- What is the most visible aspect of an OS?

File System

- Provides mechanism for on-line storage of and acces to both data and programs of the OS and all the users of the computer system.
- File system consists of two distinct parts:
  - A collection of files each storing data, and
  - A directory structure which organizes and provides information about all the files in the system

File-System Interface

Silberschatz, Galvin, Gagne
-Moumita Patra
Jul-Nov 2019

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

File System

- OS abstracts from the physical properties of its storage devices to define a logical storage unit- the file
- Files are mapped by OS onto physical devices
- File- A named collection of related information that is recorded on secondary storage
- User's perspective- smallest allotment of logical secondary storage,i.e, data cannot be written to secondary storage unless they are within a file.

File Concept

- Contiguous logical address space
- Information in a file is defined by its creator
- Many types- Consider **text file, source file, executable file, etc**

Types of file:

Data

numeric

character

binary

Program

File Attributes

- A file is named for convenience of users and is referrred to by its name.
- A name is usually a string of characters.
- When a file is named, it becomes independent of the process, the user, and even the system that created it.

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 of files

Location – pointer to a device and location of the file on that device

Size – current file size (in bytes, words, or blocks). May also include maximum allowed size.

Protection – determines who can do reading, writing, executing, and so on

Time, date, and user identification – data for protection, security, and usage monitoring

- Information about files are kept in the directory structure, which resides in the secondary storage
- Information kept in the directory structure typically includes- file's name and identifier
- Identifier locates the file attributes

File Operations

- File is an **abstract data type**
- A file is defined by the type of operations that can be performed on it
- OS provides system calls for basic file operations, such as:
- **Create-** find space for file in the file system + create entry for a new file in the directory
- **Write –**

➔ corresponding system call specifies both the file and data to be written on it.

➔ Given the file name, search directory to find location.

➔ System keeps a **write pointer** to the location in the file where write is to take place.

➔ **Write pointer** must be updated whenever a write occurs.

- **Read –**
- ➔ Corresponding system call specifies name of the file + where (in memory) the next block of file is
- ➔ Directory is searched for associated entry
- ➔ **Read pointer** is kept at the location in the file where the next read is to take place
- **Reposition within file – file seek. Does not involve any actual I/O**
- **Delete**
- ➔ Search the directory for the named file, release associated file space and erase directory entry
- **Truncate**
- ➔ Erase contents of a file but keep its attributes

- ***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**
  - **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 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, 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

- Source and object files have structures that match the expectations of the programs that read them
- Files must conform to a required structure that is understood by the OS

  Disadvantages of the OS supporting multiple file structures:
- Resulting size of the OS is cumbersome.
- OS needs to contain corresponding code to support different file structures.
- Severe problems may result if an application requires files structured in a way not supported by the OS

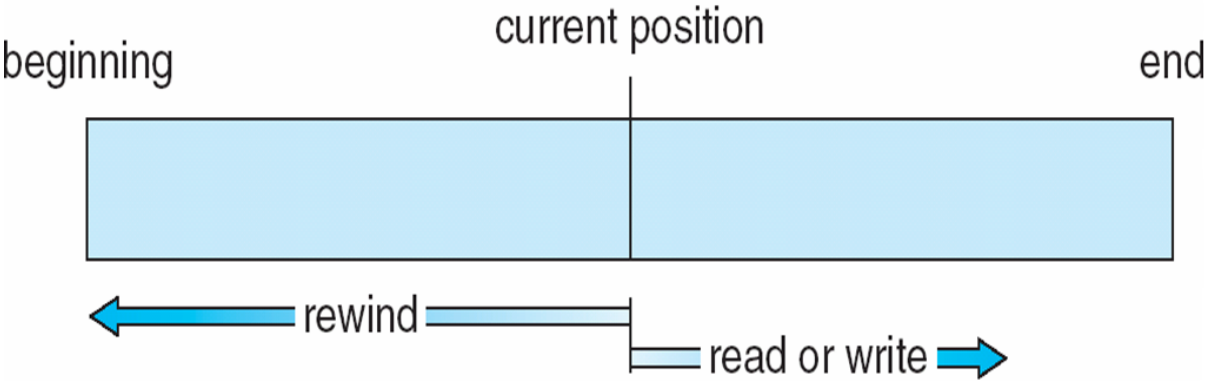
Internal File Structure

- Disk systems have a well defined block size determined by the size of a sector.
- All disk I/O is performed in units of one block (physical record). All blocks are of same size.
- Physical record size may not be same as logical record size => solution is to pack a number of logical records into physical blocks.
- Logical record size, physical block size, and packing technique determine how many logical records are in each physical block.
- Packing can be done either by OS or by the user's application program.
- Thus, file may be considered as a sequence of blocks.
- All file systems suffer from internal fragmentation. Larger the block size, greater the internal fragmentation.

Access Methods

- Information in files must be accessed and read into computer memory when file is in use.

Sequential-access File



- Information in file is processed one record after another.
- Most commonly used access method.
- read_next()- reads the next portion of the file and automatically advances file pointer which tracks the I/O location
- write_next()- appends to the end of the file and advances to the end of the newly written material

Direct Access

- A file is made up of fixed-length logical records that allows programs to read or write records rapidly in no particular order.
- Helps in immediate access to large amounts of information.
- Databases are often of this type.
- Concept- Read the block directly to get the required information.
- File operations need to be modified to support block number (relative block number), e.g- read_next() becomes read(n), n being the block number.
- Use of relative block numbers helps OS to decide where the file should be placed (allocation problem)
- It also helps to prevent user from accessing portions of the file system.
- It is easy to implement sequential access on a direct access file system, the reverse is extremely inefficient and complicated.

Access Methods

- **Sequential Access**

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

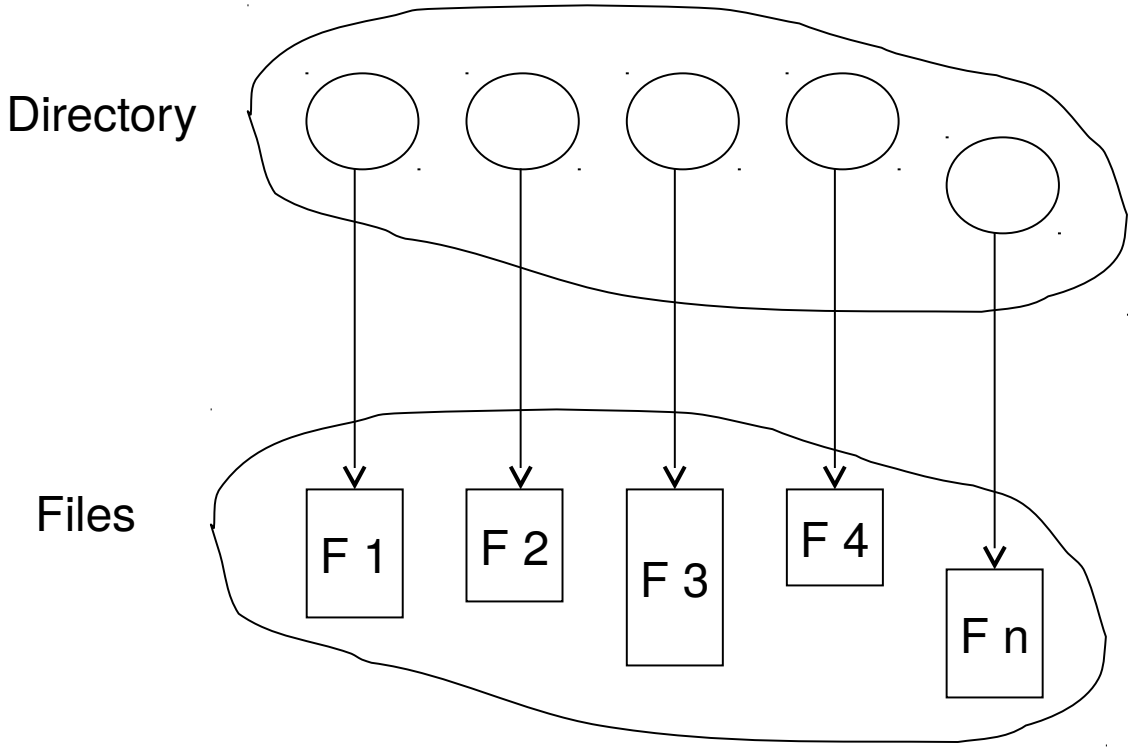
- **Direct Access –** file is fixed length logical records

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

  n = relative block number

Directory Structure

- A collection of nodes containing information about all files

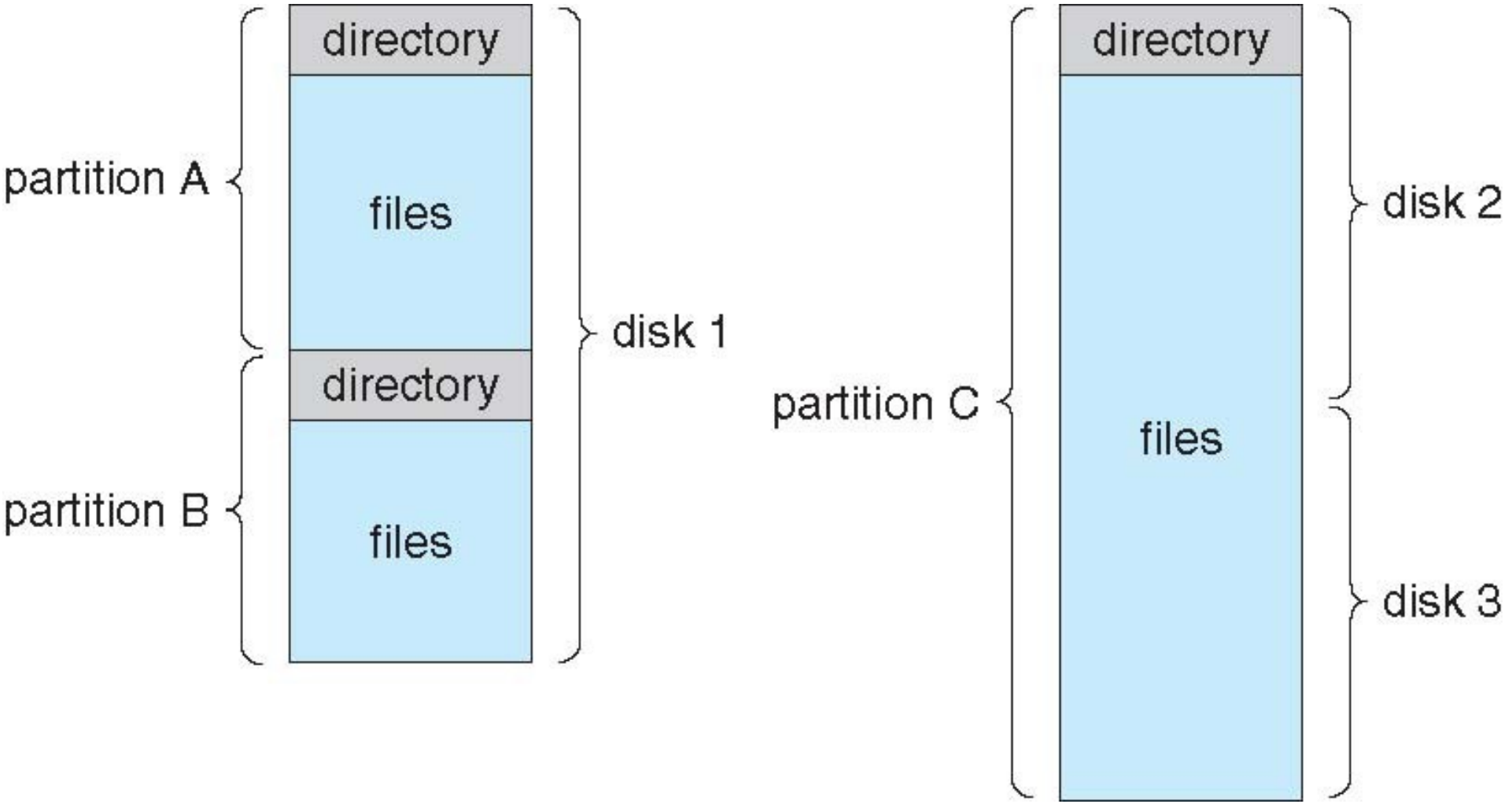


Both the directory structure and the files reside on disk

Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used – 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**

A Typical File-system Organization



Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris , it 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 which 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

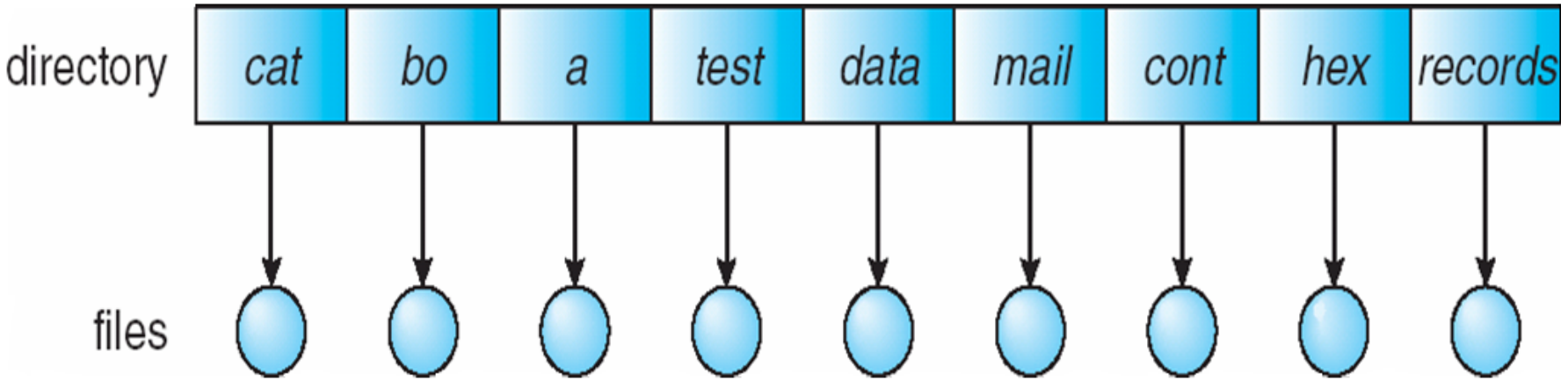
Directory Organization

- A 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

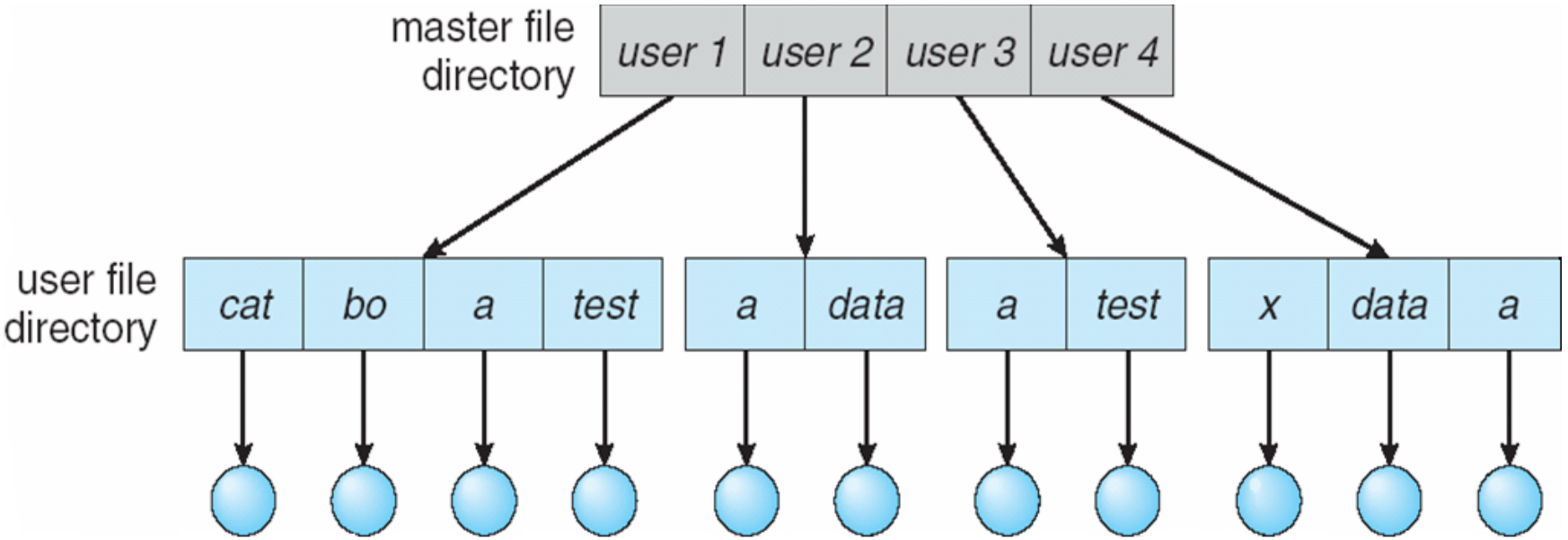
- A single directory for all users



- Naming problem
- Grouping problem

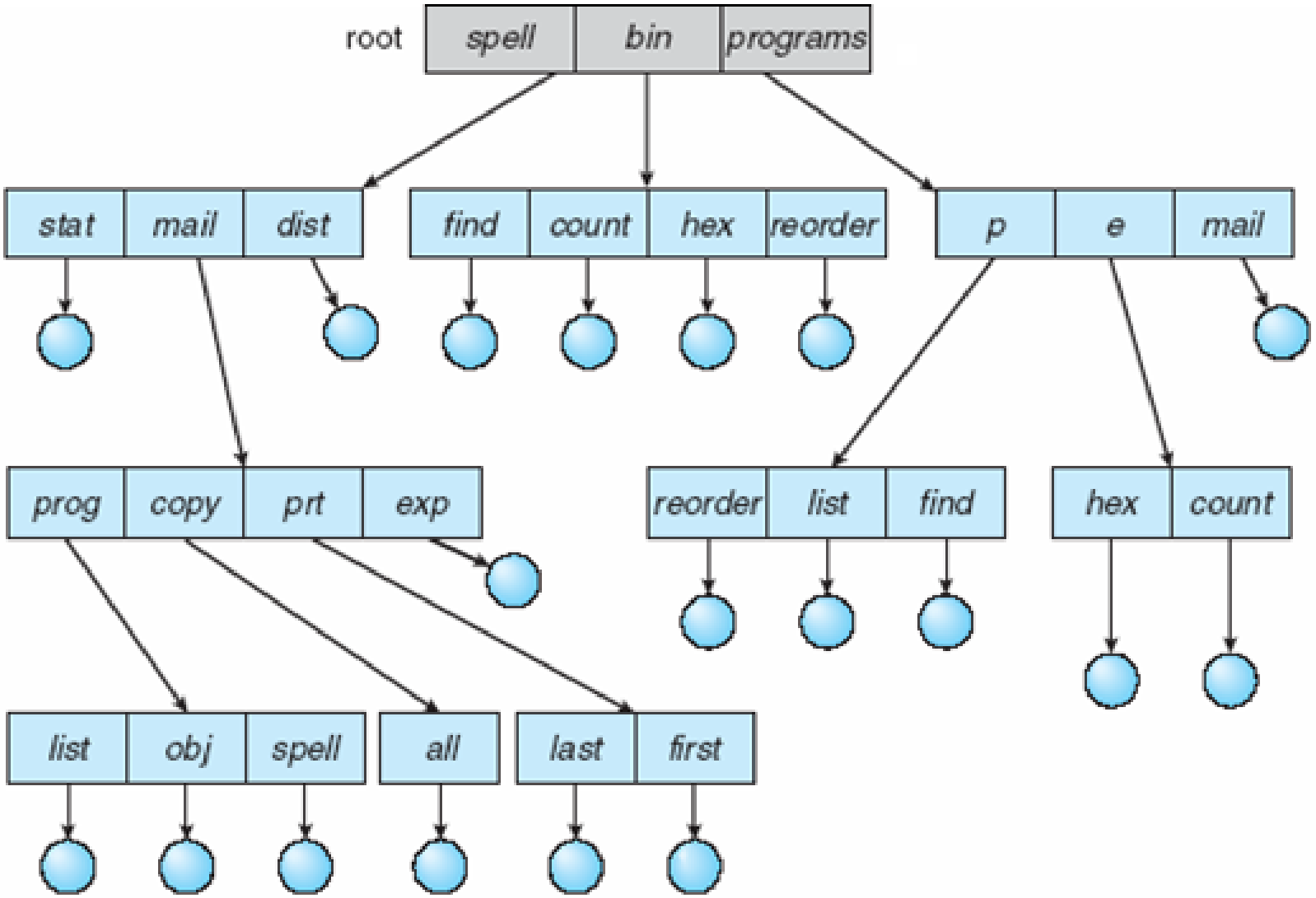
Two-Level Directory

- Separate directory for each user



- 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)

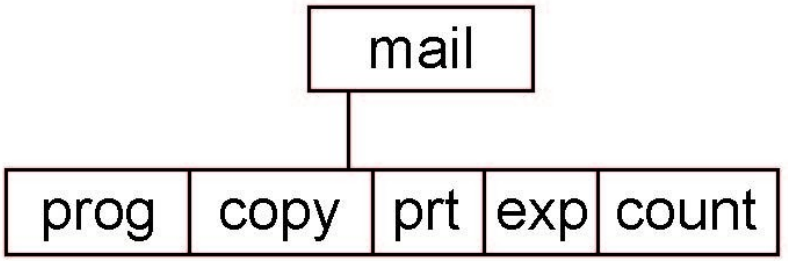
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

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

  `mkdir <dir-name>`

  Example: if in current directory **`/mail`**

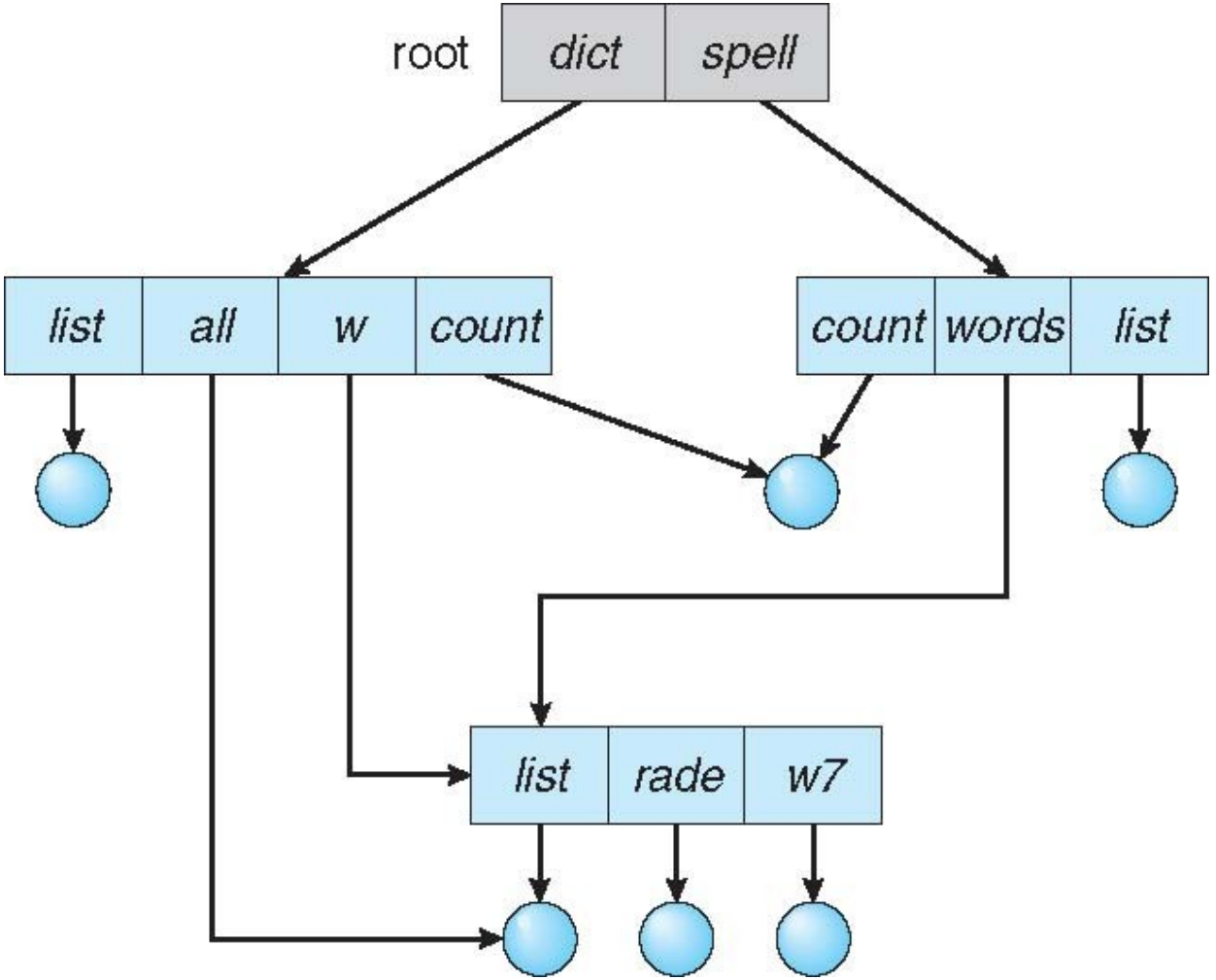
  `mkdir count`



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

Acyclic-Graph Directories

- Have shared subdirectories and files



Acyclic-Graph Directories

- Allows sharing of files and directories.
- Sharing can be implemented in several ways-
  - **Link**- Create a new directory entry called a link. Link is a pointer to another file or subdirectory.
  - Link may be implemented as an absolute or a relative path name.
  - When a reference to a file is made=> search the directory => if directory entry is marked as a link, then resolve the link.
  - **Resolve**- by using path name to locate the real file.
- Or, simply duplicate all information about them in all the shared directories.- pros and cons?

Challenges

- A file may have multiple absolute path names.
- Distinct file names may refer to the same file.
- Deletion:

➔ When to deallocate space allocated to the shared file and reuse it?

➔ Solution 1: Remove the file whenever anyone deletes it. Challenge- may leave dangling pointers to the now non-existent file.

➔ Solution 2: Remove file, whenever attempt is made via a link to access a non-existent file, show error.Challenge- What to do when a file is deleted and another file of the same name is created, before a symbolic link to the original file is used.

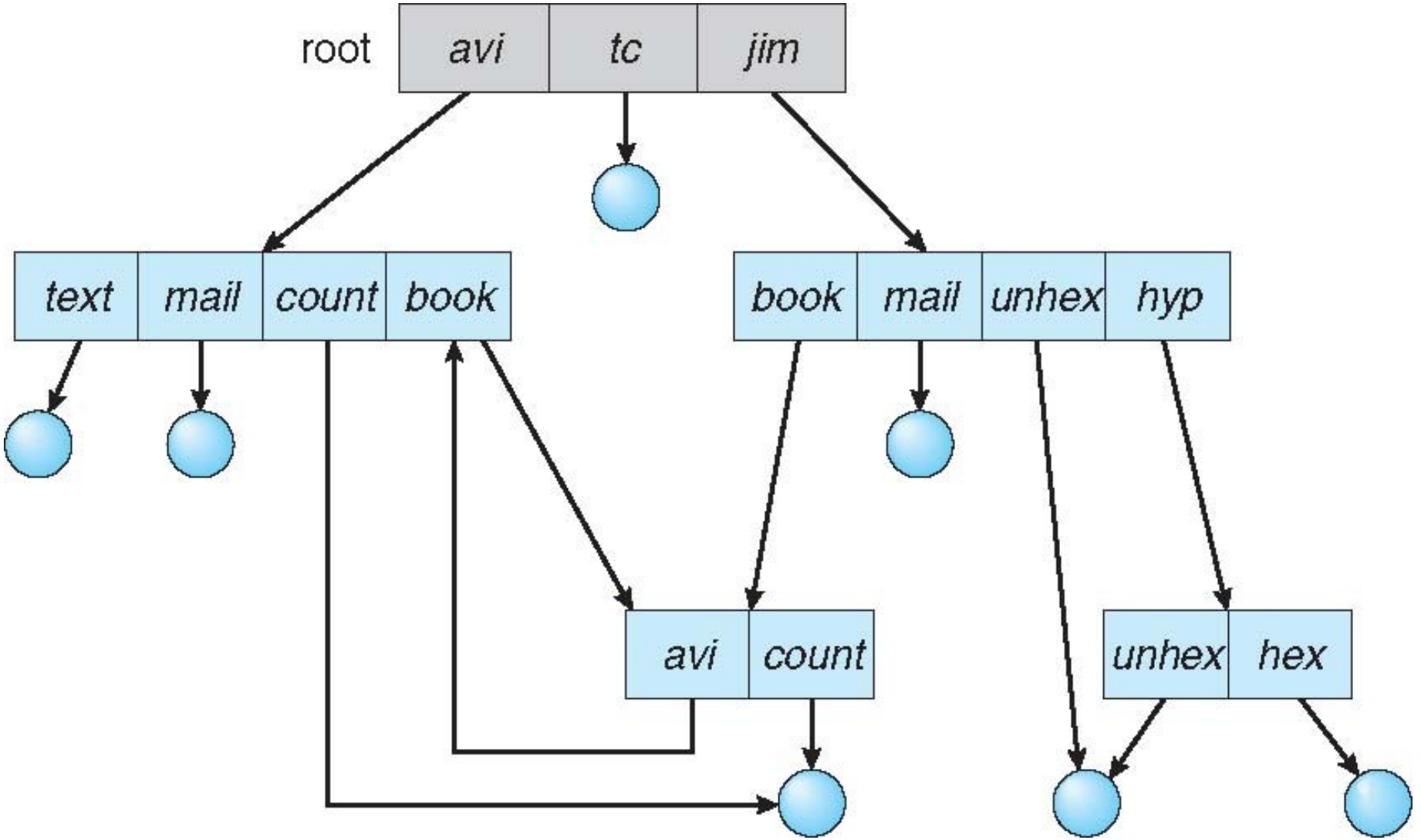
➔ Solution 3: Preserve file until all references to it are deleted. Challenge- Need to keep a list of file-references and ensure to delete all of them.

Acyclic-Graph Directories (Cont.)

- Two different names (aliasing problem)
- 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
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file
    - Always need to check for cycles!

General Graph Directory

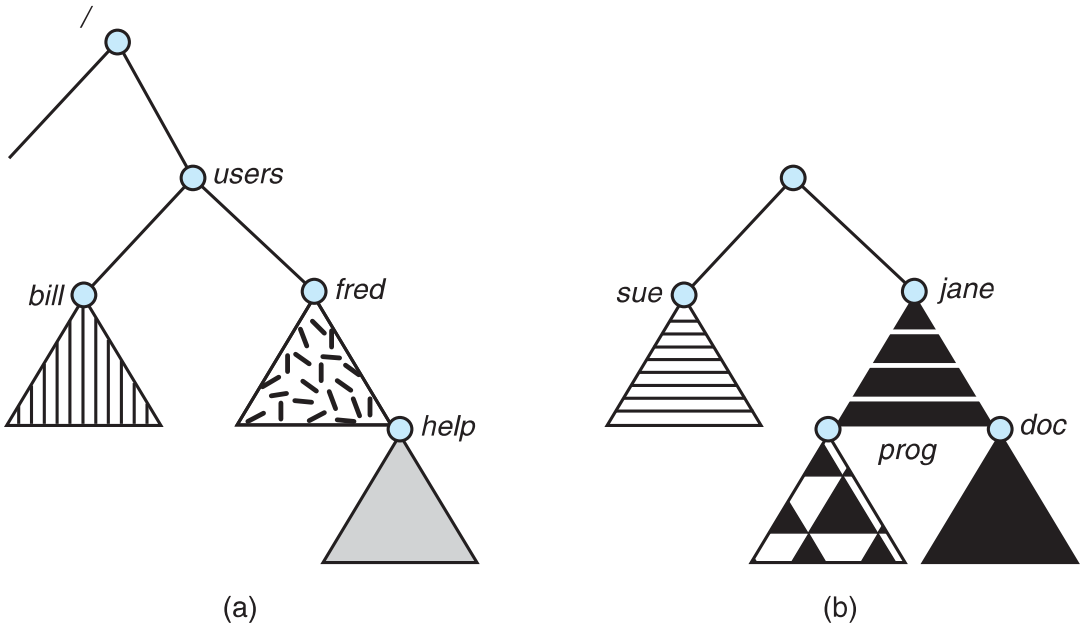


General Graph Directory (Cont.)

- Need to avoid searching any component twice- for correctness as well as for performance.
- How do we guarantee no cycles?
  - Allow only links to files 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

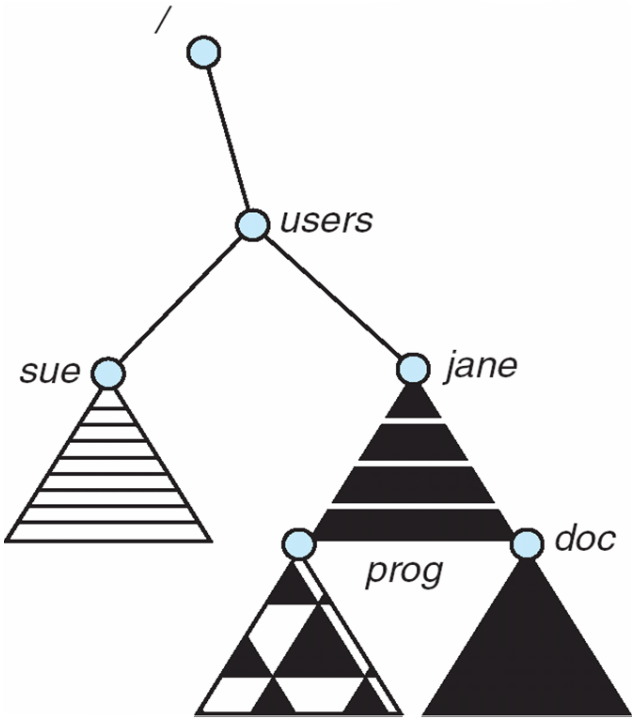
- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**
- OS is given the name of the device and the mount point
- OS verifies that the device driver contains valid file system
- Finally, OS notes that a file system is mounted in its directory structure at the specified mount point



- a- existing system, b- unmounted volume residing in /device/dsk

Mount Point

- Effects of mounting /device/dsk over /users.



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
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

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 v3 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
  - 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

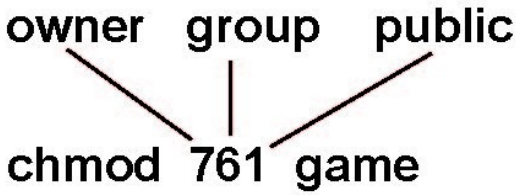
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

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

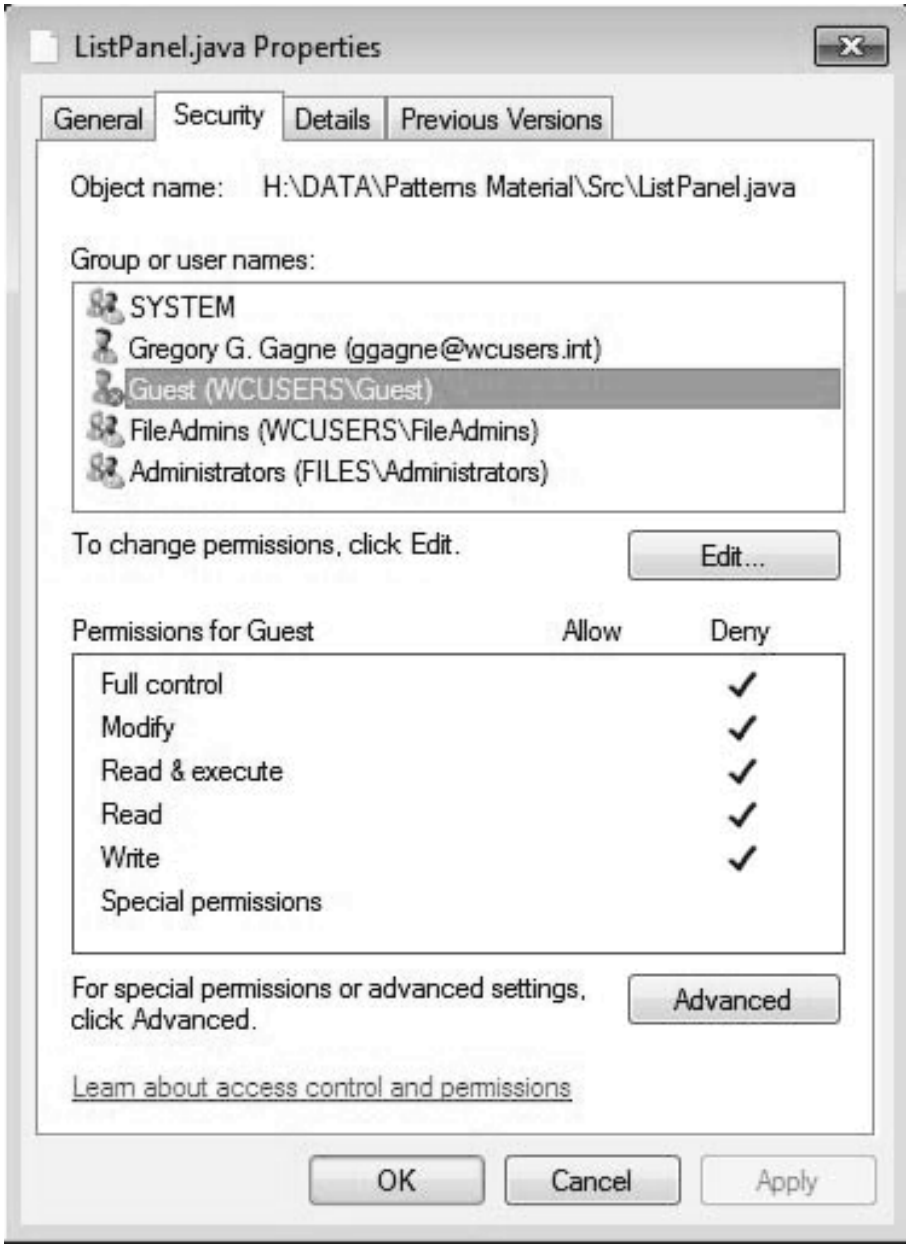
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

```
chgrp      G     game
```

Windows 7 Access-Control List Management



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/