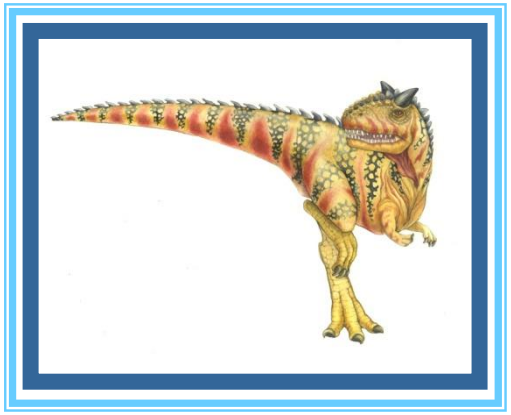
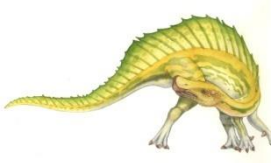
Chapter 10: File System



Chapter 10: File System

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection
- **Objective**
  - 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

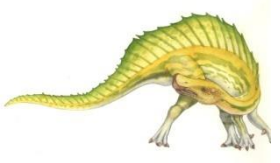


File Concept

- Contiguous logical address space
- Contents defined by file's creator
  - Many types: text file, source file, executable file
- **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 directory structure, maintained on disk
- Many variations, including extended file attributes such as file checksum

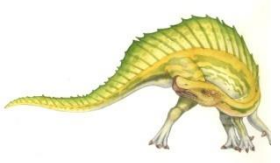
File Operations

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open*(F_i)** – search the directory structure of F_i on disk and move to memory
- ***Close*** (F_i) – move the content of F_i in memory to 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**: cache of data access information
  - **Access rights**: per-process access mode information

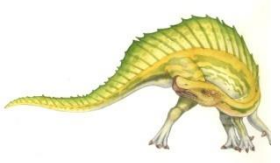


File Types – Name, Extension

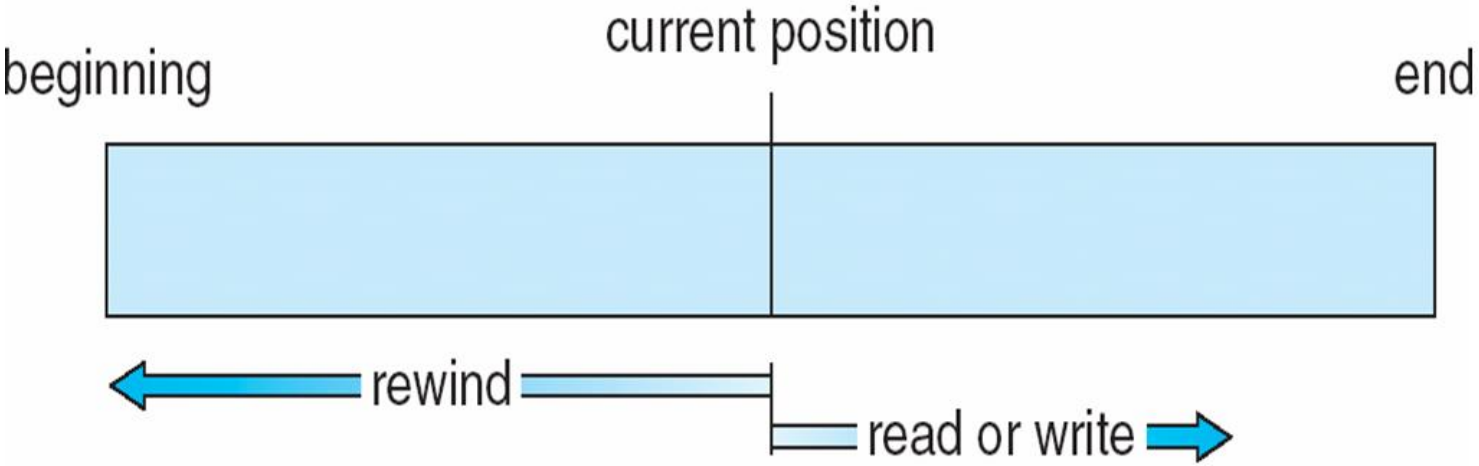
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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
- Who decides:
  - Operating system
  - Program



Sequential-access File



Access Methods

- **Sequential Access**

```
read next
write next
reset   (reset the file position)
```

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

```
read n
write n
position to n
read next
write next
```

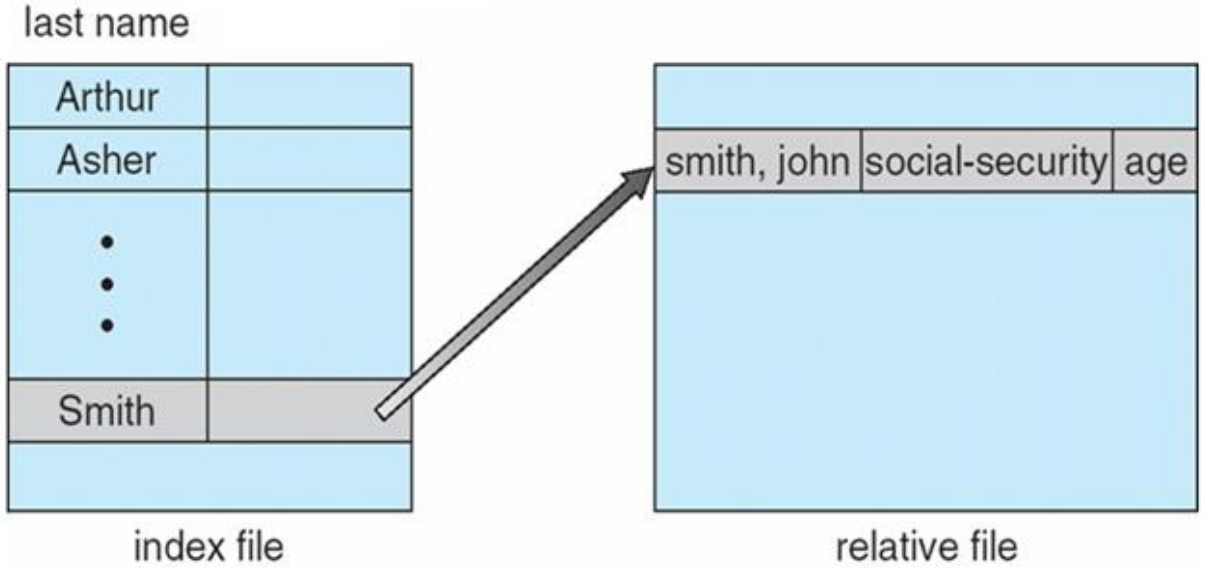
n = relative block number
(relative to the beginning of the file)

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;

Other Access Methods: Index Files

- Example of **index file**
  - Assume 64 records per block, a file of 120000 records would occupy about 2000 blocks.
  - By keeping the file sorted by a key, we can define an index consisting of the first key in each block.
- Keep the index file in memory can speed up file access
- If too large, index (in memory) of the index (on disk)
  - Primary index file→ secondary index file→ actual data

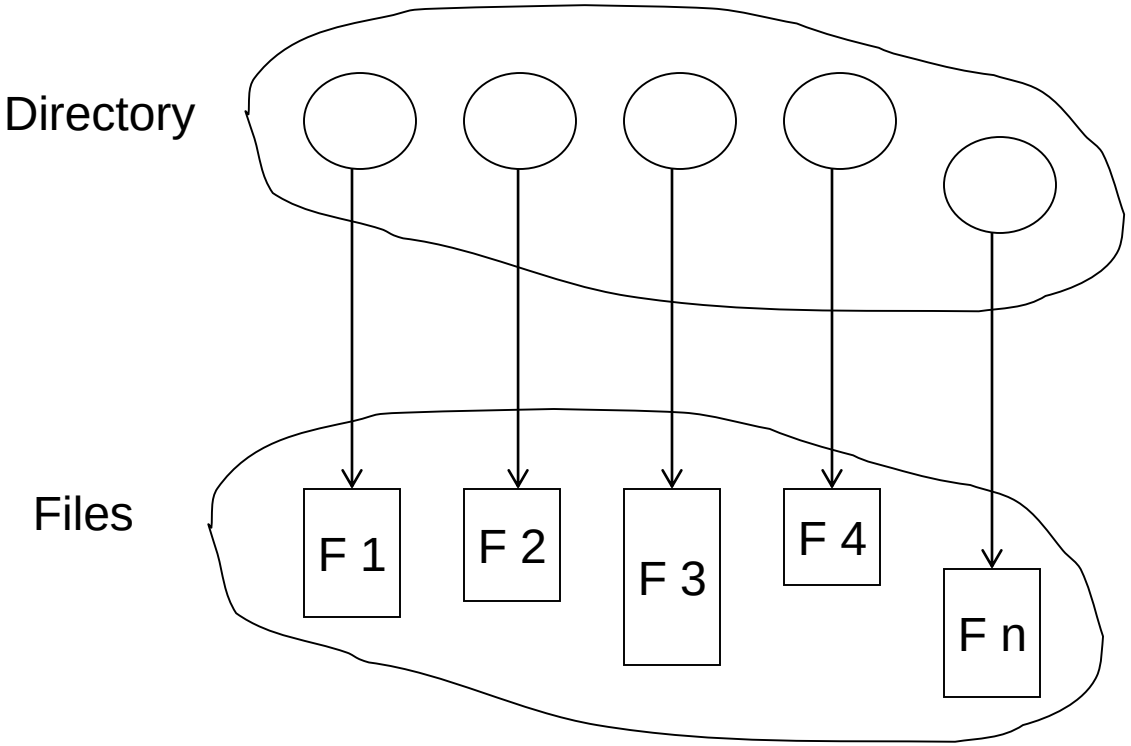


Chapter 10: File System

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

Directory Structure

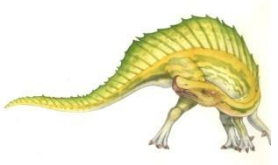
- A collection of nodes containing information about all files



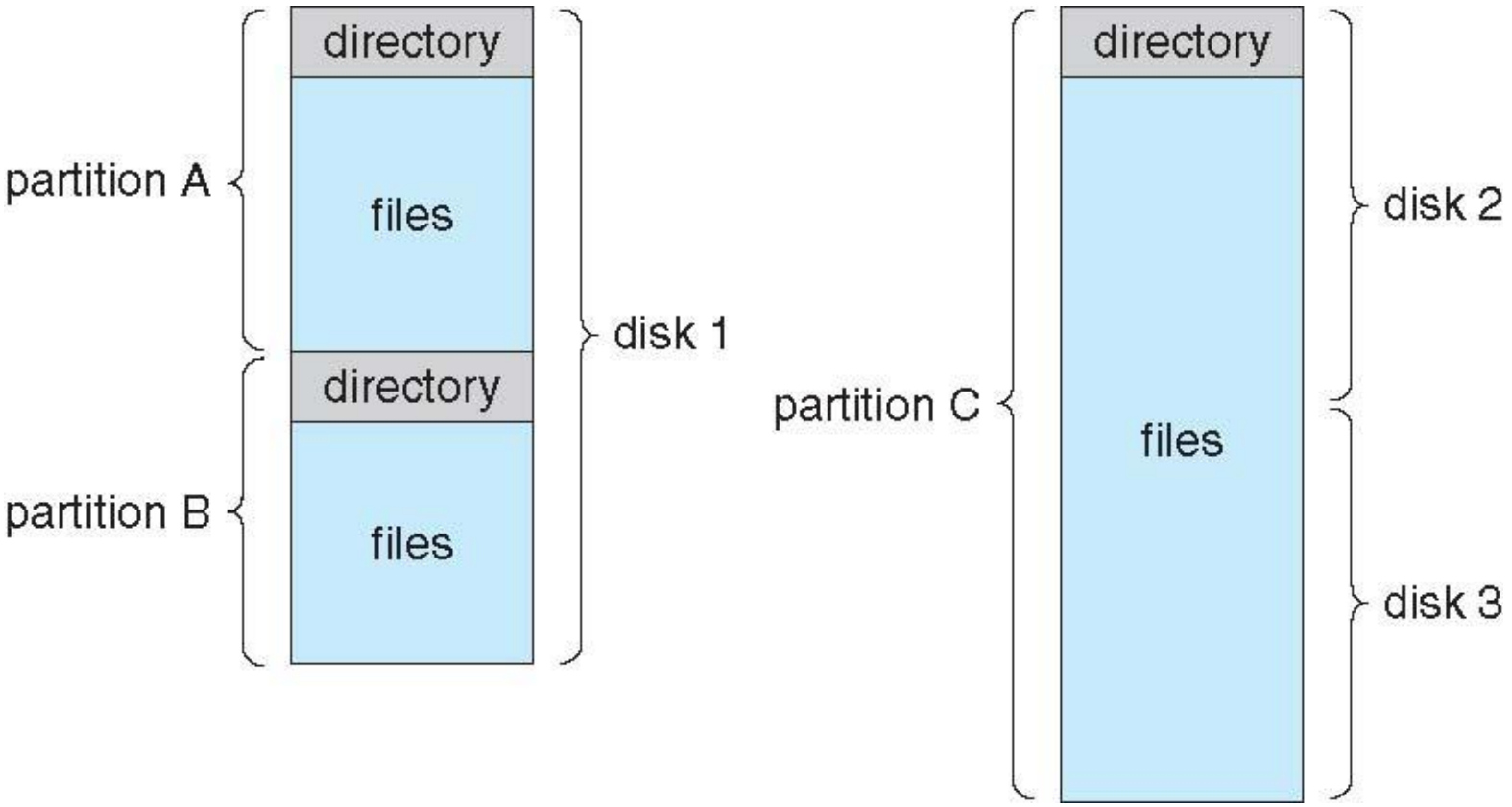
Both the directory structure and the files reside on disk

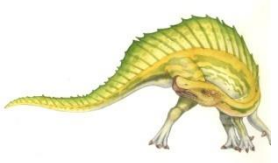
Disk Structure

- A single disk can be subdivided into **partitions**
- Multiple disks can also form **a partition**
- Disk or partition can be used as a **raw disk**– without a file system, or can be **formatted** with a file system
- Partitions containing file system known as **volume**
- Each volume containing file system also contains information about the files in the system. This info is kept in entries in device directory or volume table of contents (commonly known as **directory**)
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer



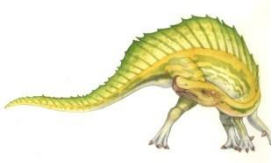
A Typical File-system Organization





Types of File Systems

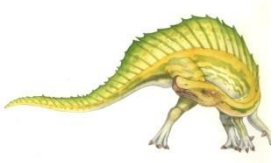
- We mostly talk of general-purpose file systems, But systems may have many file systems, some general and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – a “virtual” file system that gives debugger the access to kernel symbols for debugging (i.e., kernel looks like a file system.)
  - ctfs – “contract” file system for managing daemons
    - contains info about which processes start when the system boots and must continue to run during operation
  - lofs – loopback file system allows one FS to be accessed in place of another (i.e., provide an alternate path to an existing file system).
  - procfs – kernel interface to process structures (info. about process)
  - ufs, zfs – general purpose file systems



Operations Performed on Directory

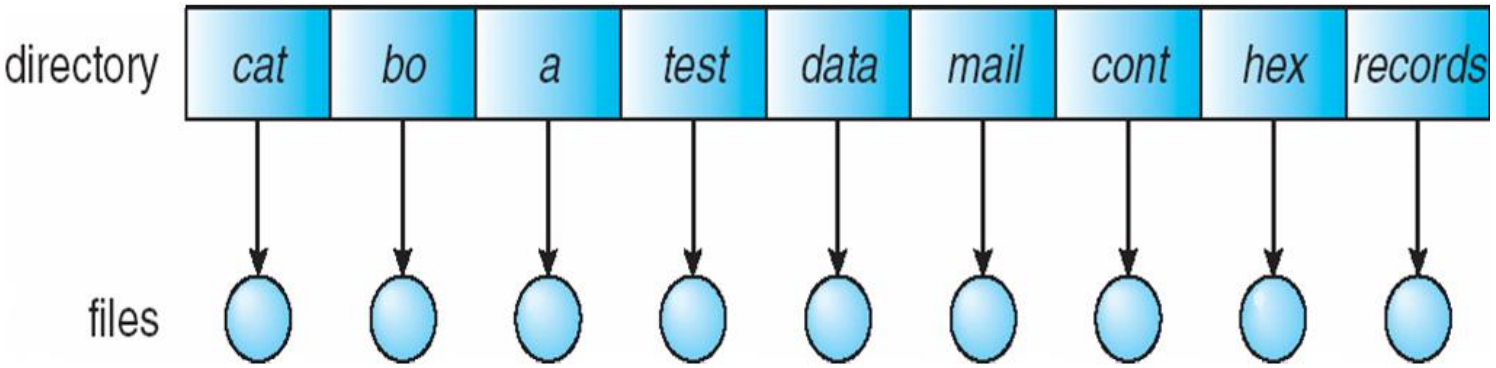
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

- Organize the directory
  - 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

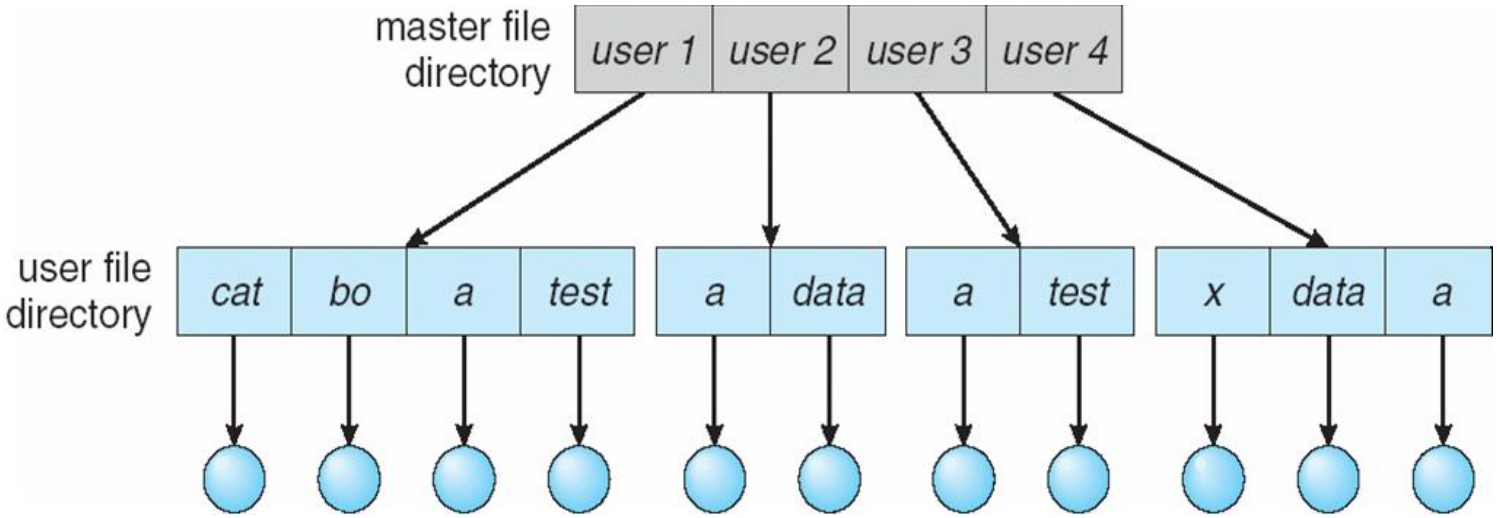


Naming problem

Grouping problem

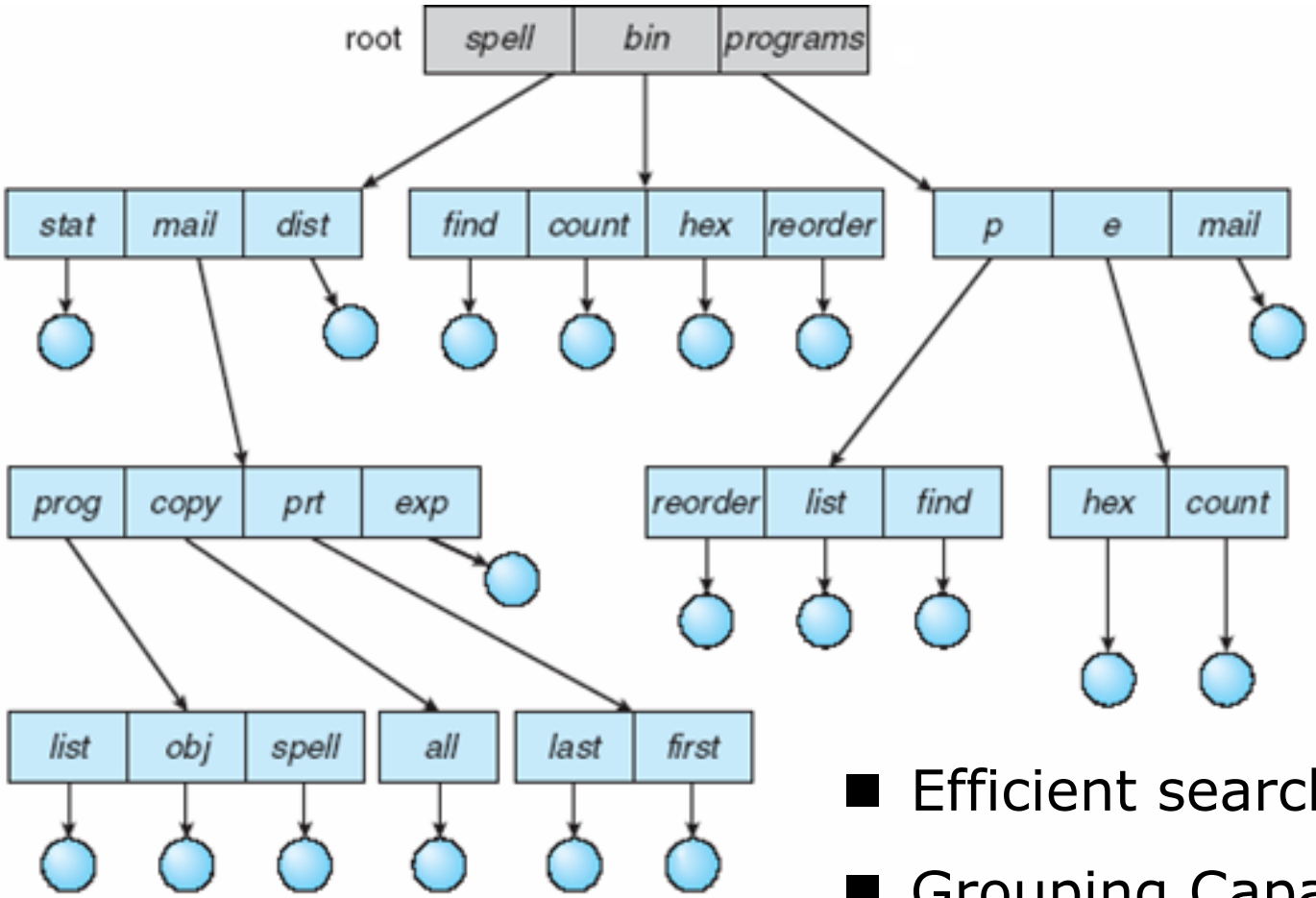
Two-Level Directory

- Separate directory for each user

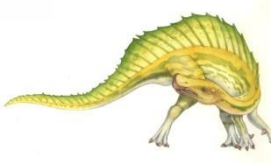


- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



- Efficient searching
- Grouping Capability



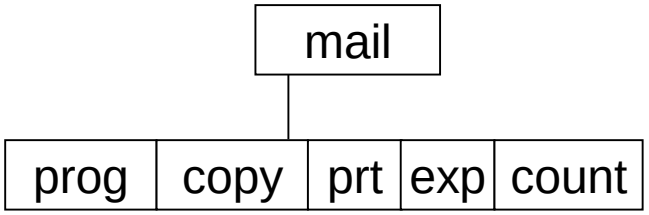
Tree-Structured Directories (Cont)

- Current directory (working directory)
  - **`cd /spell/mail/prog`**
  - **`type list`**
- **Absolute** or **relative** path name
  - root/spell/mail/prt/first
  - /prt/first if current directory is at root/spell/mail
- 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>`

e.g., if in current directory **`/mail`**

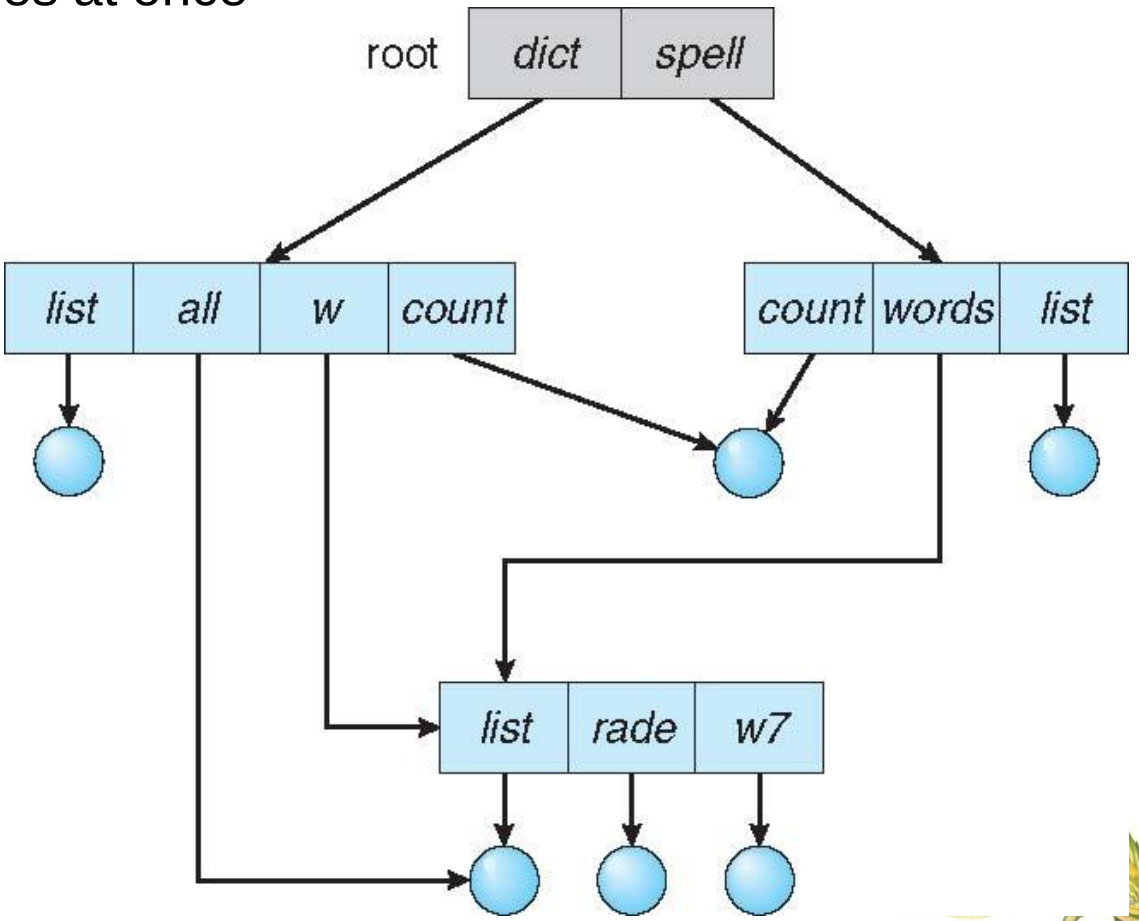
`mkdir count` ➔



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

Acyclic-Graph Directories

- Have shared subdirectories and files
- Allows the same directory or file exists in the file system in two (or more) places at once

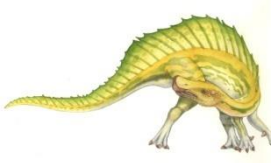


Acyclic-Graph Directories (Cont.)

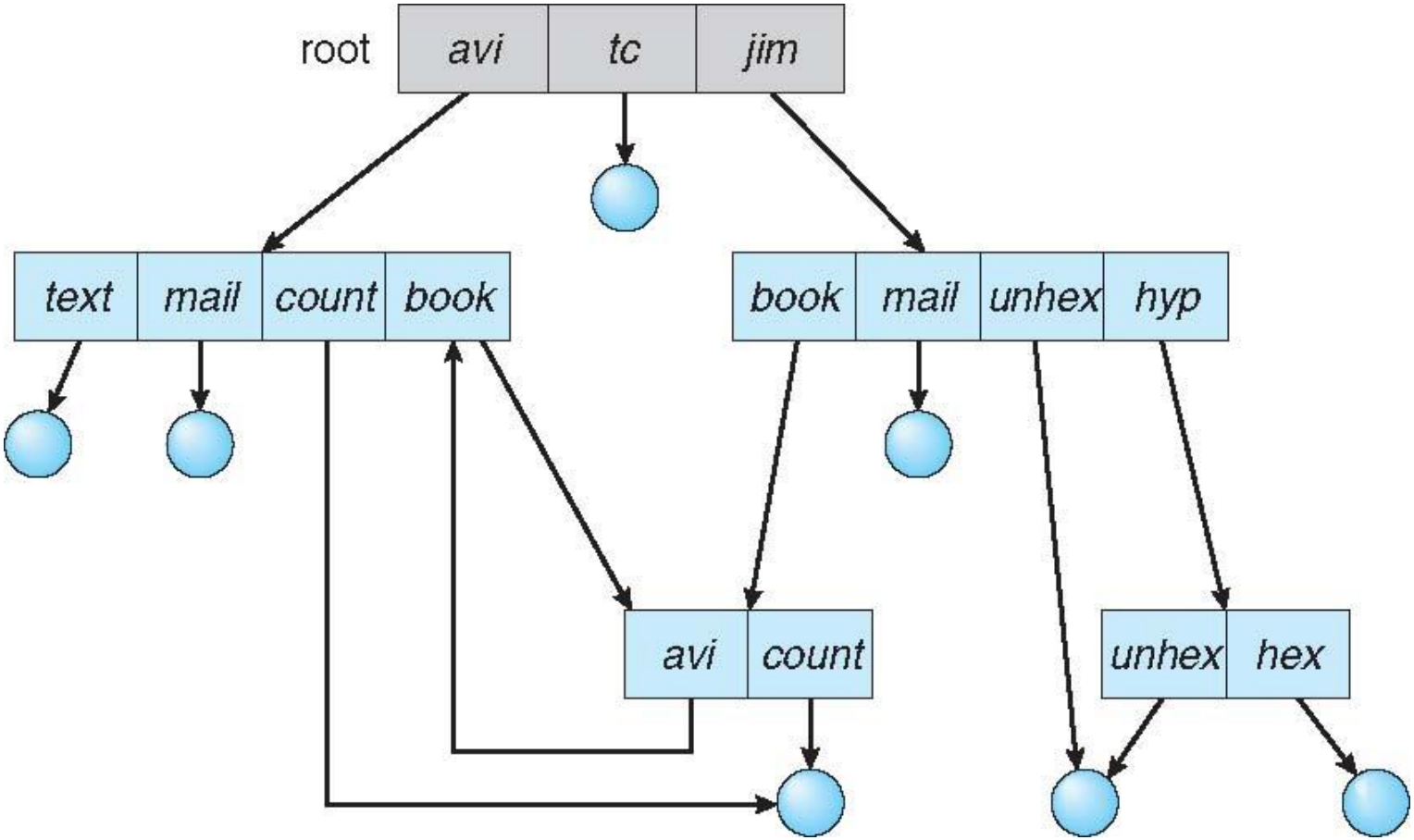
- Implementation: create a new directory entry type called a link
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file
  - The OS ignores these links when traversing the entire file system (e.g., to find a file, or to copy all files to backup storage) for correctness as well as performance.
- Problems:

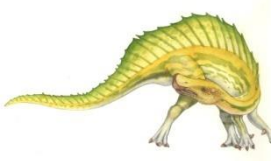
1. **aliasing**: a file has two (or more) different names (multiple absolute paths)
2. If *dict* deletes *list* ⇒ **dangling pointer**

   Solutions:
   - Symbolic links are left when a file is deleted, and its up to the user to realize that the original file is gone. (e.g., UNIX and MS Windows)
   - Preserve the shared file until all references to it are deleted. Keep a list of all references to a shared file.
   - Entry-hold-count solution (use “count” instead of a list)



General Graph Directory





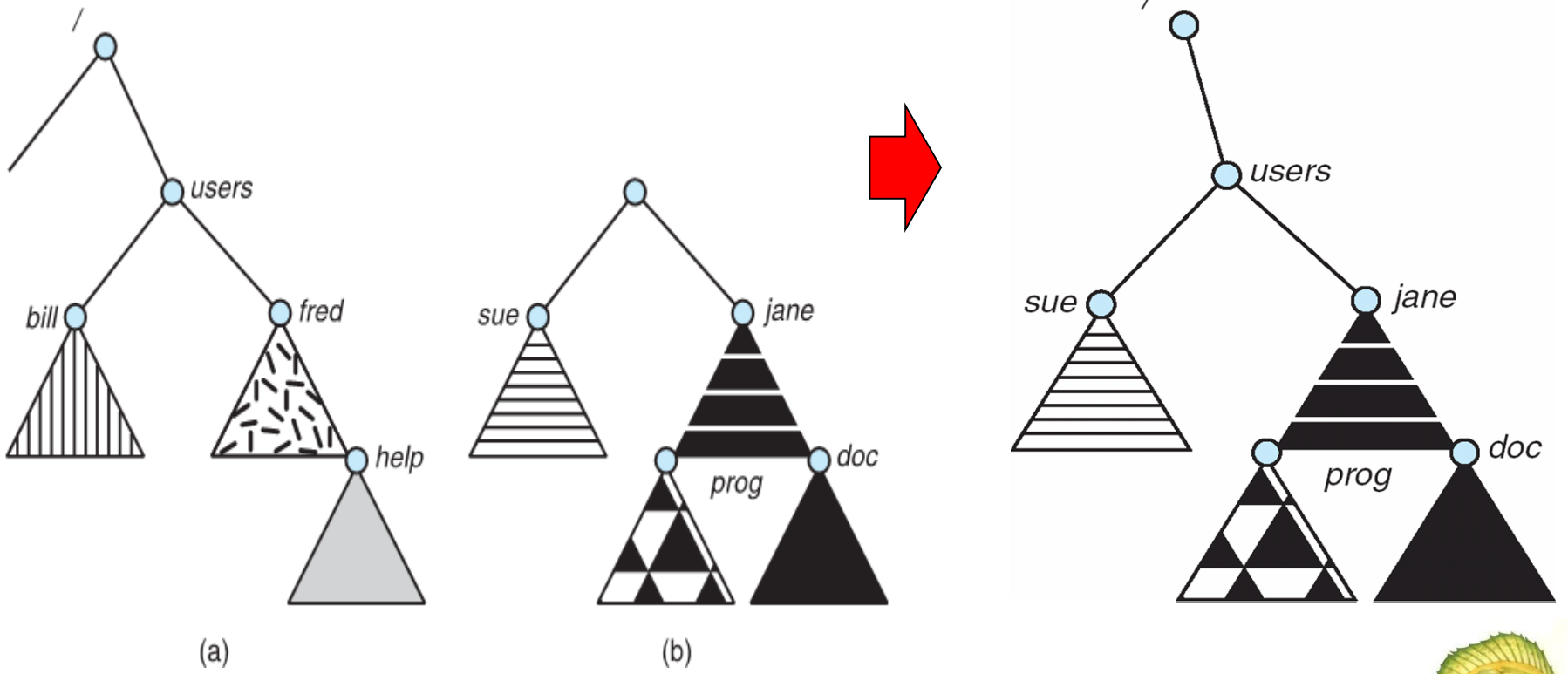
General Graph Directory (Cont.)

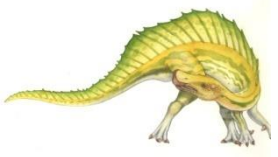
- Issues with graph directory (i.e., cycles are allowed)
  - how to avoid searching any component more than one time, for reasons of correctness as well as performance.
    - Poorly designed algorithm may enter an infinite loop through cycle
  - Reference count is ***not 0***, but the file or directory cannot be accessed.

    Garbage collection
    - To determine when the last reference has been deleted and the disk space can be reallocated. Two passes below.
      - 1. traverse the entire file system, making everything that can be accessed.
      - 2. collects everything that is not marked to a list of free space.
    - Remove cycles:
      - Every time a new link is added use a cycle detection algorithm to determine whether it is OK ➔ computationally expensive.

File System Mounting

- A file system must be **mounted** before it can be accessed
- The location within the file structure where the file system is to be attached is called **mount point**



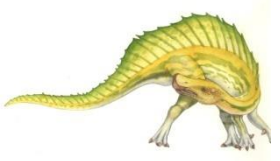


Chapter 10: File System

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

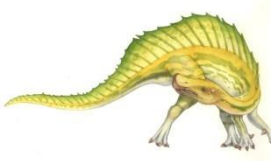
File Sharing

- 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



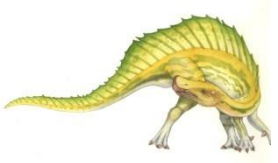
File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP (anonymous or authenticated)
  - Automatically, seamlessly using **distributed file systems** (remote directories are visible from a local machine)
  - Semi automatically via the **world wide web** (typically, anonymous)
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls



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
  - terminate all operations
  - or delay operations
- 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

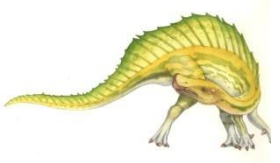
- How multiple users are to access a shared file simultaneously
  - Similar to Ch 6 process synchronization algorithms
    - Impossible to be implemented in case of remote file I/O (due to disk I/O and network latency)
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer of current location into the file.

    ➔ A file is a single physical image accessed as an exclusive resource. Concurrent access may be delayed due to contention.
  - Andrew File System (AFS) has session semantics
    - Writes only visible to sessions starting after the file is closed.

    ➔ A file may be associated temporarily with several images. Concurrent access to their images can be done without delay.

Protection

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

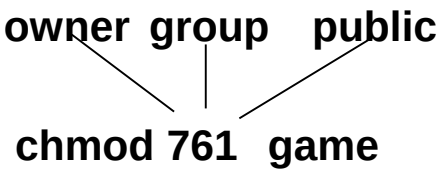


Access Lists and Groups

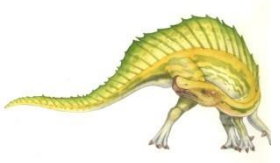
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

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

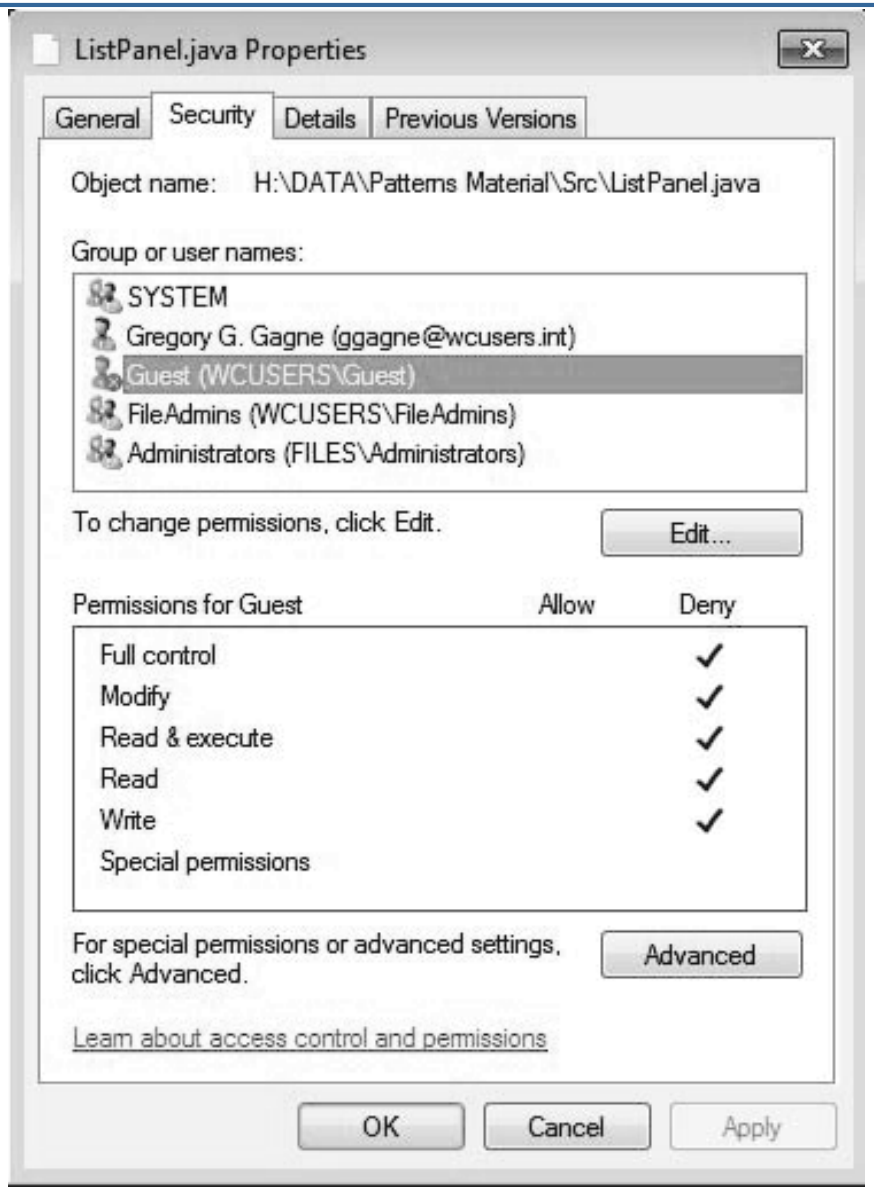
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file : `chgrp G game`



Windows 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/