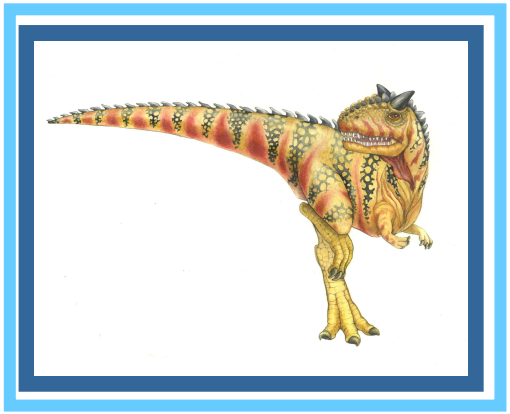
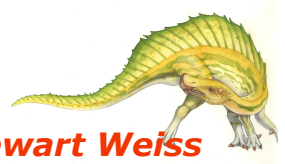
Chapter 10: File-System Interface



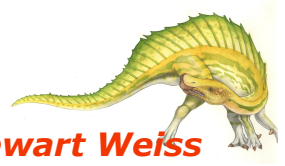
Chapter 10: File-System Interface

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



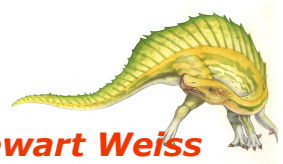
Objectives

- 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



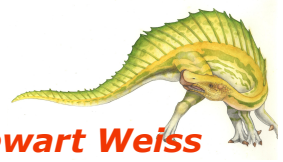
File Concept

- From user view, a file is the smallest unit of storage that can be created on a secondary storage device.
- From system view, a file is a sequence of bytes that may or may not have structure.
- Some OS define different file structures; some treat all files as unstructured and leave interpretation of structure to user-level programs. (UNIX – minimal OS structuring; Windows – much OS interpretation)
- All generally distinguish between
  - data
    - text (newlines, character encodings)
    - binary (bytes not are not character, use all 8 bits)
  - executable



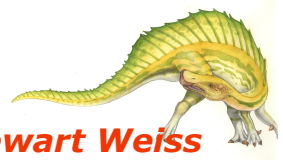
File Structures

- None - sequence of bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Structure defined by standard (e.g., jpeg, tiff, mpeg, pdf)
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program



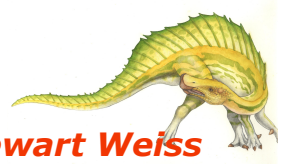
File Structures

- Executable files must always have structure that the OS supports – e.g., ELF in UNIX
- Libraries must have structure – DLLs in Windows, .so and .sa files in UNIX
- Directories are structured files – OS parses directories to maintain them
- Some OS (Mac OS) requires files to have resource fork and data fork. Resource fork contains resources that can change depending on user settings, data fork is constant.
- In general, the more the OS supports different structures, the more complex the OS, but the easier for applications.



File Attributes

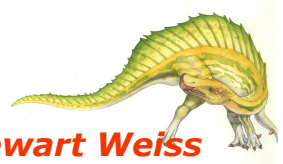
- **Name** – human readable identification
- **Identifier** – unique number that identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – information about which device file is on, and where file is stored on device
- **Size** – current file size (maybe in both bytes and device units, e.g. blocks)
- **Protection** – controls who can access file and how (read, write, execute, modify attributes, etc)
- **Time, date, and user identification** – timestamps (when created, when data modified, accessed, when attributes modified, etc), owner, group, etc
- Information about files are kept in the directory structure, which is maintained on the disk. Might be linked lists, or table.



Required File Operations

- Six essential operations defined for files:

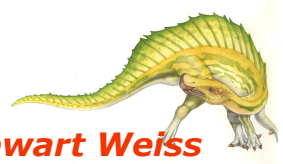
1) **Create –** create a new file, giving it a name; usually some minimum number of bytes is allocated to it
2) **Write –** add data to file; usually OpSys maintains write-pointer (wp) and data is written to current wp. Call minimally specifies buffer containing data to be written, file ID, sometimes more.
3) **Read** – read data from file; usually OpSys maintains read-pointer (rp) and call reads from current rp. Call specifies file ID, memory buffer to put data. Sometimes rp and wp are a single per-process pointer.
4) **Reposition within file (seek) –** move the rp or wp to a specified location within file. Call specifies location, either absolute or relative, file ID.



Required File Operations

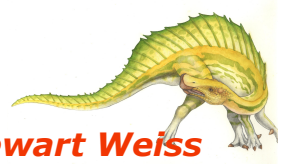
5) **Delete –** from the OpSys perspective, remove file and all of its contents from device on which it is stored. Also removes directory entry on device. On some OpSys, e.g. UNIX., processes do not delete files; they remove the names of the files from the parent directory and nothing more. When a file has no names in any directories, the OpSys deletes the file.

6) **Truncate –** zero the file contents but keep the file in existence. Allows a file to remain in existence so timestamps are preserved as well as protections, etc. while deleting all data.



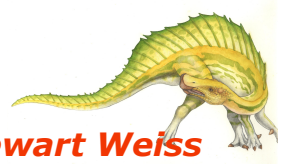
Other File Operations

- Other common operations include:
  - renaming files, copying files,
  - appending to files,
  - creating new "links" to file
- In addition, operations that change attributes of files such as protection and ownership are often supported by operating systems.



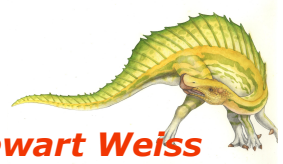
Opening and Closing Files

- Most OpSys's require a file to be opened in order to read, write, or seek. Some do automatic open() on first reference.
- The purposes of opening a file are to
  - allow the operating system to control which processes access the file,
  - facilitate operations that access or modify the file's data,
  - make reading, writing and seeking more efficient, and
  - utilize memory better.
- The ***open*** operation locates the specified file, creates an in-memory structure to facilitate access to the file, and returns a pointer to that structure that can be used for subsequent file operations. The open often allows specification of the mode – e.g., reading, writing, appending, reading+writing. Sometimes specifies whether buffered, unbuffered, synchronous, asynchronous, etc.
- A ***close*** operation undoes the work of open.



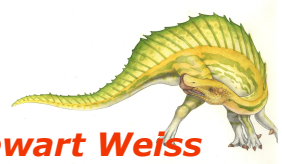
Open File Data

- When a file is opened, various information must be stored in an easily accessible way in memory:
  - Storage location of the file: which device and where on device are the bytes of the file
  - Location of file attributes – owner, timestamps, etc.
  - Location in memory of temporary buffers for reads/writes to/from file
  - Access mode – reading/writing/both/append
  - Transmission mode – synchronous/asynchronous
  - File pointers – "read head", "write head"
  - Reference count – how many times is the file open



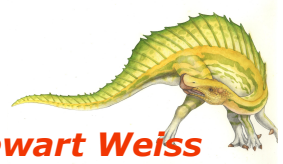
Per-Process Open File Data

- In a system in which only one process can open a file at a time, structures are simple; all data is stored in a single system table, an ***open file table***.
- More common: system allows files to be open multiple times. In this case, system separates data into ***per-process data*** and ***system-wide data***.
- Each process can have its own access mode, own file pointers, own buffers for data transfers.
- Location of file, file properties, etc is system-wide
- Two tables: System-wide Open File Table, Per-process Open File Table
  - Per-process table is in process address space
  - System-wide table is in kernel memory



Further Distinctions

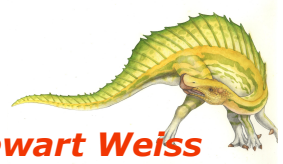
- UNIX allows related processes and threads to share open files, so they share read pointers for example, but unrelated processes cannot share open files.
- So UNIX puts file pointers in system-wide table, and per-process tables of related processes point to same system table entry, whereas per-process tables of unrelated processes point to different system table entries.



Open File Locking

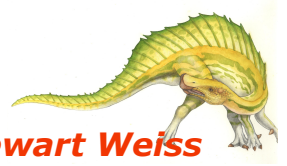
- Provided by some operating systems and file systems
- Mediates access to a file
  - Exclusive lock – only one process can hold the lock (like a writer lock)
  - Shared lock – multiple processes can hold lock (like a reader lock)
- Mandatory or advisory:
  - **Mandatory** – operating system enforces access – e.g. if exclusive lock is held by process, no other process will be allowed to acquire lock
  - **Advisory** – operating system does not enforce access -- processes can find status of locks and decide what to do

UNIX locks usually advisory; Windows, mandatory



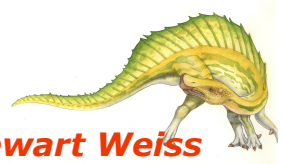
File Locking Example – Java API

```
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();
```



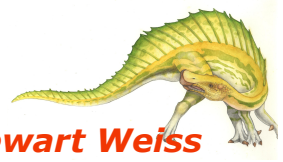
File Locking Example – Java API (cont)

```
                    // 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();
            }
        }
    }
```



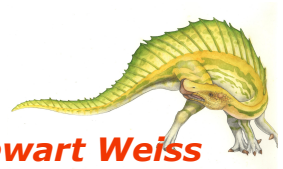
File Name Extensions

- Some operating systems identify file types by extensions such as pdf, jpg, exe, and so on.
- MS-DOS identifies several extensions and uses them explicitly. Windows adds functionality through GUI to more extensions through the registry.
- Macintosh OS files have a creator and a file type stored with the file, each a 4-character field. Applications register their 4-char creator fields with OS, and type fields are standardized. New types are registered with Apple.
- UNIX does not use extensions or types. It is a user-level feature. GUIs add functionality of extensions.



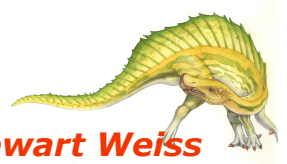
File Types – Name, Extension

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

- Method by which file is accessed
  - Sequential : data is read starting from beginning, one record after another in sequence, like reading the contents of a tape. Very restrictive but requires little support.
  - Direct Access (relative access) : data records mustbe fixed size and can be accessed anywhere within the file, like accessing the bytes on a disk. More useful but requires more OS support
- Secondary Access Methods are built on top of direct access methods:
  - Indexed files: index file that can be searched by key with pointers to direct access file, called the relative file
  - Indexed sequential access method (IBM's ISAM), three-level structure: master index is a table that points to sorted index file on disk, which has pointers to relative file, which is also sorted.

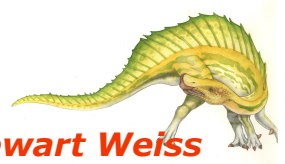


Access Operations

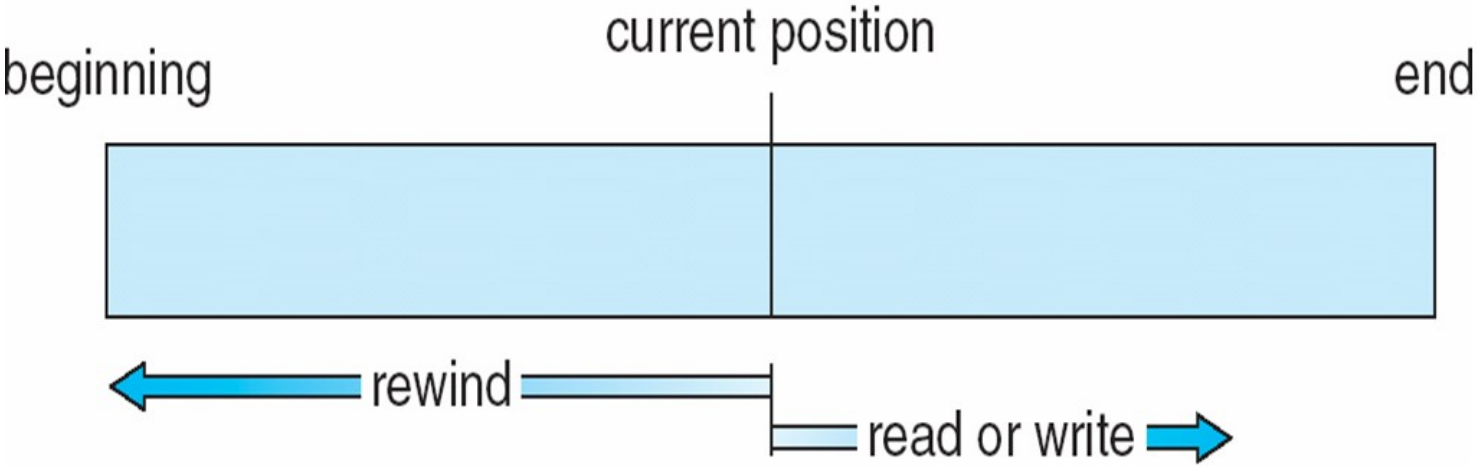
- **Sequential Access**
  read next
  write next
  rewind (move pointer to beginning)
  seek (move pointer to a specific position sequentially)

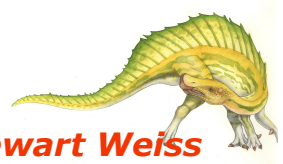
- **Direct Access**
  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  n = block number relative to first block



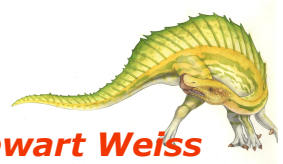
Sequential-access File



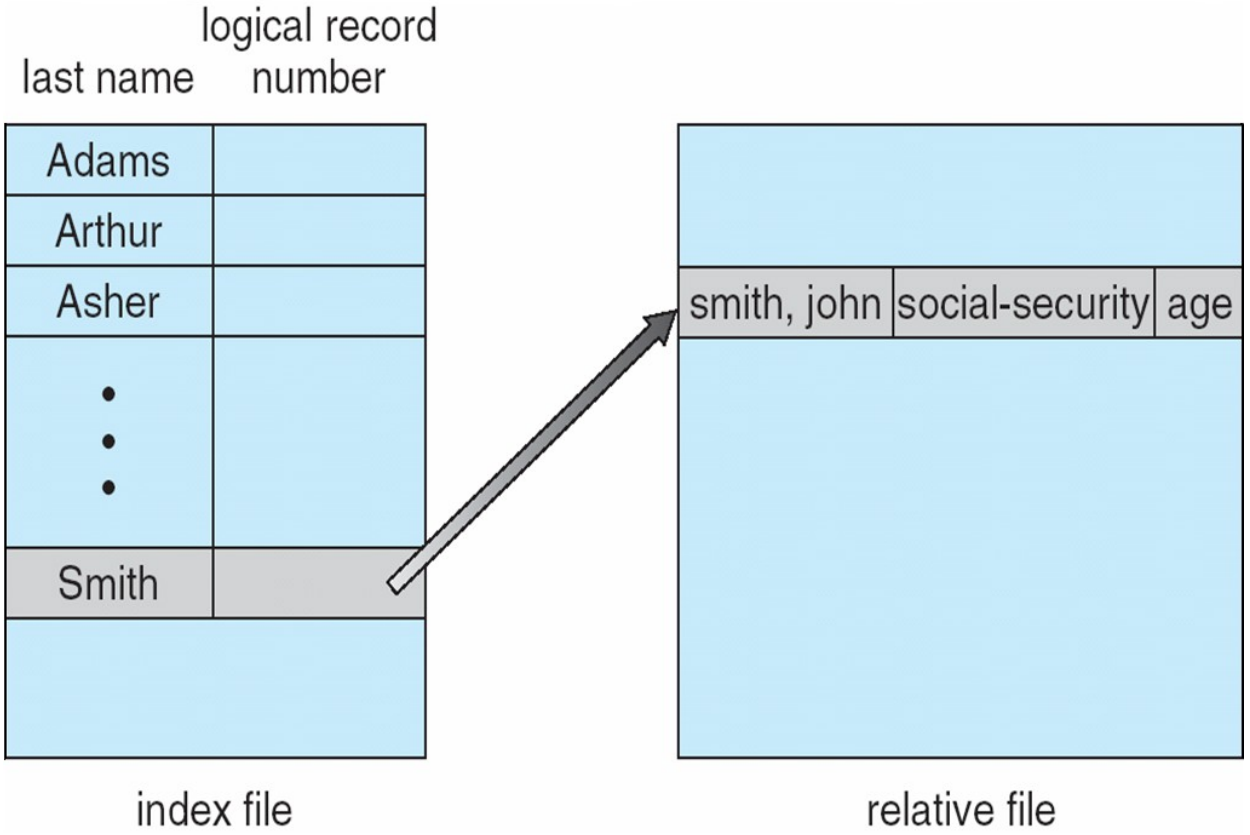


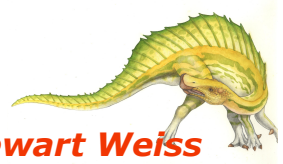
Simulation of Sequential Access on Direct-access File

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



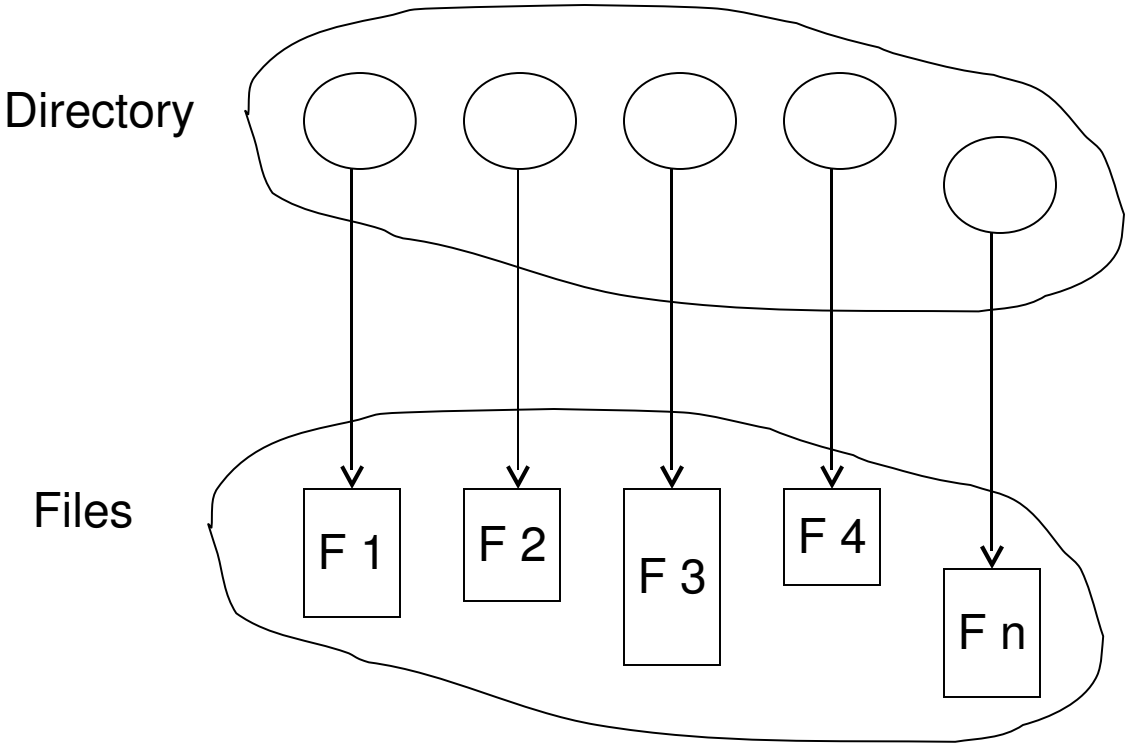
Example of Index and Relative Files



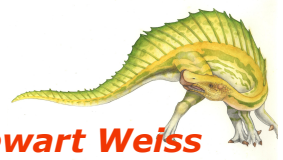


Directory Structure

- A collection of nodes containing information about all files

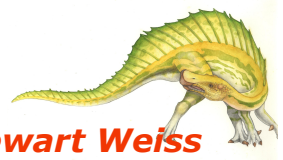


Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

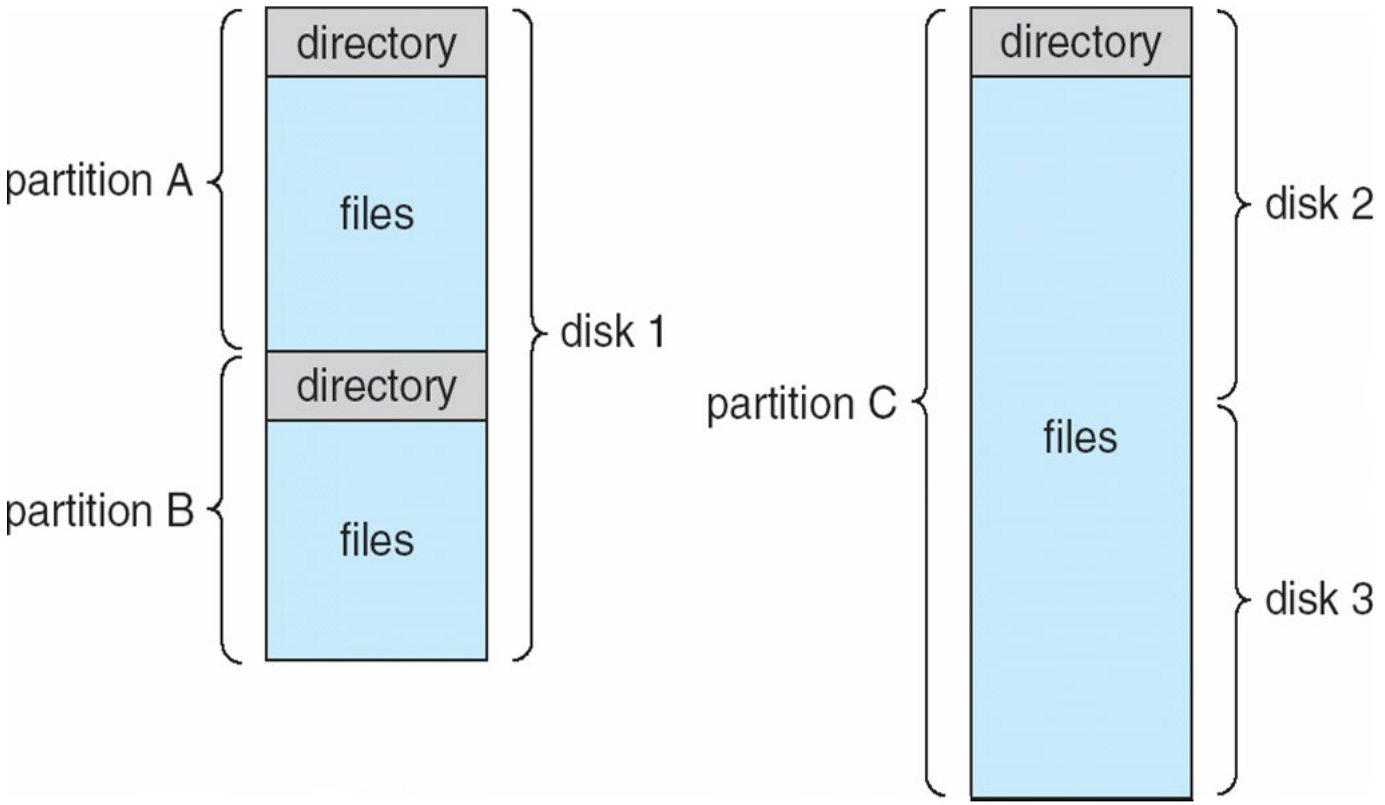


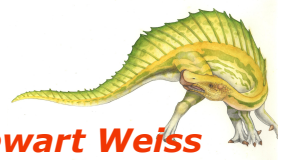
Disk Structure

- 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 (by IBM), 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
- UNIX uses an ***inode table*** as an index of where files are within a single file system.
- As well as general-purpose file systems there are many special-purpose file systems, frequently all within the same operating system or computer



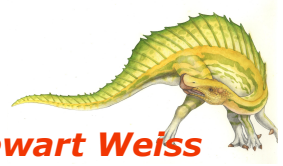
A Typical File-system Organization





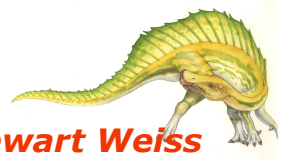
Operations Performed on Directory

- A user-level directory can be viewed as a table that maps file names to the actual files in the file system. Files on disk can be represented by indices into the entries in the device directory. Operations on directories include
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system
  - Create a new name for an existing file
  - Remove a name for a file; if no more names delete file

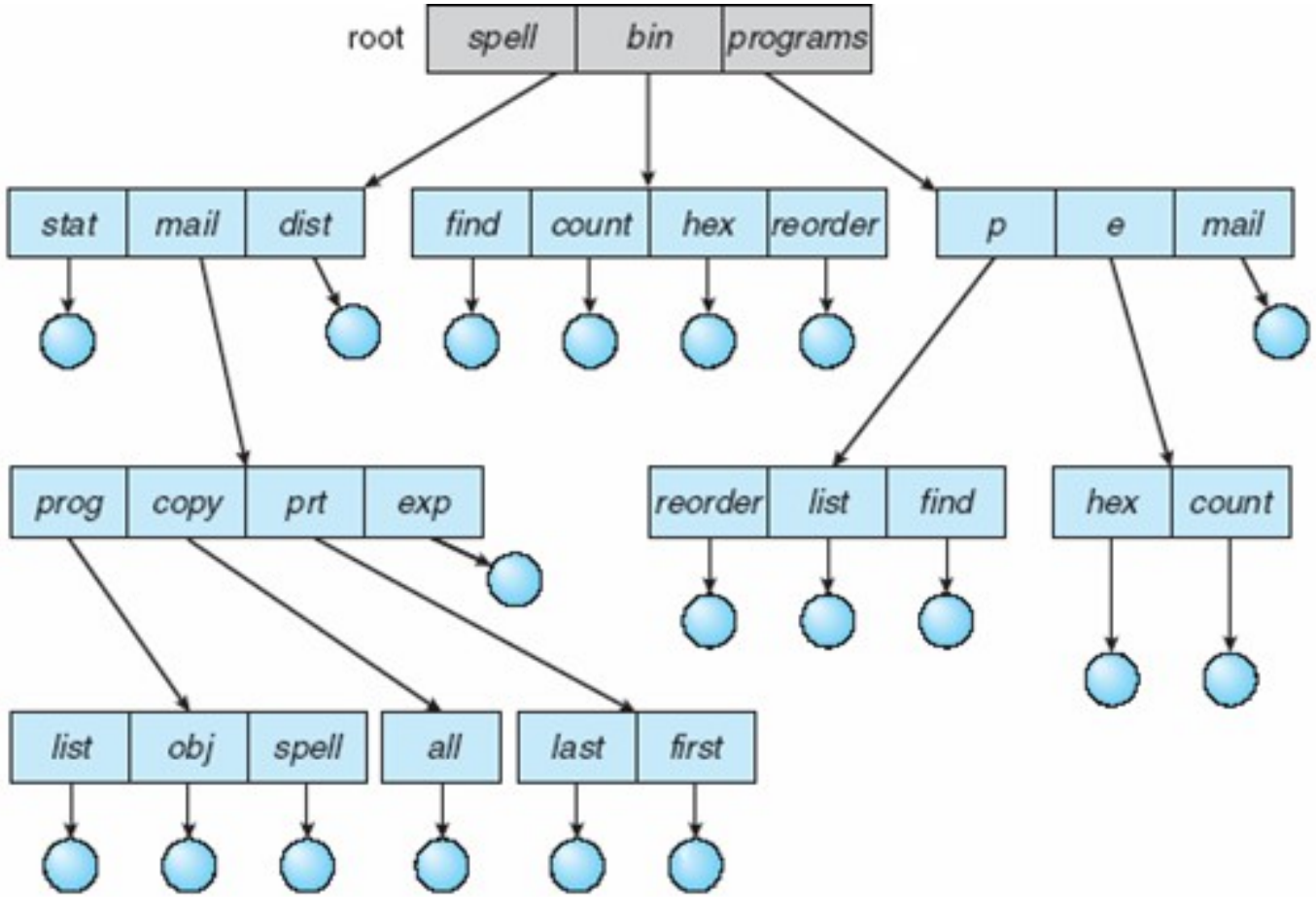


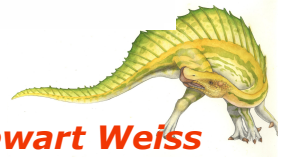
Organize the Directory (Logically) to Obtain

- 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 (called links in UNIX)
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)



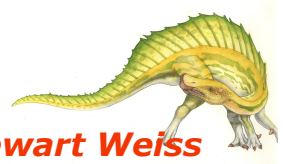
Tree-Structured Directories





Tree-Structured Directories (Cont'd.)

- Allows efficient searching
- Provides grouping Capability
- Concept of Current working directory



Tree-Structured Directories (Cont'd.)

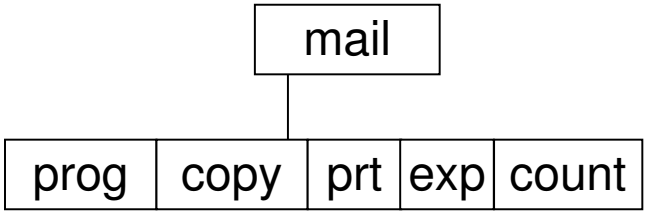
- **Absolute** or **relative** path name
- Delete a file

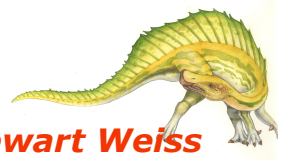
  rm <file-name>
- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

  Example: if in current directory /mail

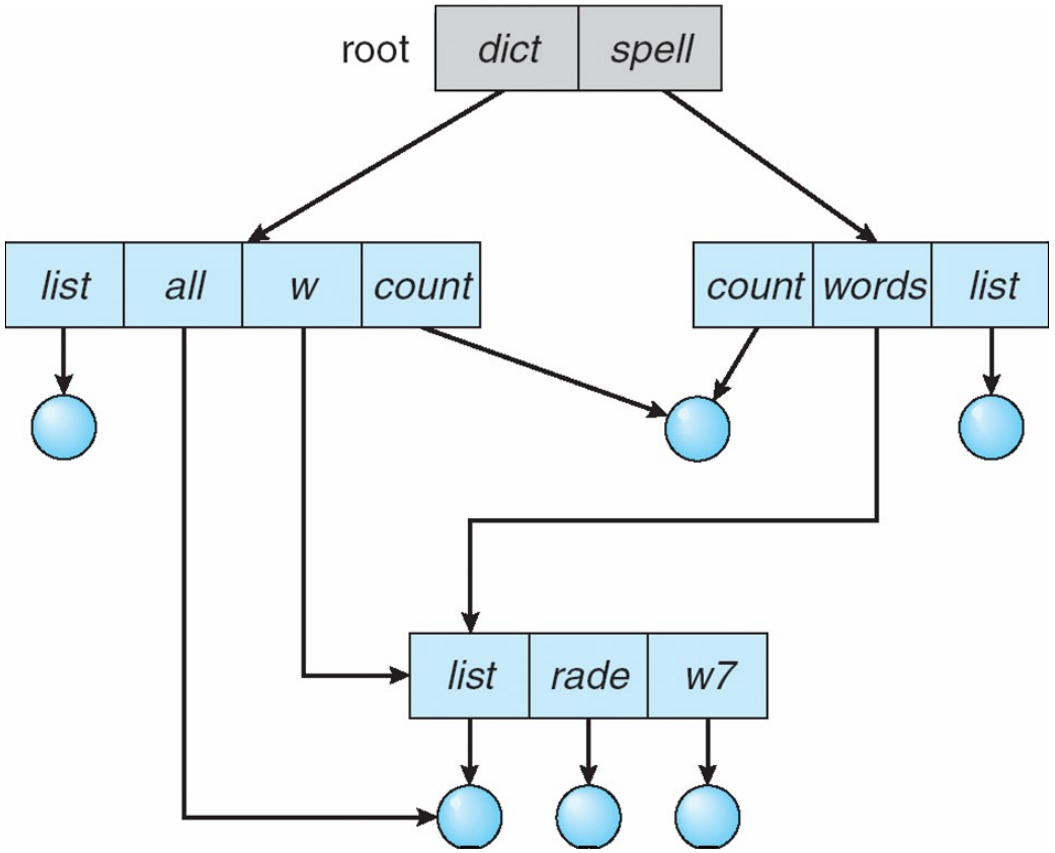
  mkdir count
- Deleting directory – delete entire subtree or require directory to be empty

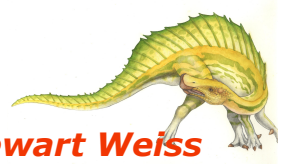




Acyclic-Graph Directories

- Have shared subdirectories and files

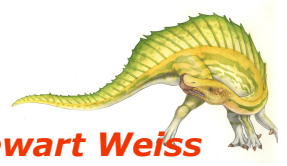




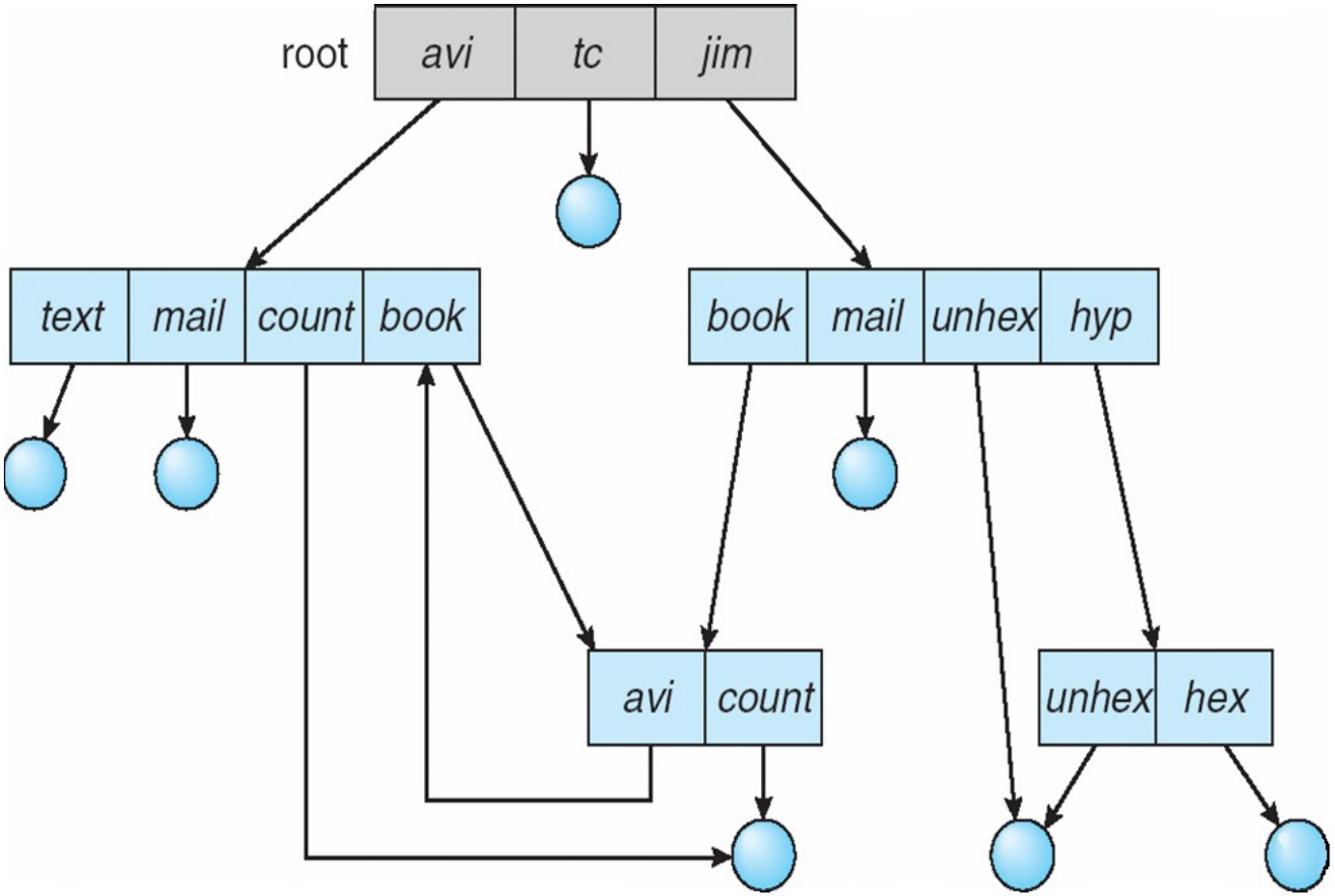
Acyclic-Graph Directories (Cont.)

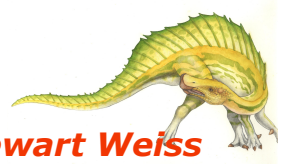
- Two different names (aliasing)
- If *dict* deletes *list* ⇒dangling pointer

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file



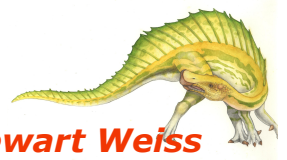
General Graph Directory





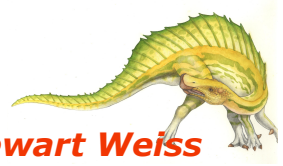
General Graph Directory (Cont.)

- 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 to determine whether it is OK

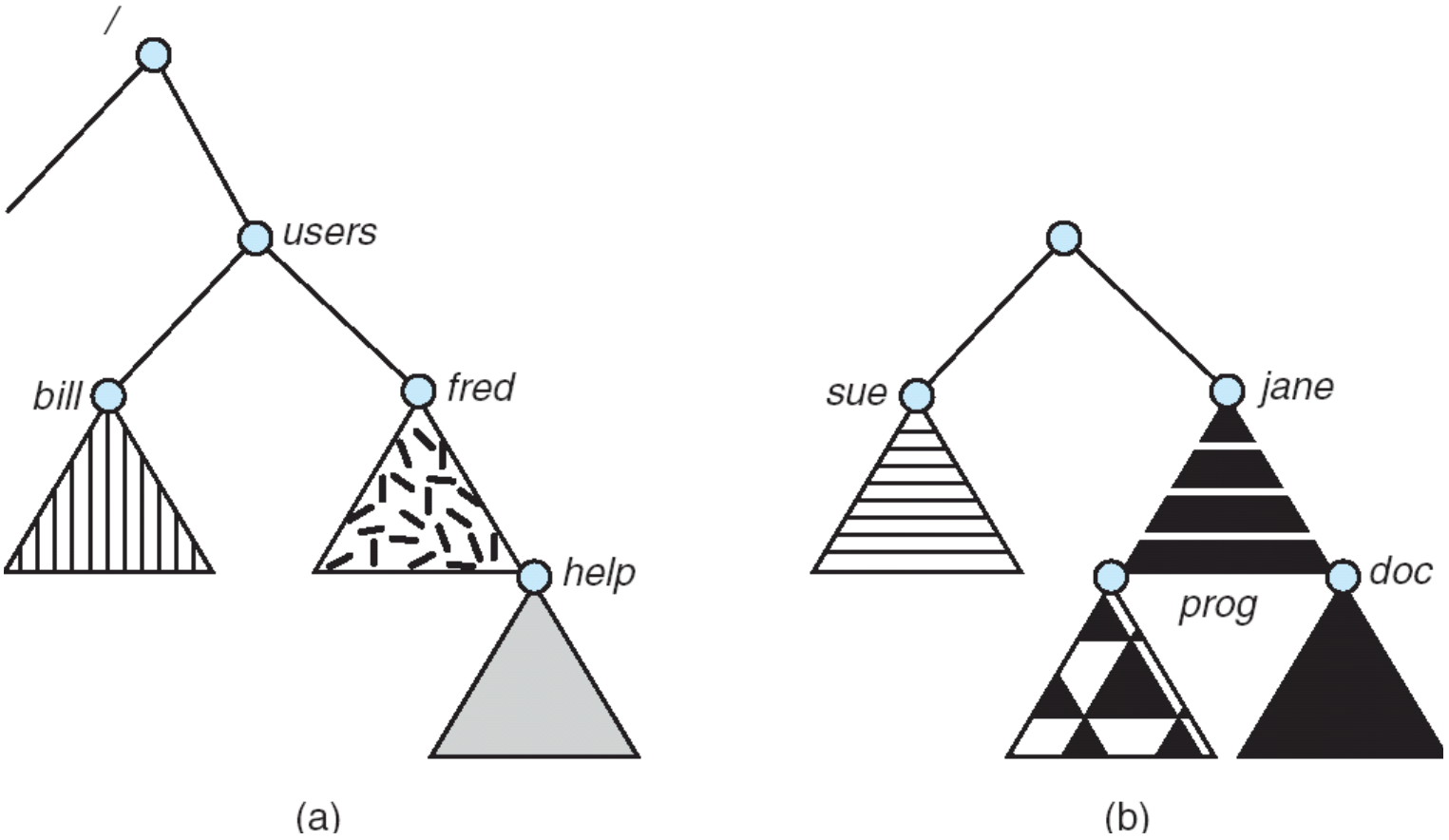


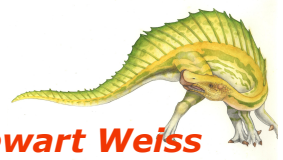
File System Mounting

- 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 mount point is a directory in the existing file system.
- Explicit and auto-mounting possible:
  - explicit : mount(/dev/disk1, /data/temp);
  - automounting of optical disks, flash drives etc



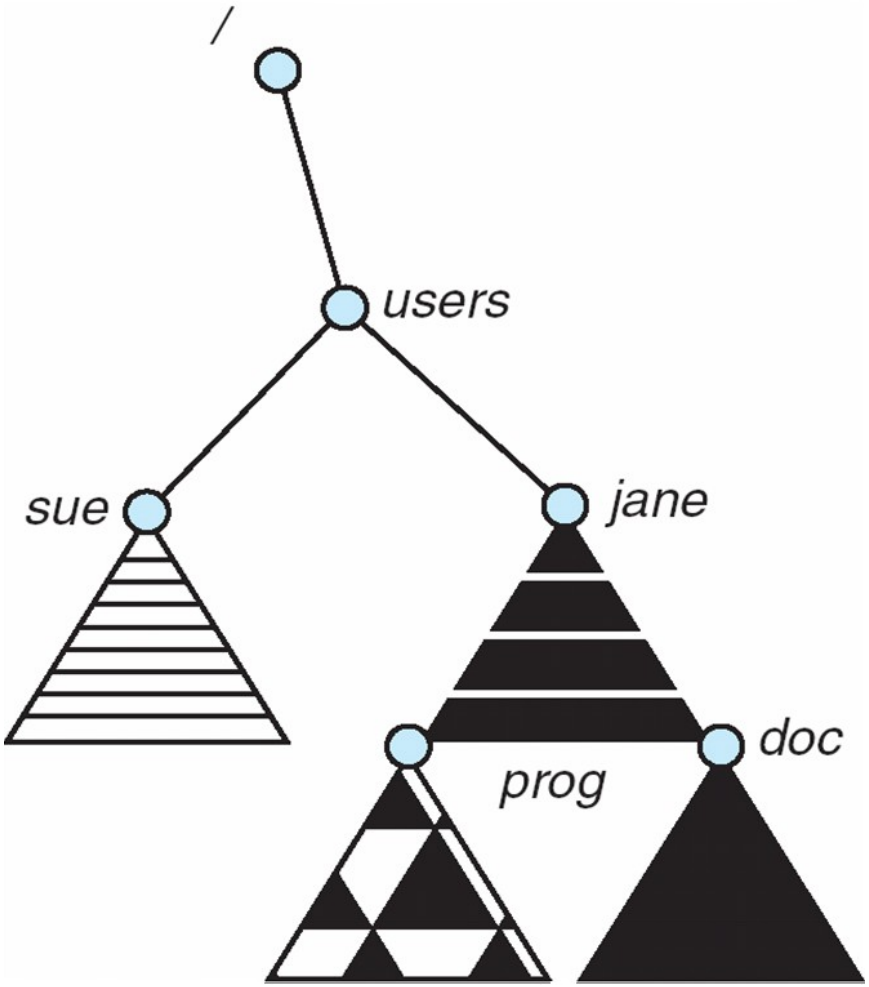
(a) Existing. (b) Unmounted Partition

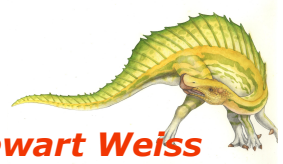




modified by Stewart Weiss

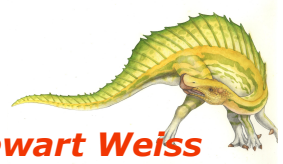
Mount Point





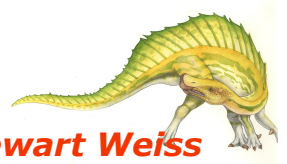
File Sharing

- 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
- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights
- Windows supports access rights and more general method of identifying users.



Protection

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

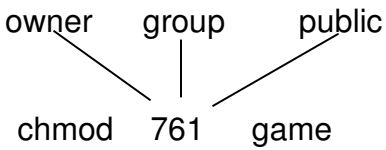


Access Lists and Groups

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

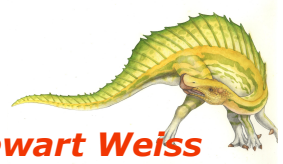
			RWX
a) **owner access**	7	⇒	1 1 1 RWX
b) **group access**	6	⇒	1 1 0 RWX
c) **public access**	1	⇒	0 0 1

- 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.



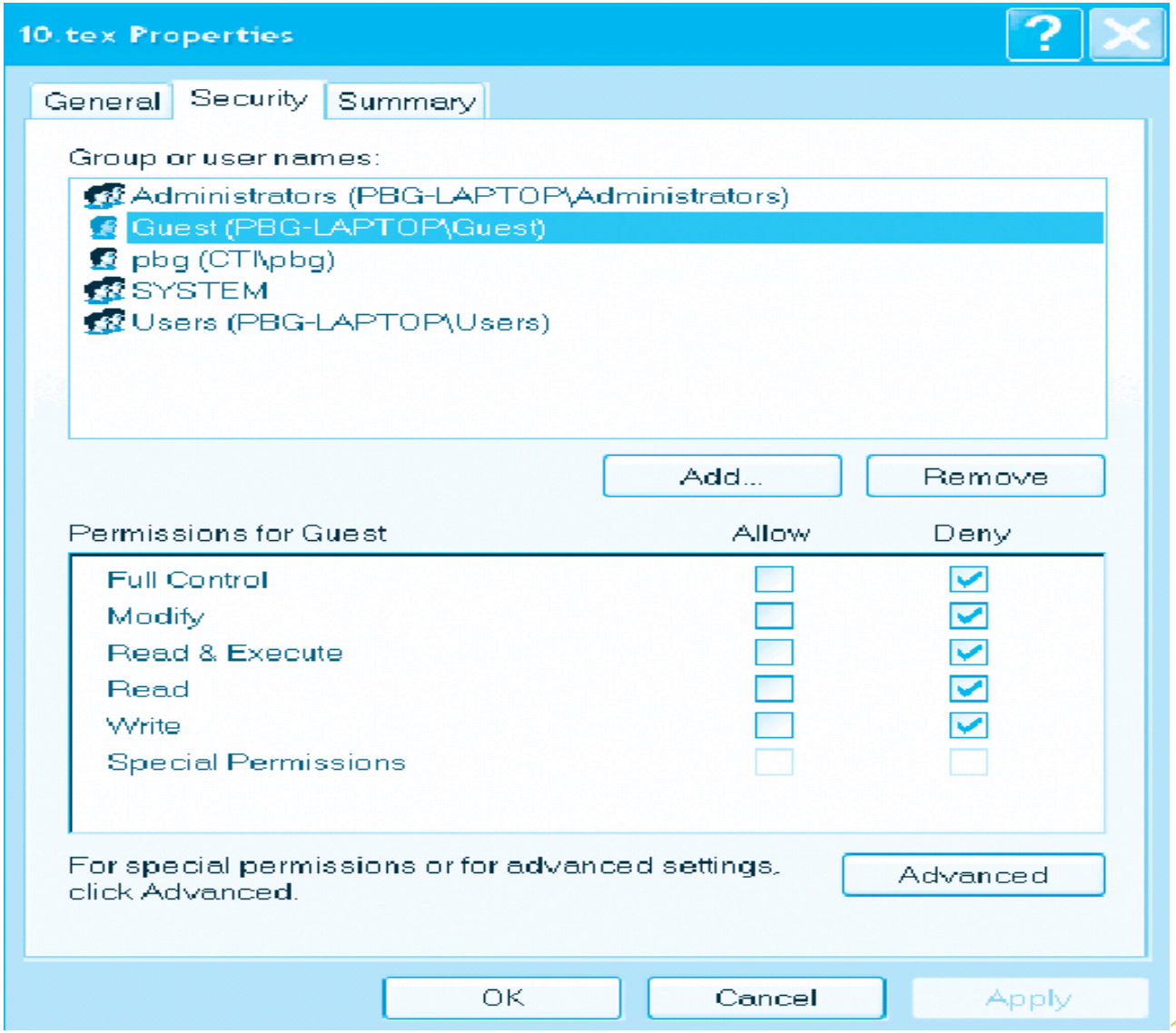
Attach a group to a file

chgrp G game



modified by Stewart Weiss

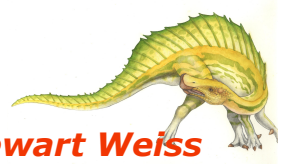
Windows XP Access-control List Management



modified by Stewart Weiss

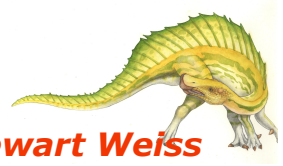
A Sample UNIX Directory Listing

-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/



More about Protection

- Windows access control lists attach allow and deny lists to objects. Each list can specify what operations are allowed or denied. More extensive than the simple UNIX model – operations can include changing permissions, reading attributes, taking ownership, modifying attributes.
- Some OS now combine traditional UNIX model with use of access control lists like the ones used in Windows, e.g. Solaris, SE Linux – access control lists can be attached to files optionally.
- Some allow attaching passwords to each file.



End of Chapter 10