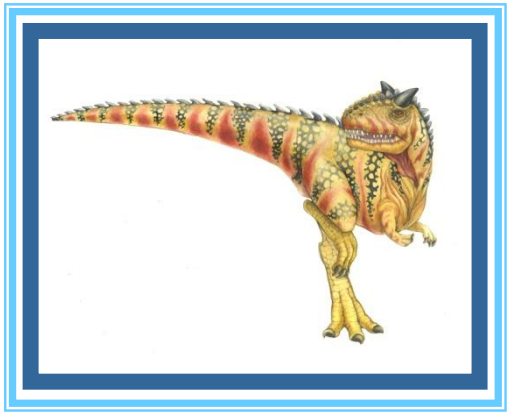
Chapter 10: File-System Interface



Chapter 10: File-System Interface

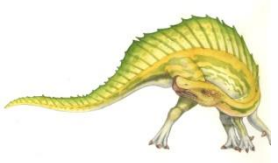
- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection

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

- A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device.
- The file system consists of two distinct parts: a **collection of file**s, each storing related data, and a **directory structure**, which organizes and provide information about all the files in the system.
- A file is a collection of logically related entities. It is a **logical storage unit**
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program: Source file, executable file

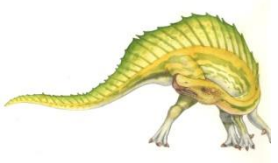


File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load 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 the directory structure, which is maintained on the disk



File Operations

File is an **abstract data type.** The operating system can provide system calls to create, write, read, reposition, delete, and truncate files

- **Create**
- **Write**
- **Read**
- **Reposition within file,** also known as file **seek**
- **Delete**
- **Truncate:** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged.
- *Open(F_i)* – search the directory structure on disk for entry F_i, and move the content of entry to memory
- *Close (F_i)* – move the content of entry F_i in memory to directory structure on disk

Open Files

- Several pieces of data are needed to manage open files:
  - **File pointer:** Is a pointer to last read/write location, each process that has the file open. Is unique for each process operating on file
  - **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**: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

Open File Locking

- Provided by some operating systems and file systems. Some operating systems provide facilities for locking an open file.
- File locks allow one process to lock a file and prevent other processes from gaining access to it.
- Mediates access to a file
- A **shared lock** is akin to a reader lock in that several processes can acquire the lock concurrently.
- An **exclusive lock** behaves like a writer lock; only one process at a time can acquire such a lock
- Mandatory or advisory locking mechanism:
  - **Mandatory** – the operating system ensures locking integrity
  - **Advisory** – software developers to ensure that locks are appropriately acquired and released

File Types – Name, Extension

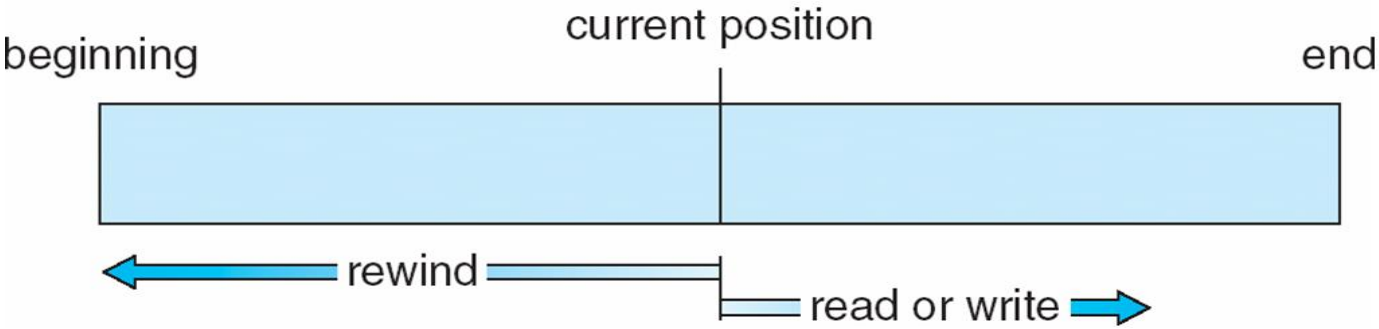
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual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

- Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

Sequential Access: Information in the file is processed in order, one record after the other.

- Reads and writes make up the bulk of the operations on a file.
- **read next()—**reads the next portion of the file
- **write next()—**appends to the end of the file and advances to the end of the newly written material (the new end of file)



Access Methods

Direct Access:

- The direct-access method is based on a **disk model** of a file, since disks allow **random access** to any file block.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on the order of reading or writing for a direct-access file.
- Direct-access files are of great use for immediate access to large amounts of information
- The file operations include the block number as a parameter
- **read(n) and write (n)** where *n* is the block number,
- The block number provided by the user to the operating system is normally a **relative block number**.

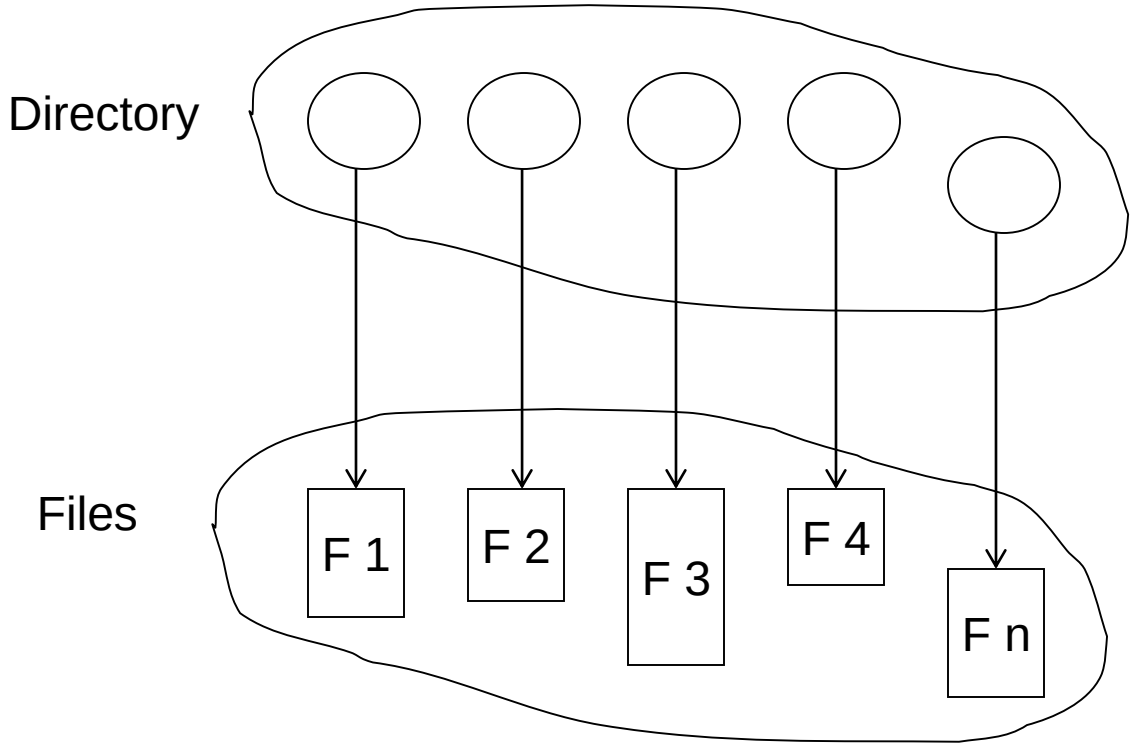
Simulation of Sequential Access on Direct-access File

Simulate sequential access on a direct-access file by keeping a variable *cp* that defines our current position as follows:

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

Directory Structure

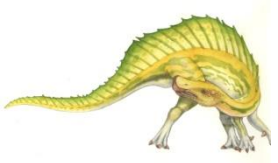
- A collection of nodes containing information about all files



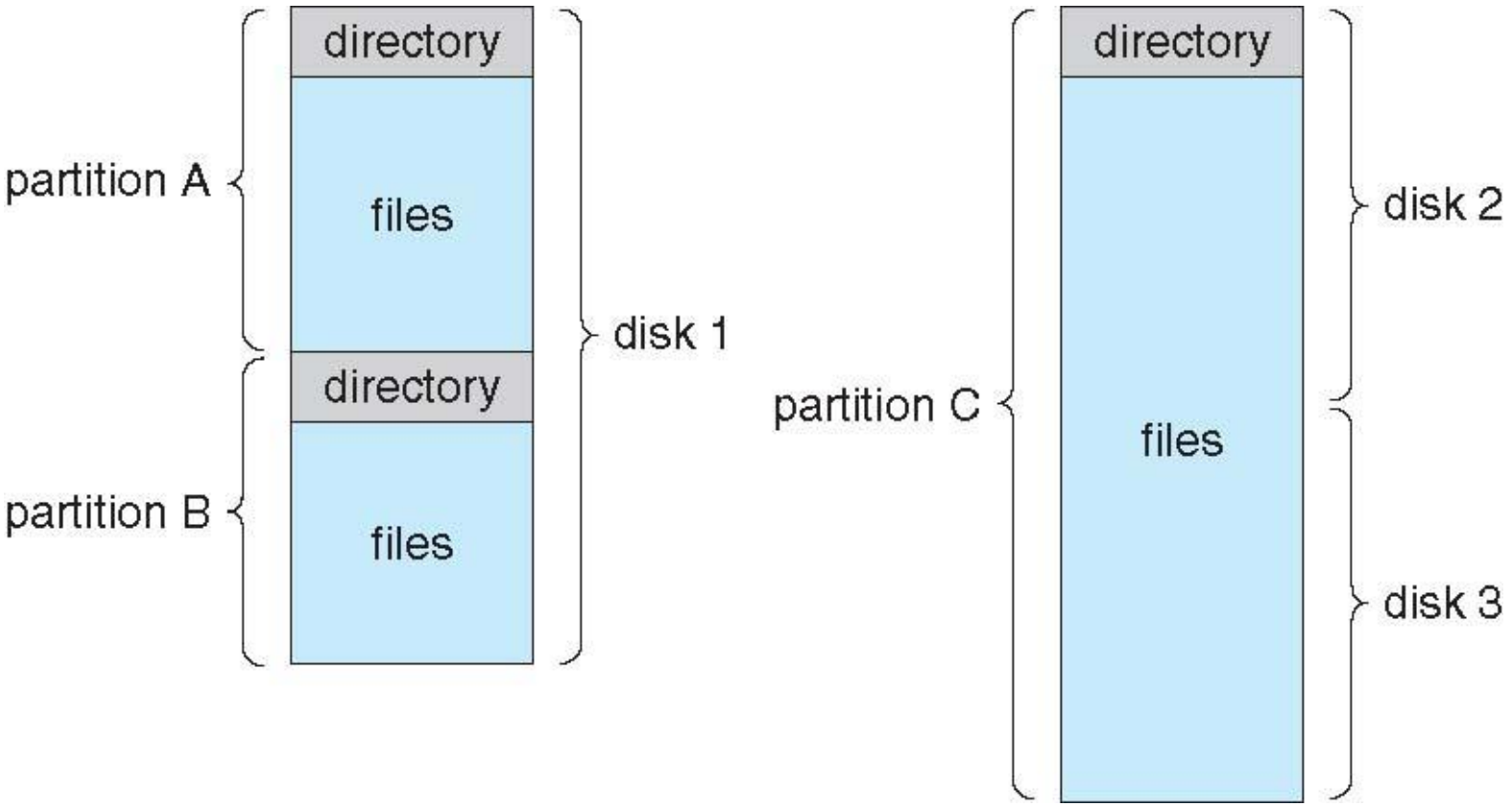
Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

Disk Structure

- Disk can be subdivided into **partitions**
- 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 about file in the system 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



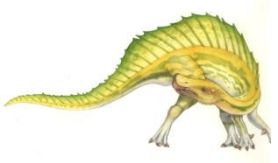
Operations Performed on Directory

The directory can be viewed as a symbol table that translates file names into

- their directory entries.
- 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

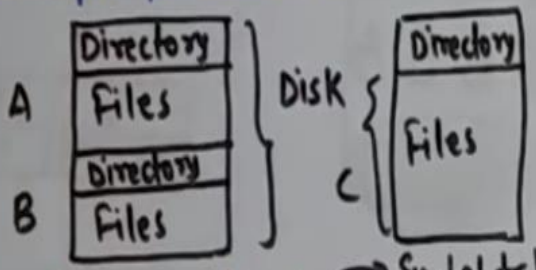
- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)



File Directory

File Directories:

A physical disk can be broken up into multiple partitions, or mini-disks.



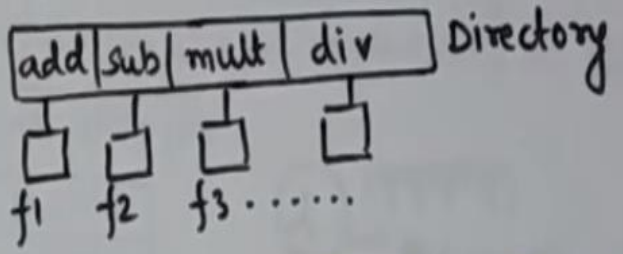
→ Symbol table that translates file name into their directory entries.

Operations on directory:

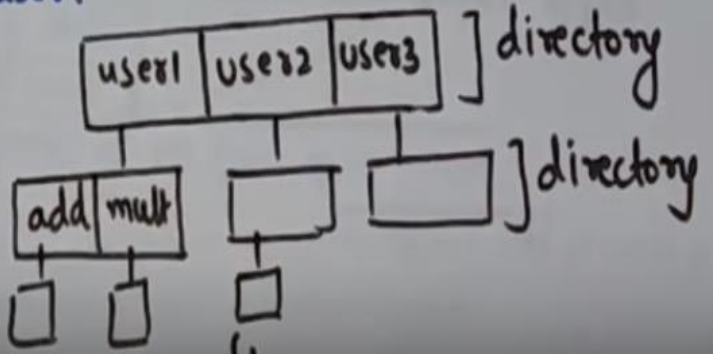
↳ i) Search for a file
ii) Create New file
iii) Delete a file
iv) List a directory
v) Rename a file
vi) Traverse file System

i) Single-Level Directory: All the files are contained in same directory.

→ Each file must have unique name.

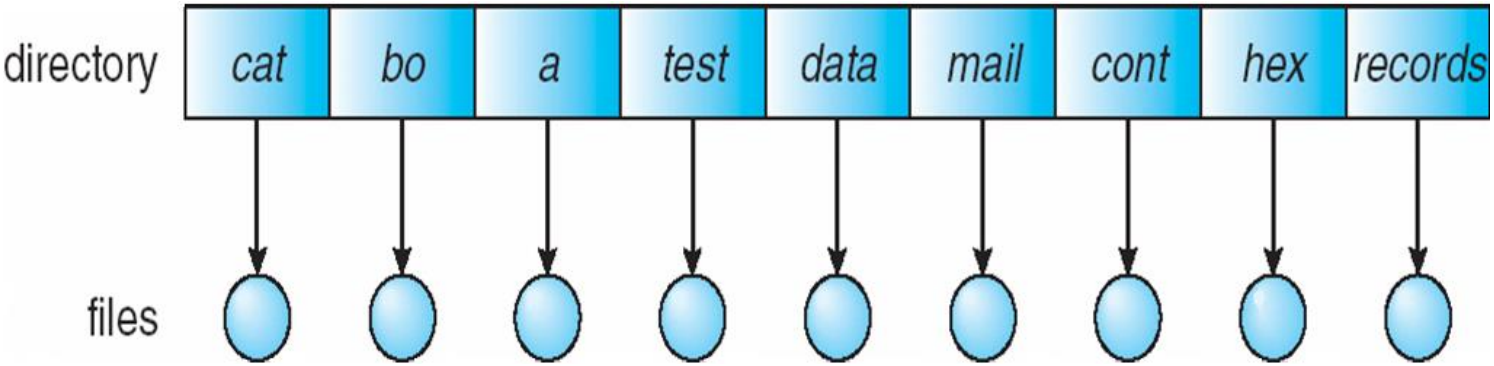


ii) Two-Level Directory: Create a directory for each user.



Single-Level Directory

- A single directory for all users

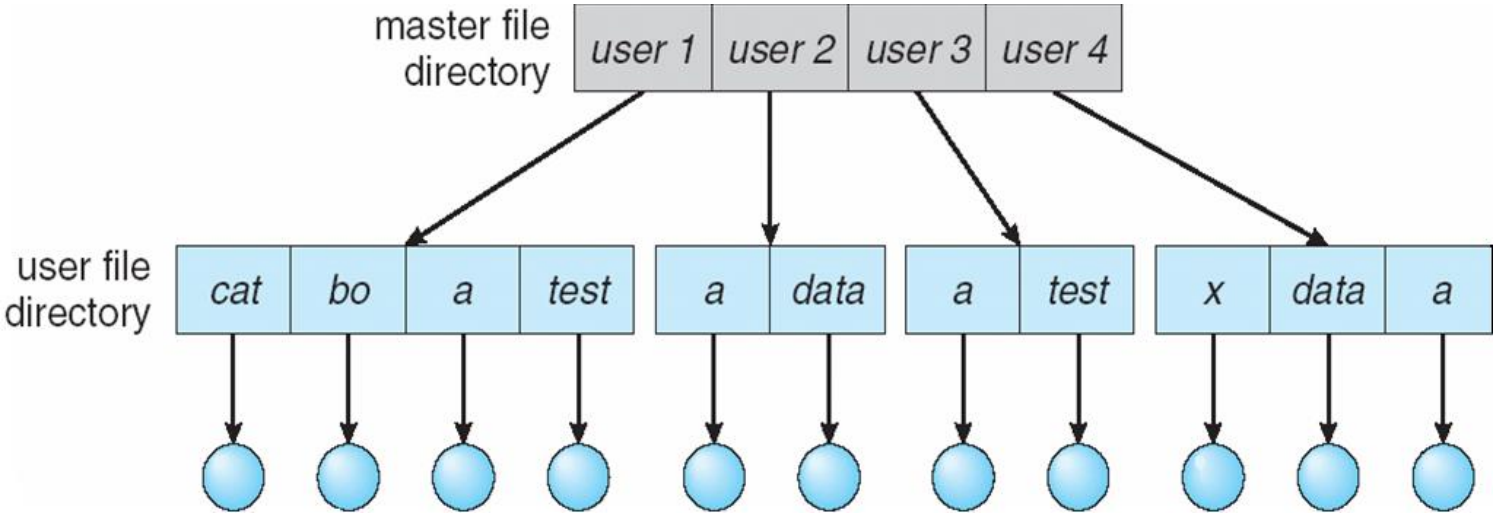


Naming problem

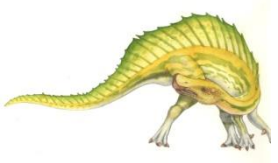
Grouping problem

Two-Level Directory

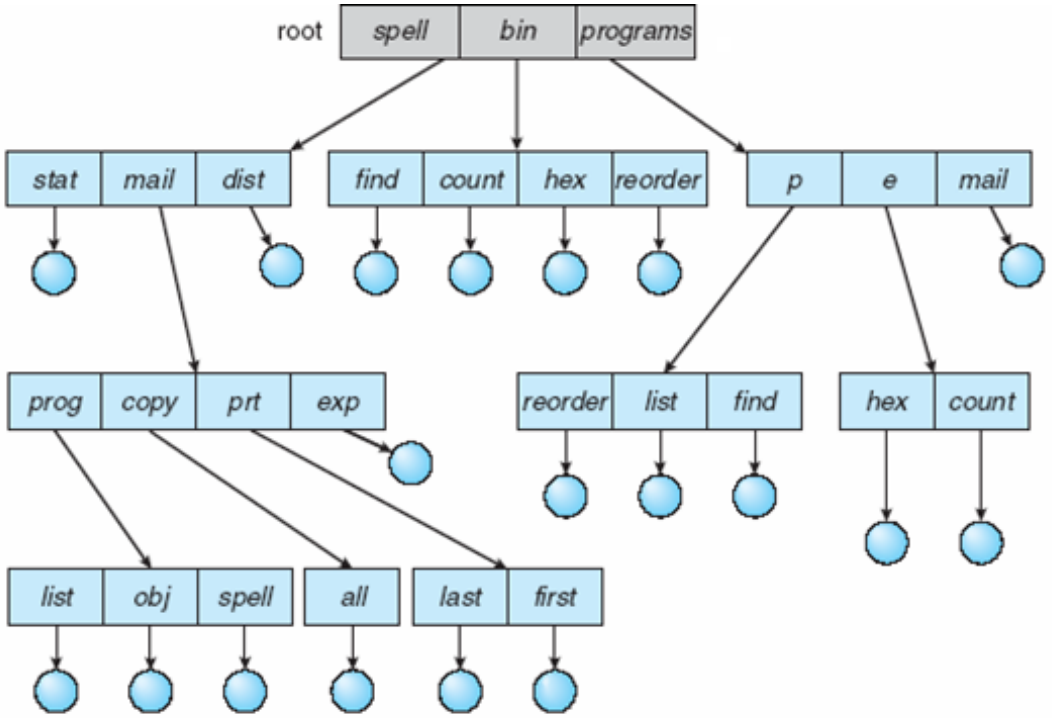
- Separate directory for each user, and each user has **user-file directory(UFD).**
- **UFD** maintained in master file directory **(MFD)** for each user



- Path name
- Can have the same file name for different user
- Efficient searching



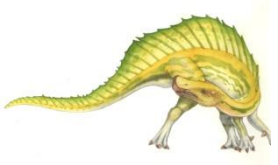
Tree-Structured Directories



Tree Structured Directory. In **Tree structured directory** system, any **directory** entry can either be a file or sub **directory**. ... Searching is more efficient in this **directory structure**. The concept of current working **directory** is used. A file can be accessed by two types of path, either relative or absolute.

Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- User can create subdirectories and organize files
- Every file has a unique path, any user can access the files of other users by specifying the file path.
- Current directory (working directory)
  - **cd /spell/mail/prog**
  - **type list**



Tree-Structured Directories (Cont)

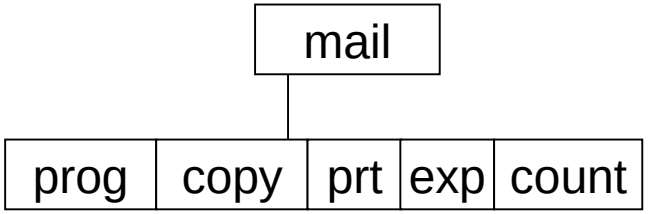
- **Absolute** or **relative** path name exists
- Absolute path **begins at root**, relative path defines path from **current directory**
- Creating a new file is done in current directory
- Delete a file

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

  mkdir <dir-name>

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

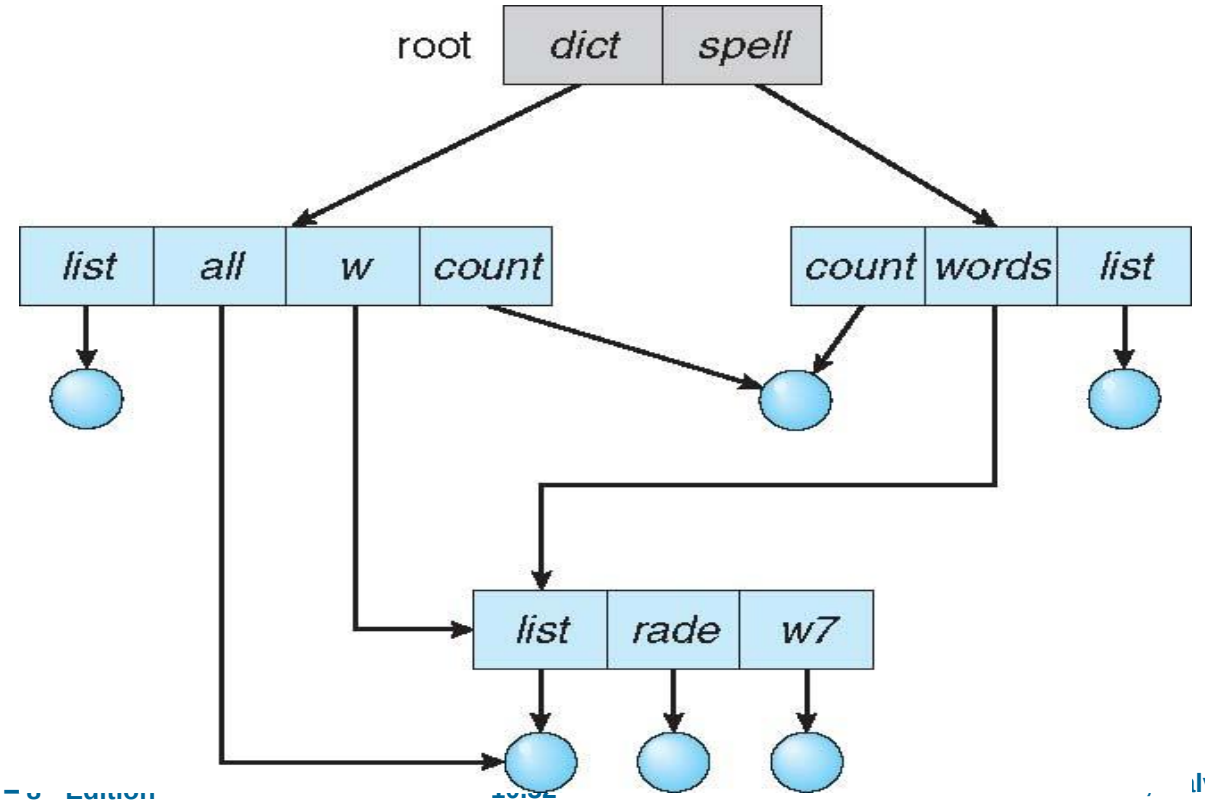
  mkdir count



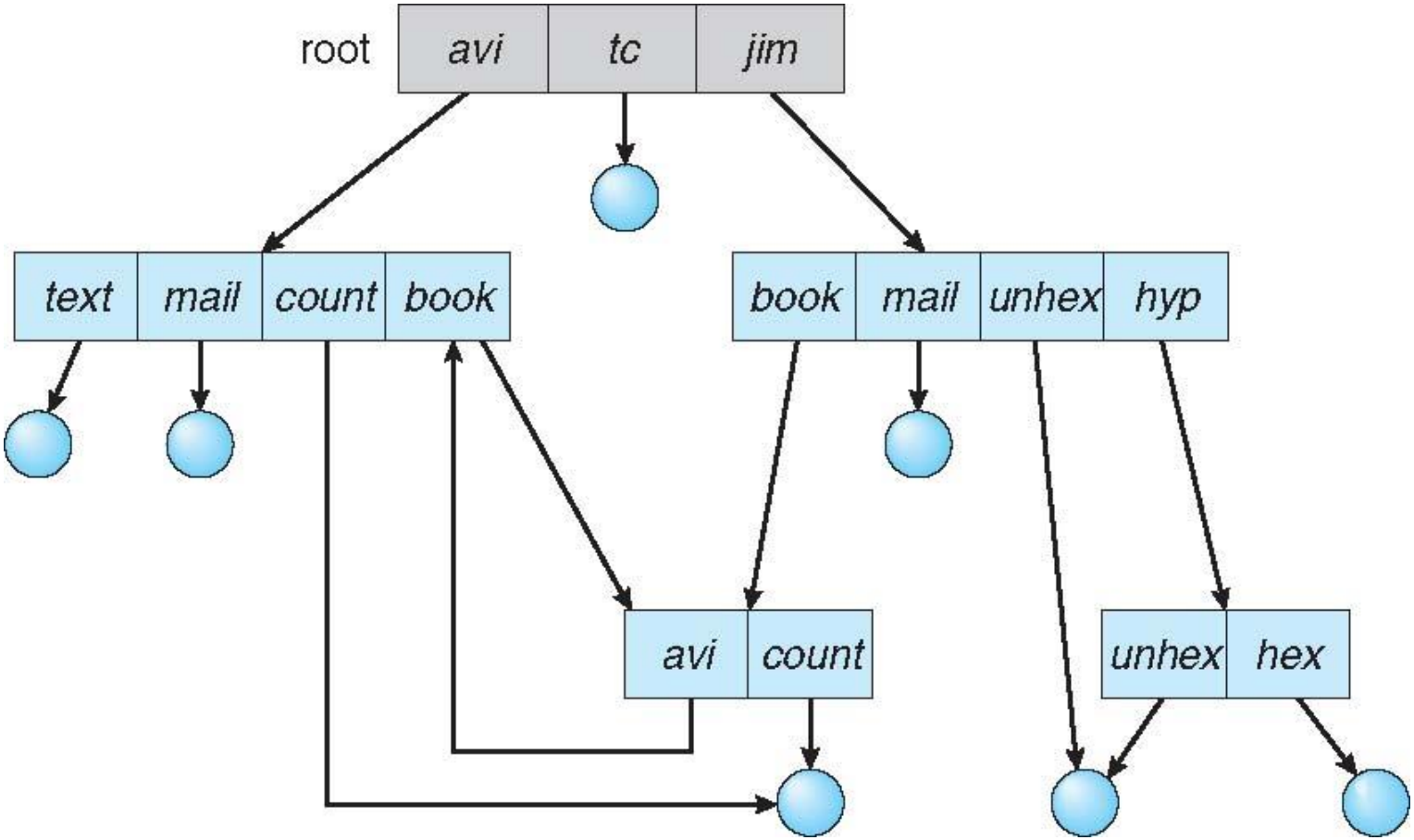
Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

- It is a natural generalization of tree-structured directory, a graph **with no cycles**
- Two or more directory entry can point to the same file or sub directory.
- File or sub directory is shared between the two directory entries.
- If any user makes a change, it would be reflected to both the users.



General Graph Directory



General Graph Directory (Cont.)

- when we add links, the tree structure is destroyed, resulting in a simple graph structure(Cyclic graph)
- Here a search could be very complex leading to infinite searching in cycle
- An issue exist when a file can be deleted in presence of cycle
- How do we know when a new link will complete a cycle?
  - Use algorithms to detect cycles in graphs
  - Allow only links to file not subdirectories
  - Garbage collection

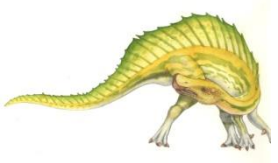
  Thus, an acyclic-graph structure is much easier to work with

File System Mounting

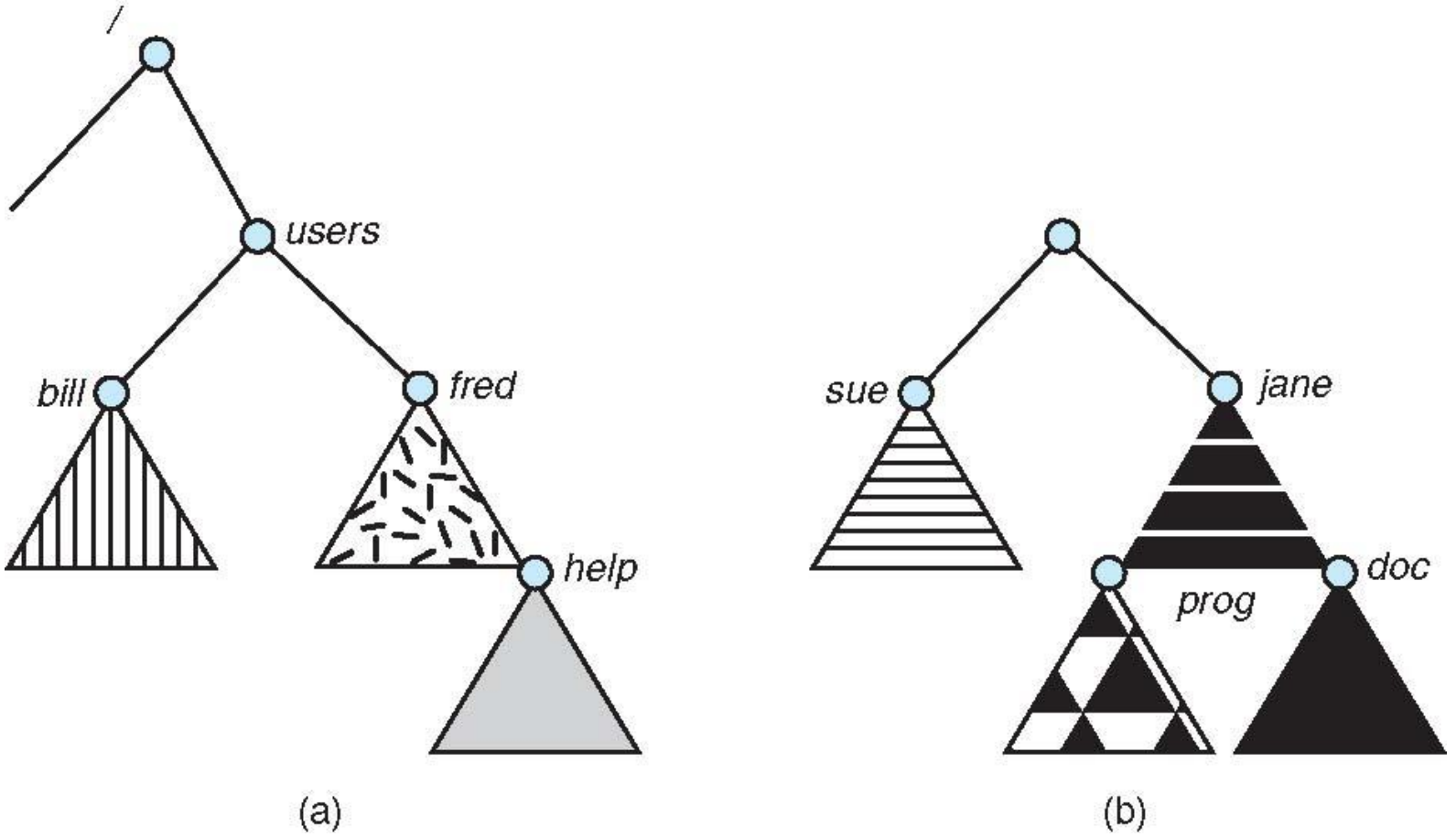
- File mounting is the process of associating a storage device or a partition with a mount point in the file system hierarchy of an operating system.
- **Mount point**: It is an empty directory in which we are adding the file system during the process of mounting.
- A file system must be **mounted** before it can be accessed

Example:

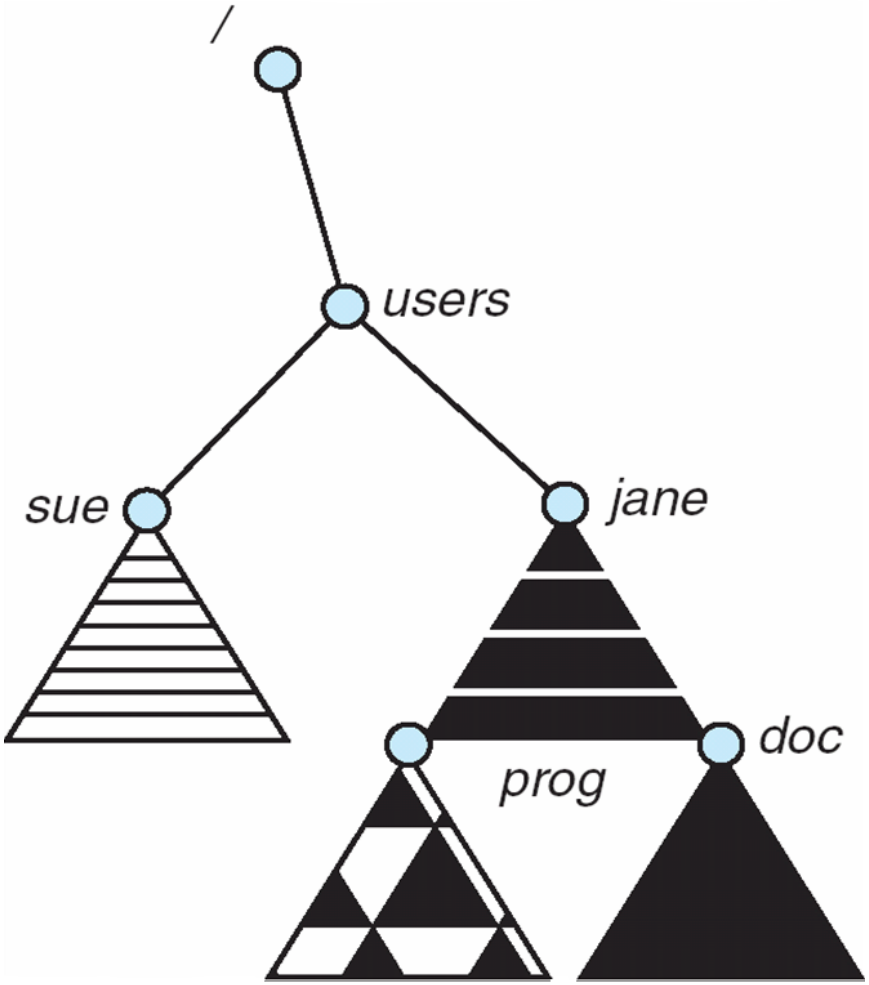
- In windows when we connect the external storage devices, windows automatically detect the file system and mount it to the drive letter. Drive letter may be **D:** or **E:**.
- In MAC OS Whenever the system encounters a disk for the first time it searches for a file system on the device.
  - If it finds one, it automatically mounts the file system under the **/Volumes directory**, adding a folder icon labeled with the name of the file system

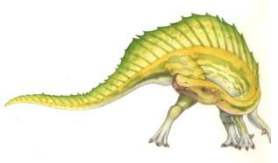


(a) Existing (b) Unmounted Partition



Mount Point





End of Chapter 10