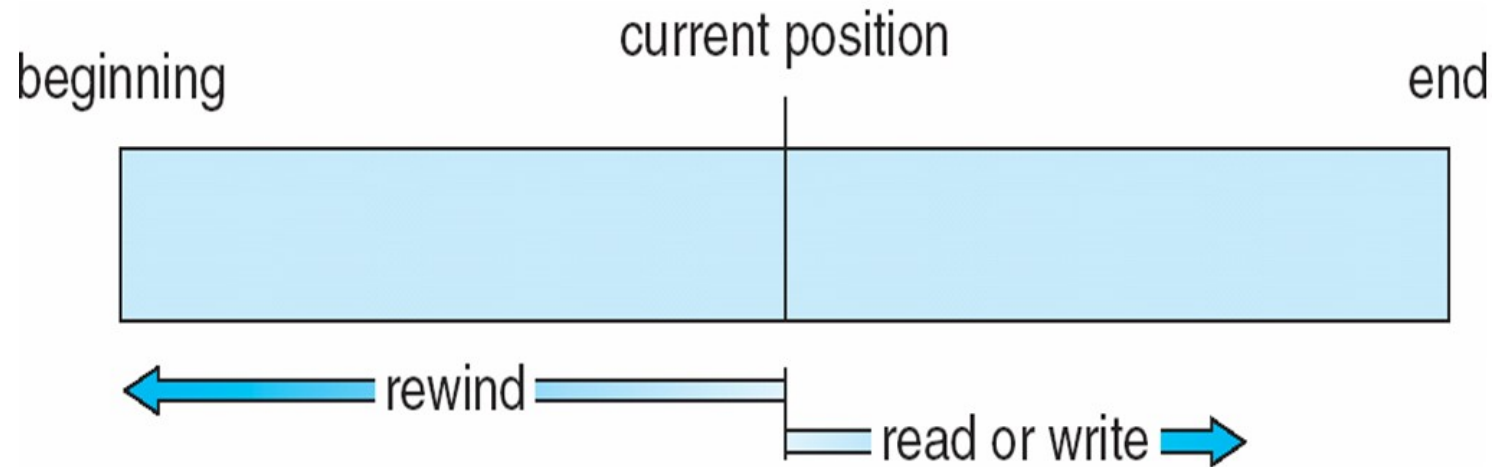
- **Sequential Access:** Information in the file is processed in order, one record after the other
  - read next
  - write next
  - reset
  - no read after last write  
(rewrite)
- A read operation-**read *next***- reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- The write *operation-write next*-appends to the end of the file and advances to the end of the newly written material (the new end of file).
- **Direct Access :** A file is made up of fixed length of logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
  - read  $n$
  - write  $n$
  - position to  $n$
  - read next
  - write next
  - rewrite  $n$

$n$  = relative block number





# Sequential-access File





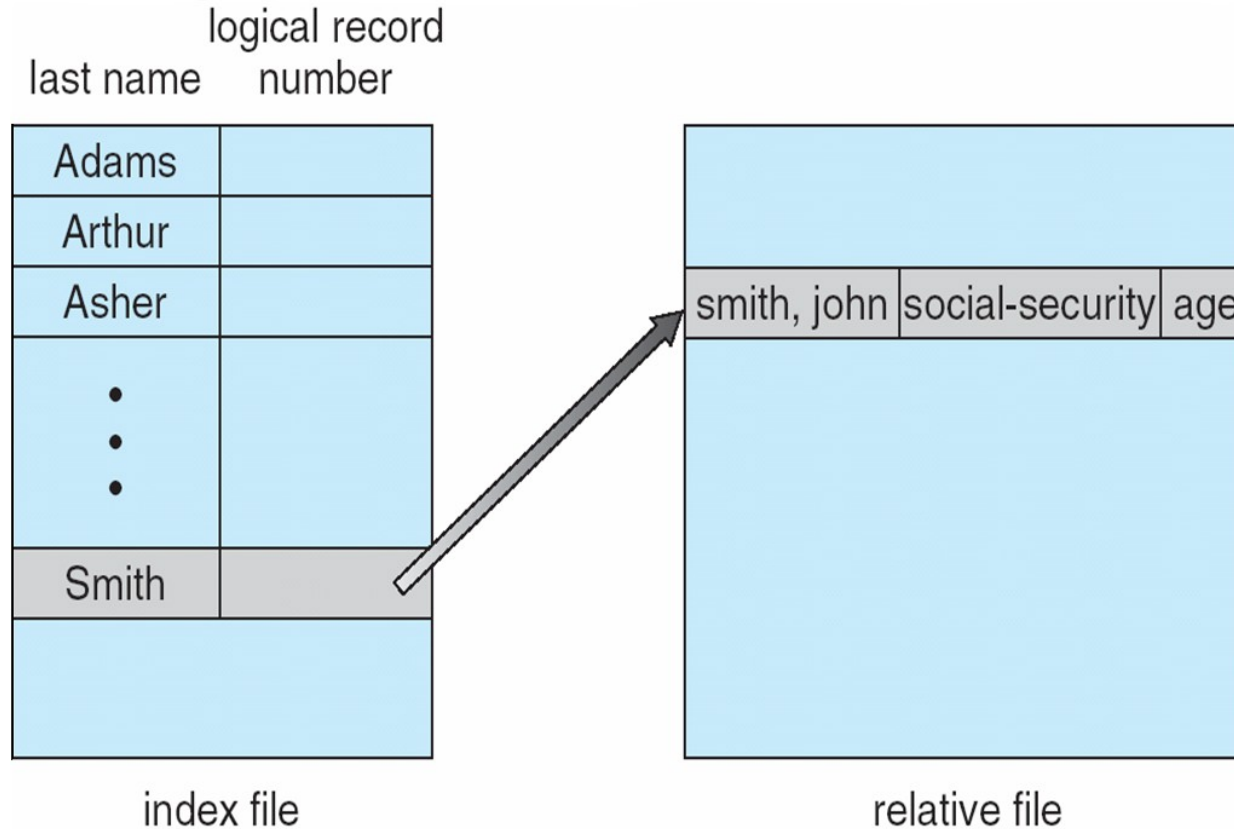
# Simulation of Sequential Access on Direct-access File

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





# Example of Index and Relative Files

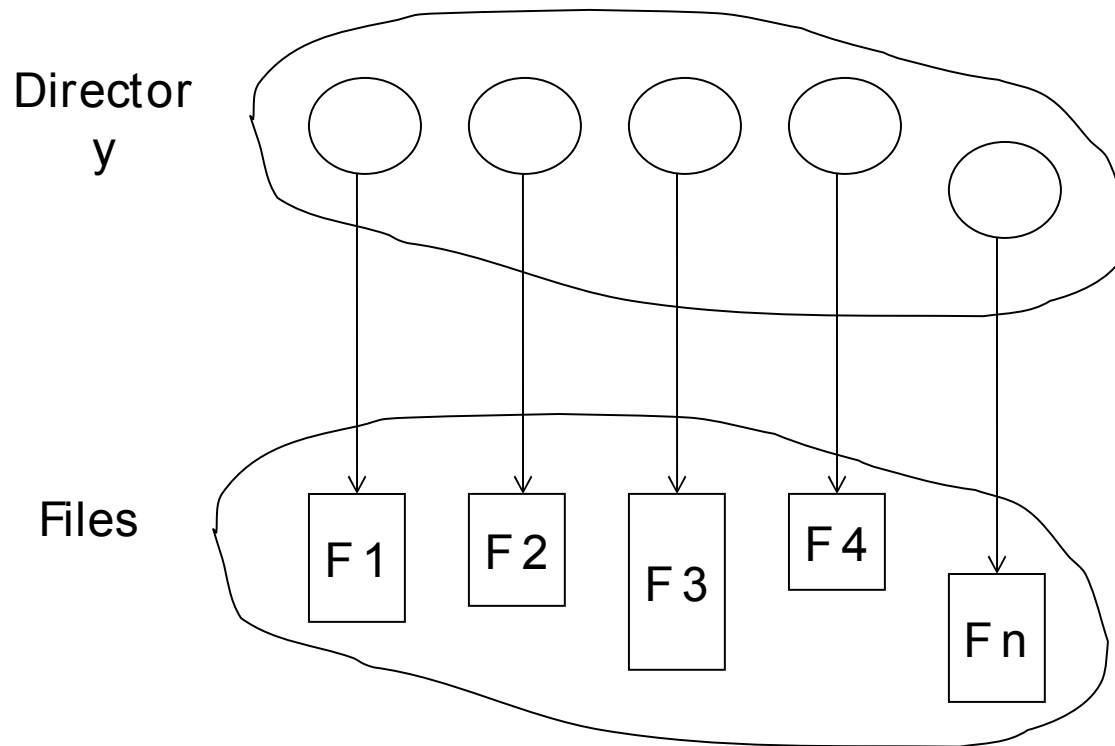






# Directory Structure

- A collection of nodes containing information about all files



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





# Disk Structure

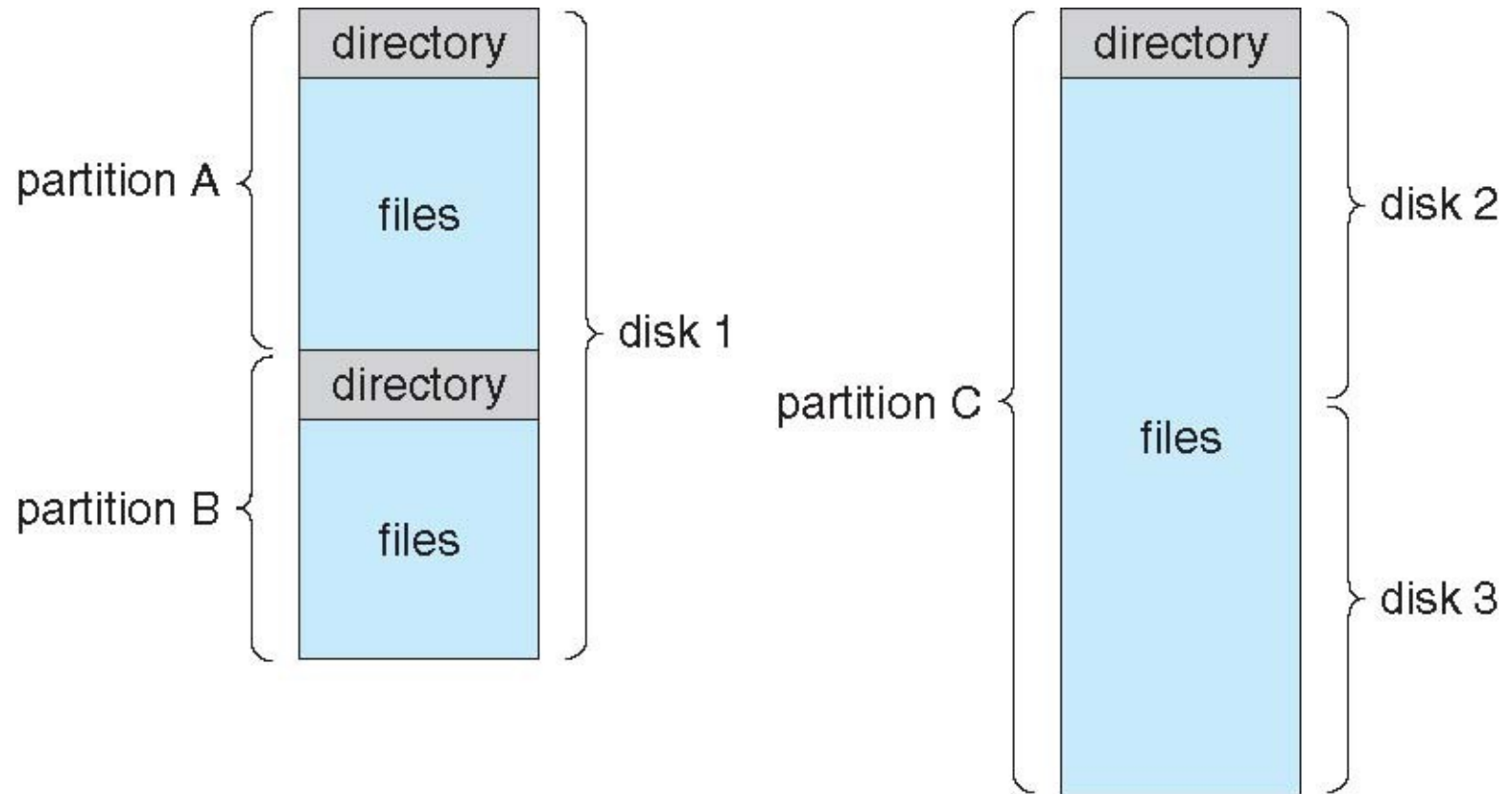
---

- 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





# 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

---

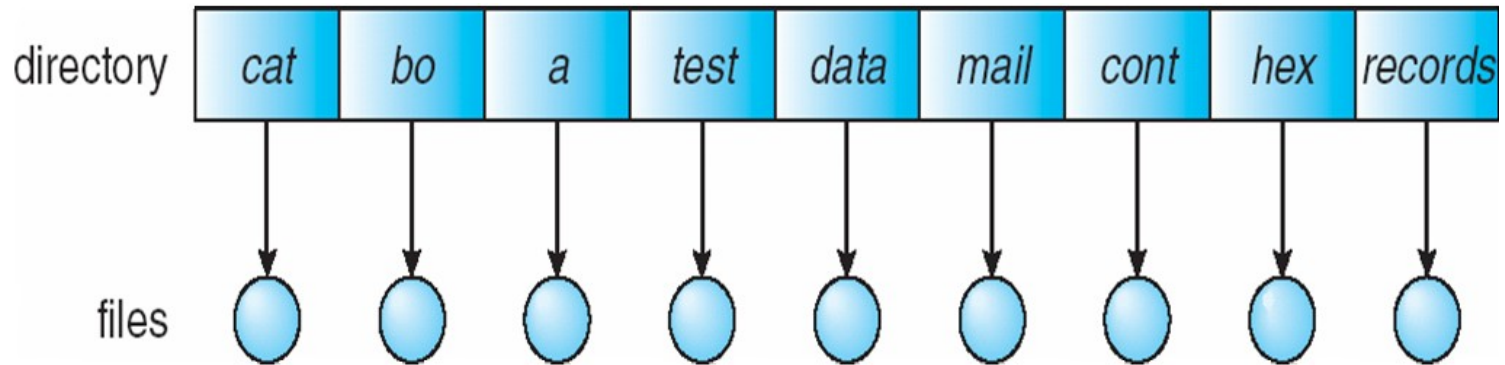
- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)





# Single-Level Directory

- A single directory for all users



Naming problem

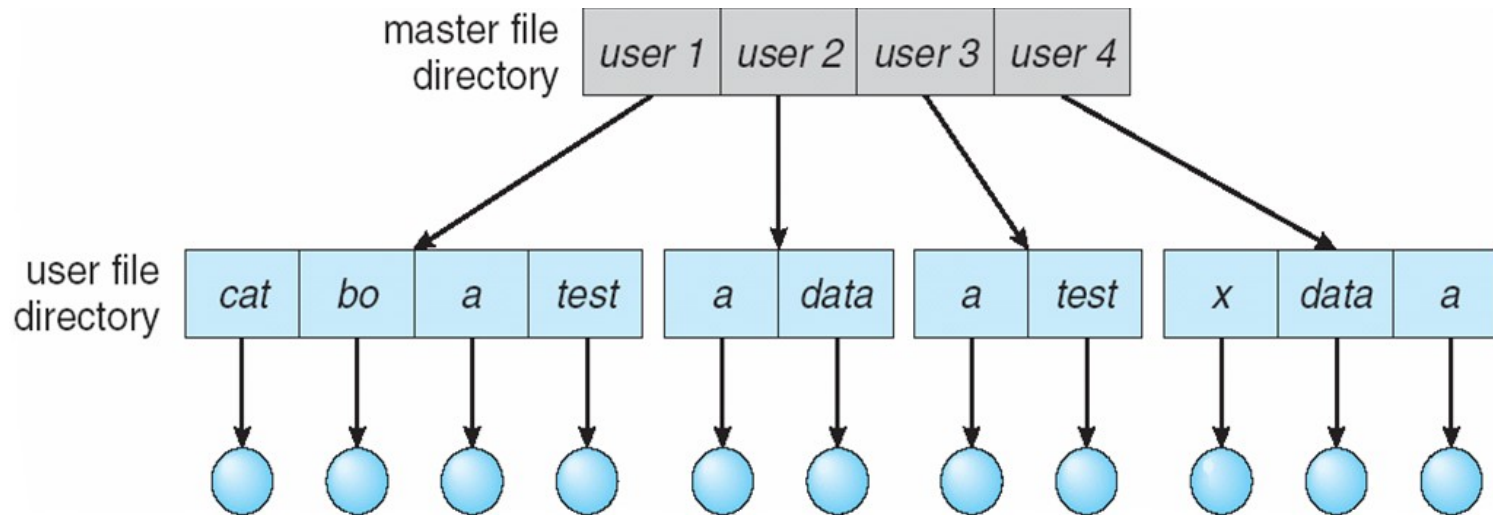
Grouping problem





# Two-Level Directory

- Separate directory for each user

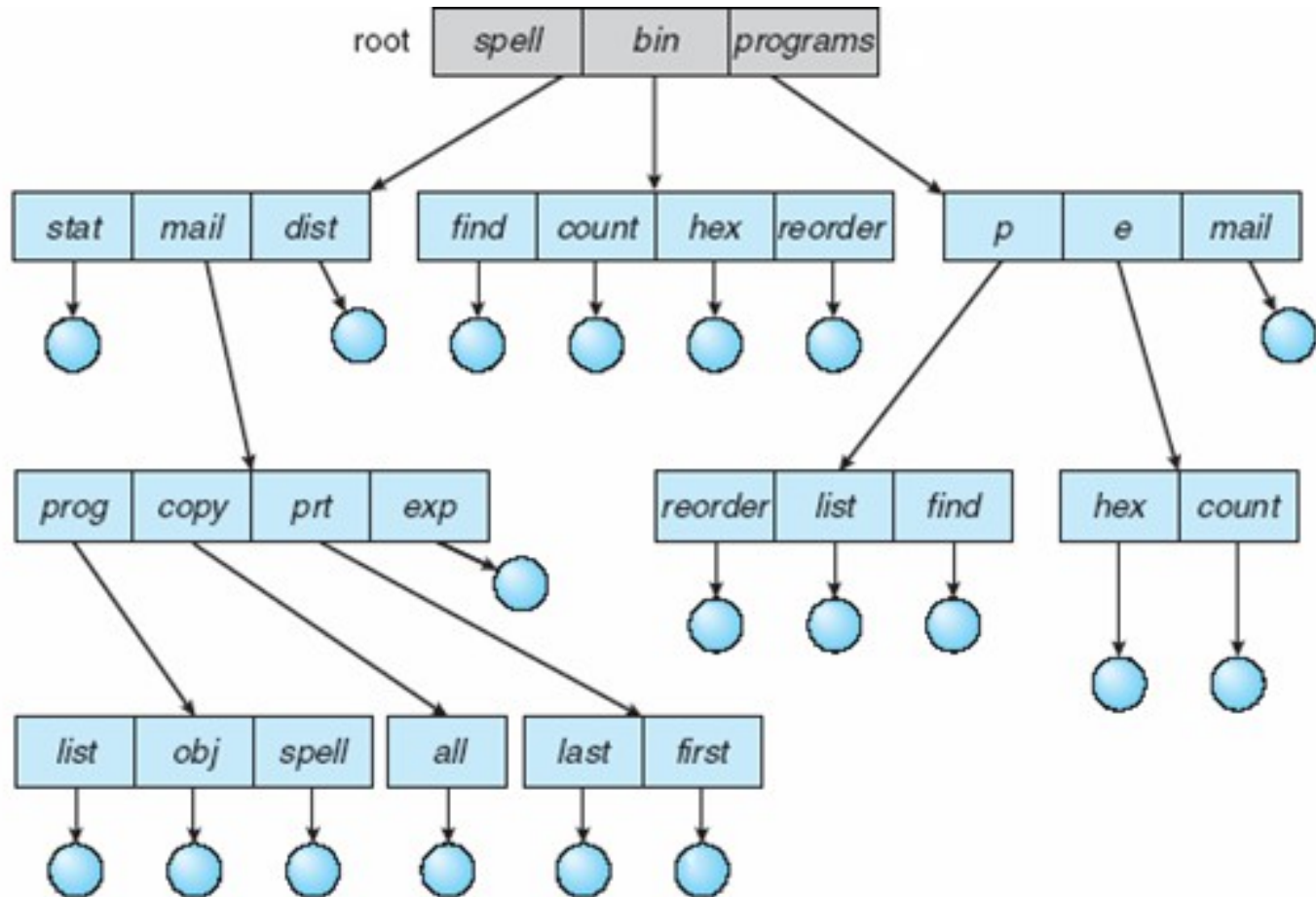


- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability





# Tree-Structured Directories







# Tree-Structured Directories (Cont.)

---

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



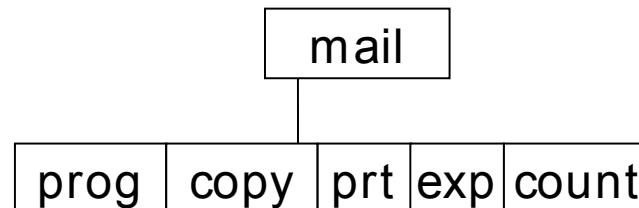


# Tree-Structured Directories (Cont)

- **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”





# File System Mounting

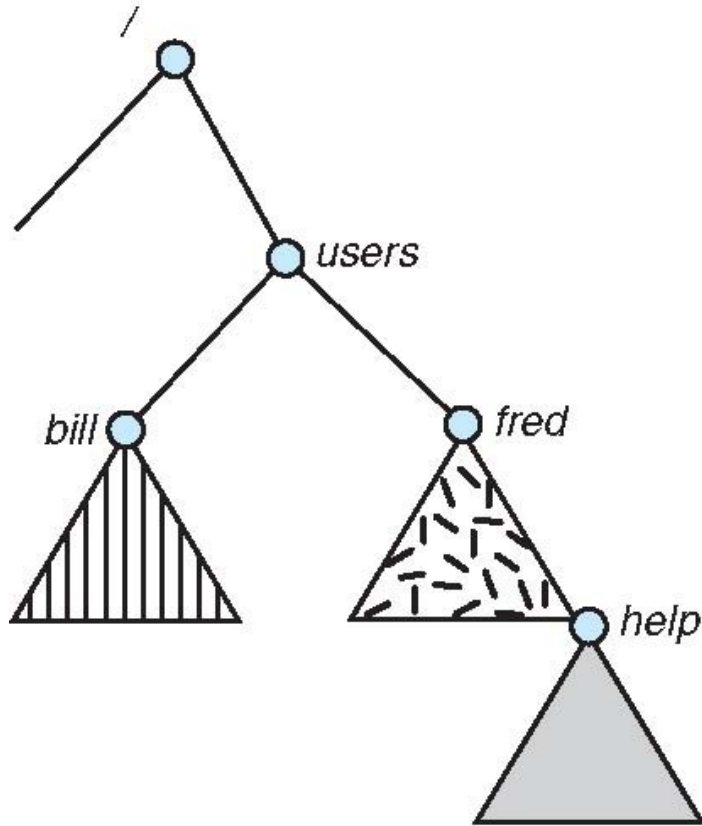
---

- 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**

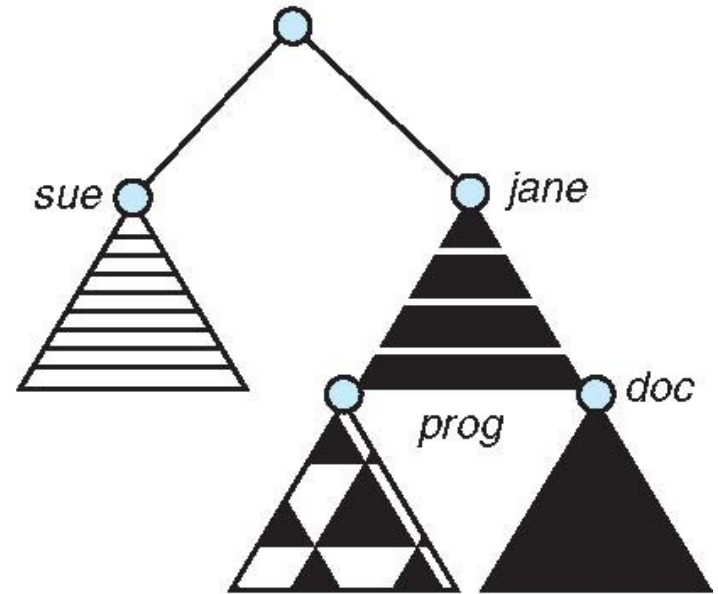




# (a) Existing (b) Unmounted Partition



(a)

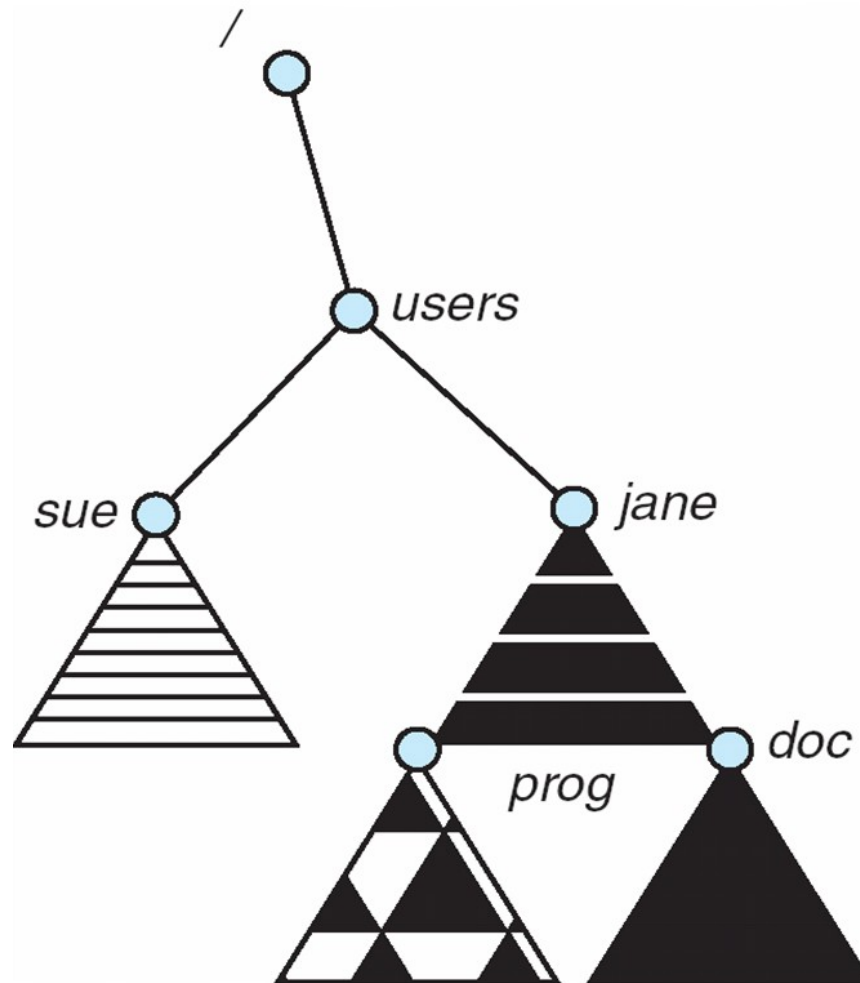


(b)





# 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





# 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**



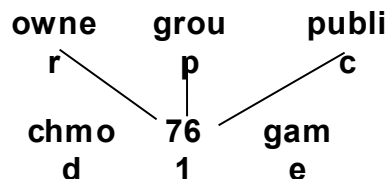


# Access Lists and Groups

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

		RWX	
a) <b>owner access</b>	7	⇒	1 1 1
		RWX	
b) <b>group access</b>	6	⇒	1 1 0
		RWX	
c) <b>public access</b>	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.

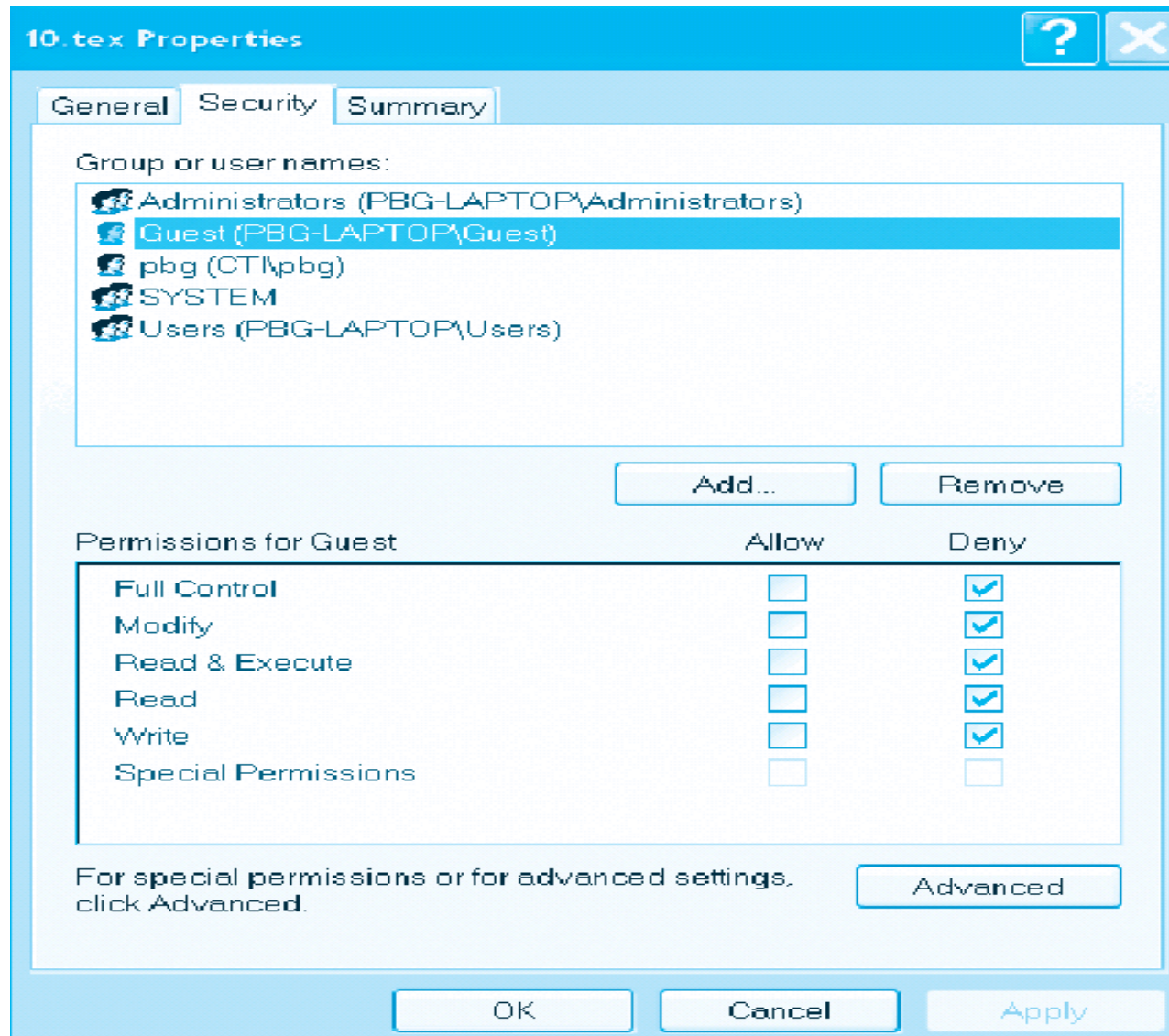


Attach a group to a file  
 chgrp G game





# Windows XP Access-Control List Management





# 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

---

