# UNIT-V
# Storage Management

Mr. S. C. Sagare

# File-System Interface

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

# CHAPTER 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.

# Introduction

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks
- Operating system provides a uniform logical view of stored information so that the computer system will be convenient to use.
- Operating system abstracts from the physical properties of its storage devices to define a **logical storage unit, the file**. Files are mapped by the operating system onto physical devices.
- These storage devices are usually nonvolatile, so the contents are persistent between system reboots.

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file

- Commonly, files represent programs (both source and object forms) and data.
- Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files may be free form, such as text files, or may be formatted rigidly.
- In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user
- Thus the concept of a **file** is extremely general

- The information in a file is defined by its creator.
- Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on. A file has a certain defined structure, which depends on its type
- A **text** file is a sequence of characters organized into lines (and possibly pages).
- A **source** file is a sequence of functions, each of which is further organized as declarations followed by executable statements.
- An **executable file** is a series of code sections that the loader can bring into memory and execute.

# File Attributes

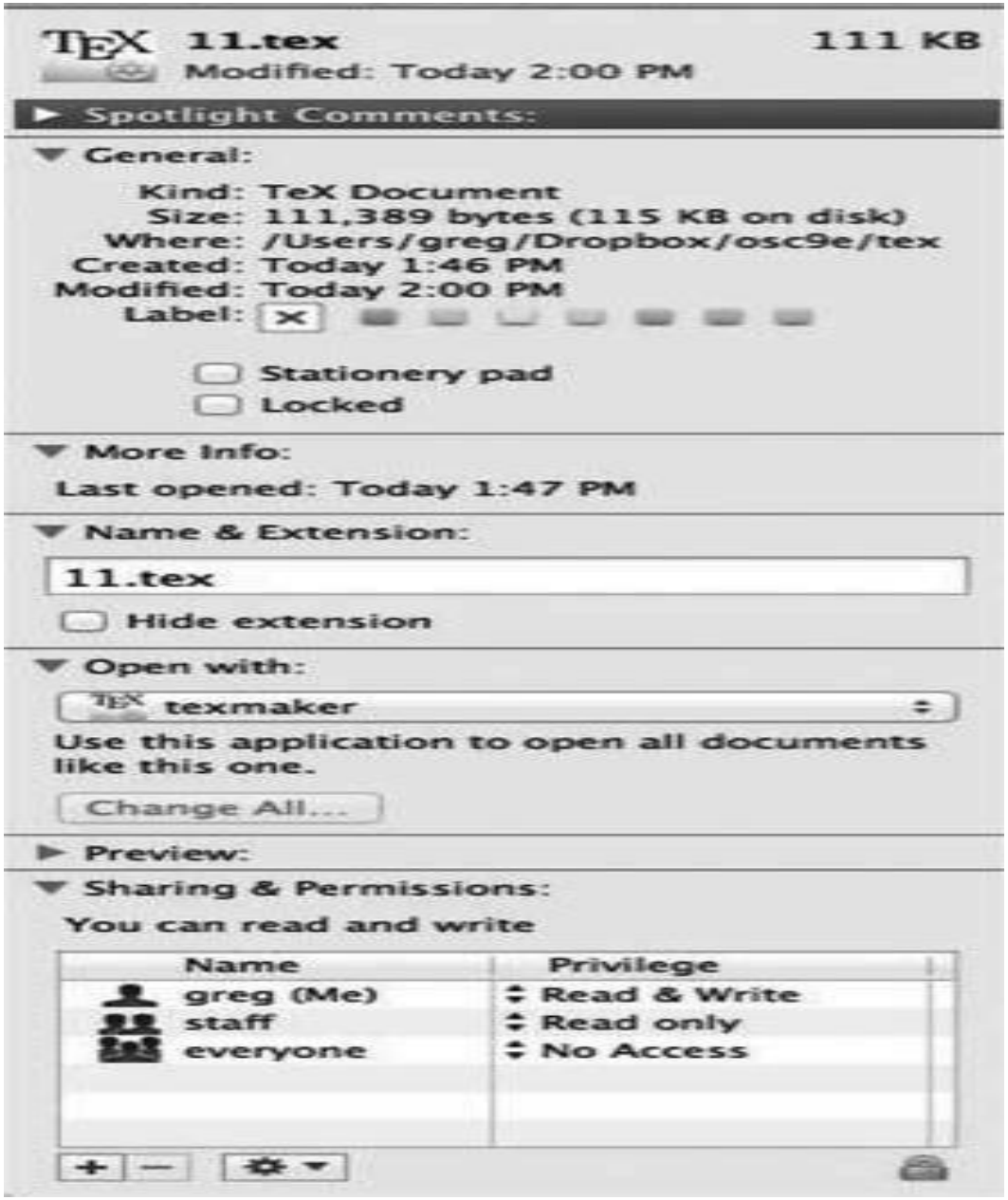
- **Name** – Symbolic file name is the only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system. It's a non-human-readable name for the file.

- **Type** – needed for systems that support different types
- **Location** – pointer to **device** & **file location** on that device

- **Size** – current file size (in bytes, words, or blocks) & possibly MAX allowed size
- **Protection** – Access-control information determines who can do reading, writing, executing & so on.

- **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 info Window on Mac OS X



# File Operations

- A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files.

- The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

# File Operations

- **Create:-** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

- **Write:-** To write a file, we make a system call specifying both the name of the file and the information to be written to the file.

- **Read:-** To read from a file, we use a systemcall that specifies the name of the file and where (in memory) the next block of the file should be put

- **Reposition within file:-**
- **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

# File Operations

- The operating system keeps a table, called the **open-file table, containing** information about all open files.
- When a file operation is requested, the file is specified via an index into this table, so no searching is required.
- When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table.

# Open Files

- Several pieces of data are needed to manage 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: Most file operations require the system to modify data within 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: Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

# 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 |

# Locking Open Files

- 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.
- File locks are useful for files that are shared by several processes—for example, a system log file that can be accessed and modified by a number of processes in the system.
- A **shared lock** is like 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.

# Access Methods

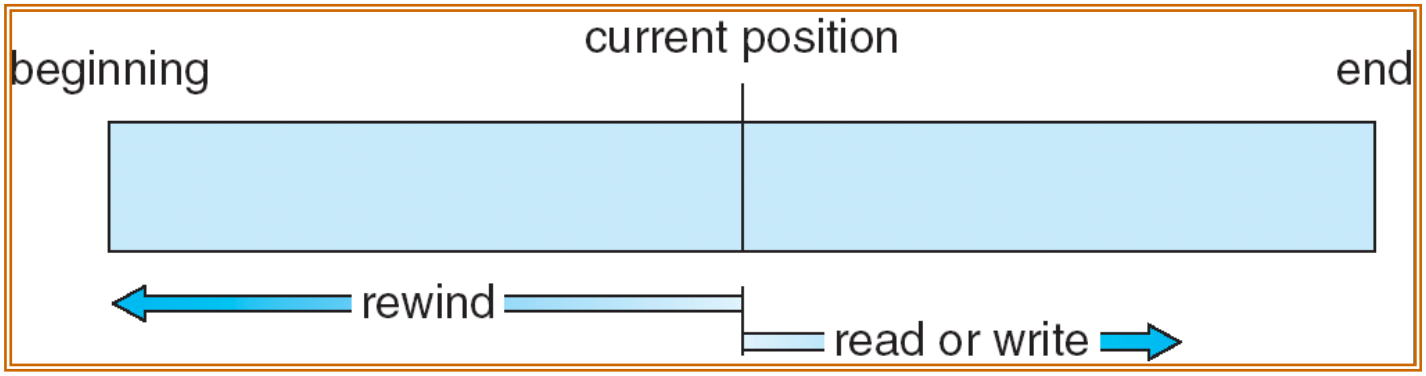
- **Sequential Access**

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

- **Direct Access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*
  *n* = relative block number

# Sequential-access File



## 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:--

  The simplest access method is sequential access. Information in the file is processed in order, one record after the other

Direct Access :--

Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.

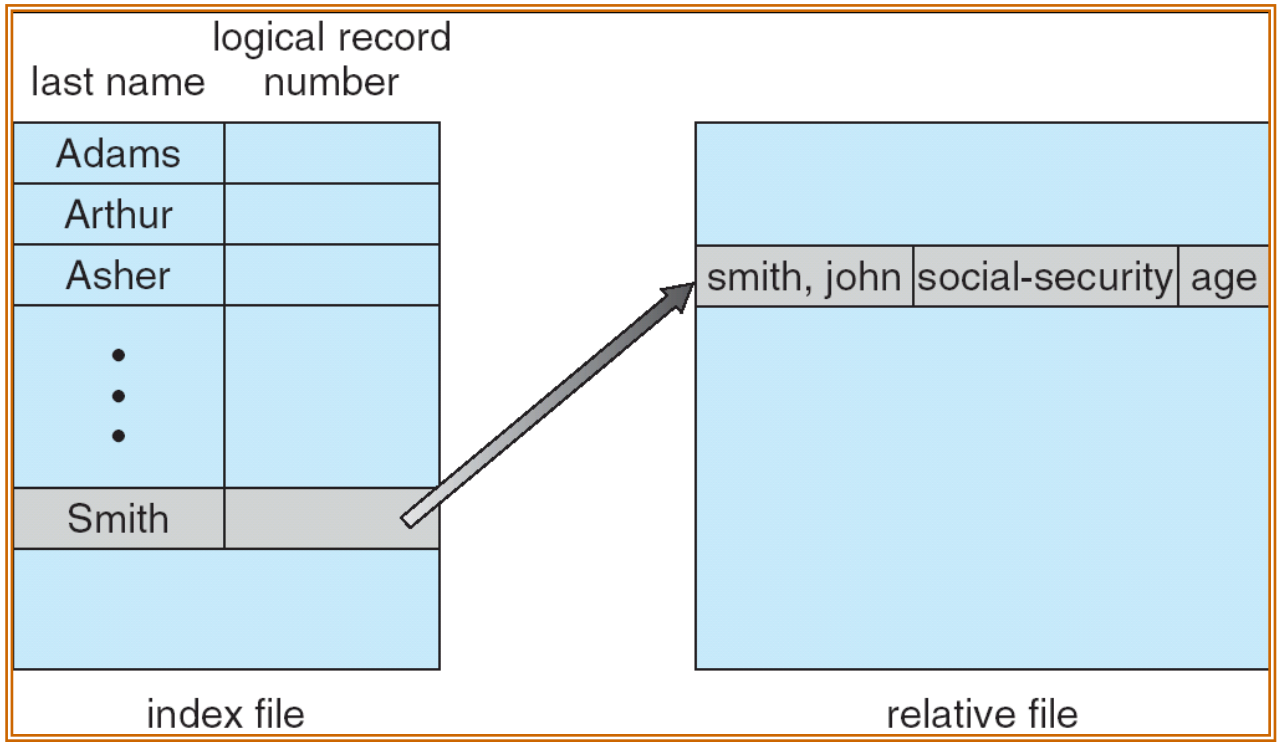
The direct-access method is based on a disk model of a file, since disks allow random access to any file block

- For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block14, then read block53, and then write block7. There are no restrictions on the order of reading or writing for adirect-access file.

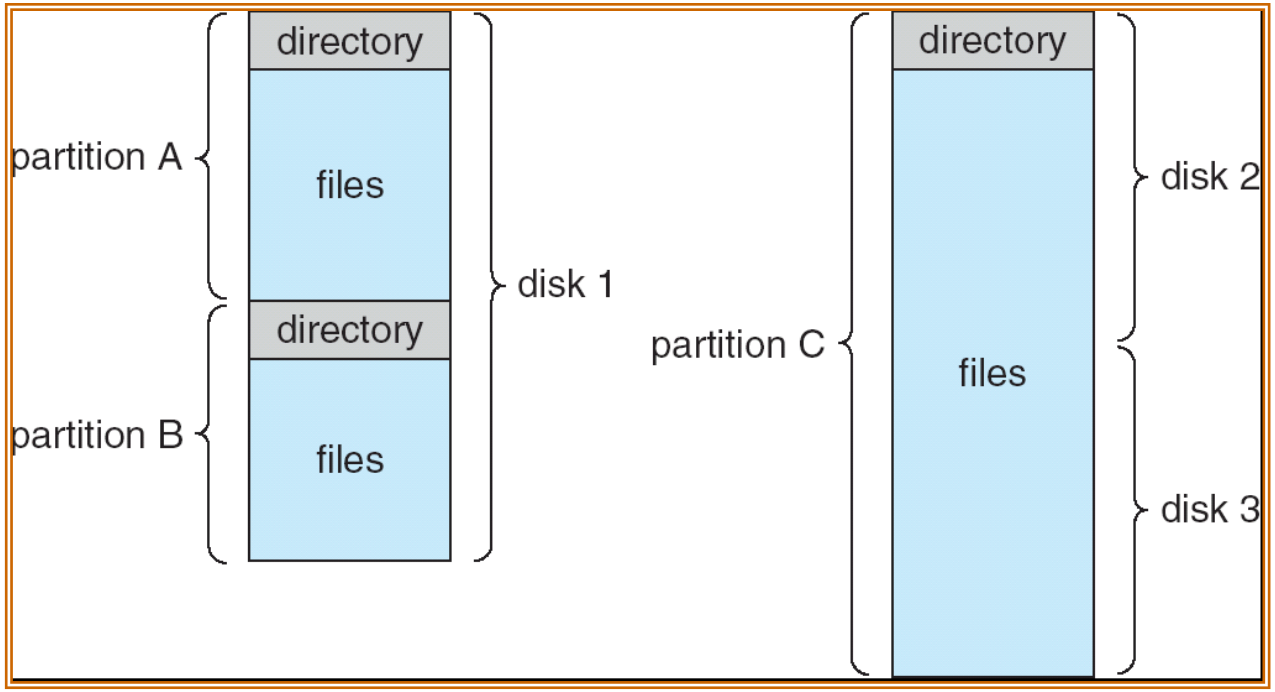
# Simulation of Sequential Access on a Direct-access File

| sequential access | implementation for direct access |
|---|---|
| *reset* | *cp* = 0; |
| *read next* | *read cp*;<br>*cp* = *cp* + 1; |
| *write next* | *write cp*;<br>*cp* = *cp* + 1; |

# Example of Index and Relative Files



# A Typical File-system Organization




- Could use entire disk for FS, but
  - system could have multiple FS types (e.g., swap)
- Disk divided into miniature disks called *partitions* or *slices*

# Storage Structure

- a general-purpose computer system has multiple storage devices, and those devices can be sliced up into volumes that hold file systems.
- Computer systems may have zero or more file systems, and the file systems maybe of varying types.
- Consider the types of file systems in the Solaris example

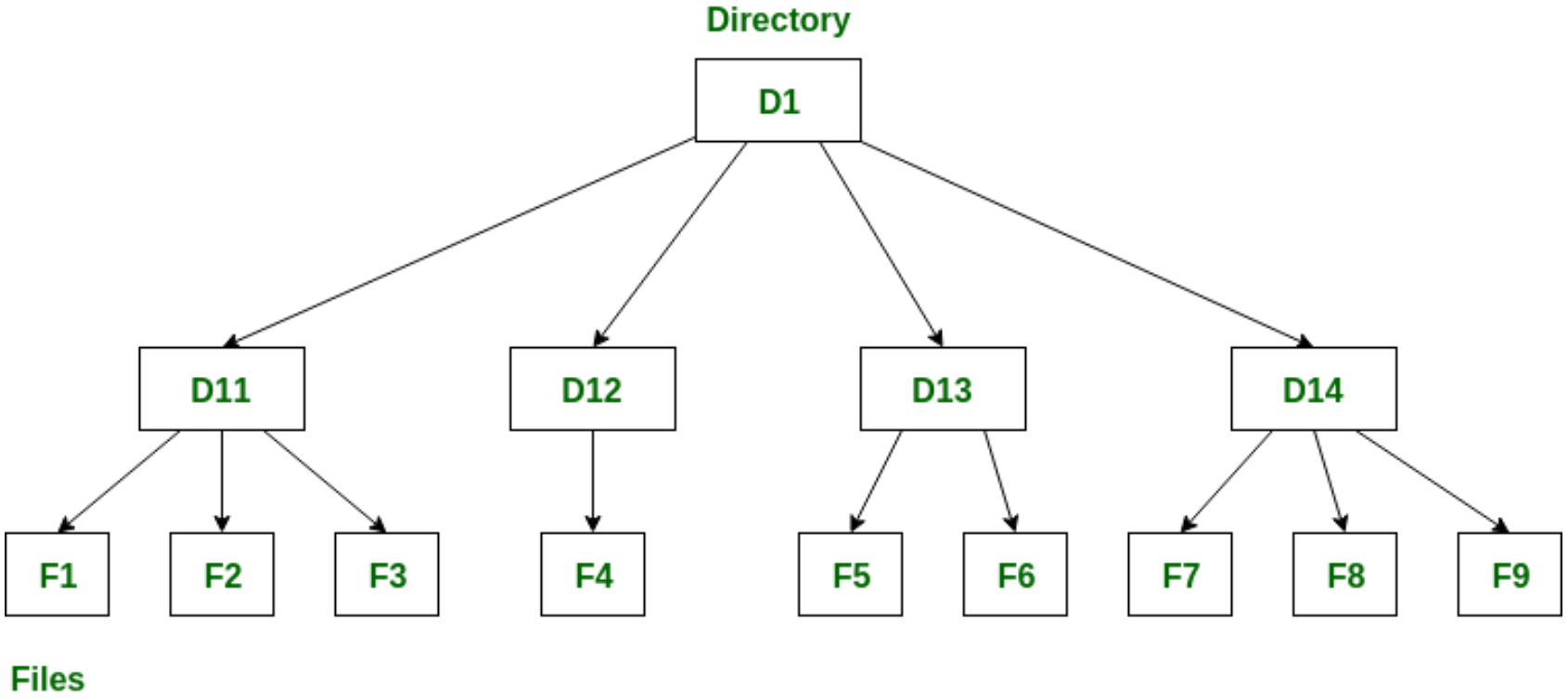
- **tmpfs—a "temporary" file system that is created in volatile main memory** and has its contents erased if the system reboots or crashes
- **objfs—a "virtual" file system (essentially an interface to the kernel that** looks like a file system) that gives debuggers access to kernel symbols
- **ctfs—a virtual file system that maintains "contract" information to manage** which processes start when the system boots and must continue to run during operation
- **lofs—a "loop back" file system that allows one file system to be accessed** in place of another one
- **procfs—a virtual file systemthat presents information on all processes as** a file system
- **ufs, zfs—general-purpose file systems**

# Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries.
- If we take such a view, we see that the directory itself can be organized in many ways.
- The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory

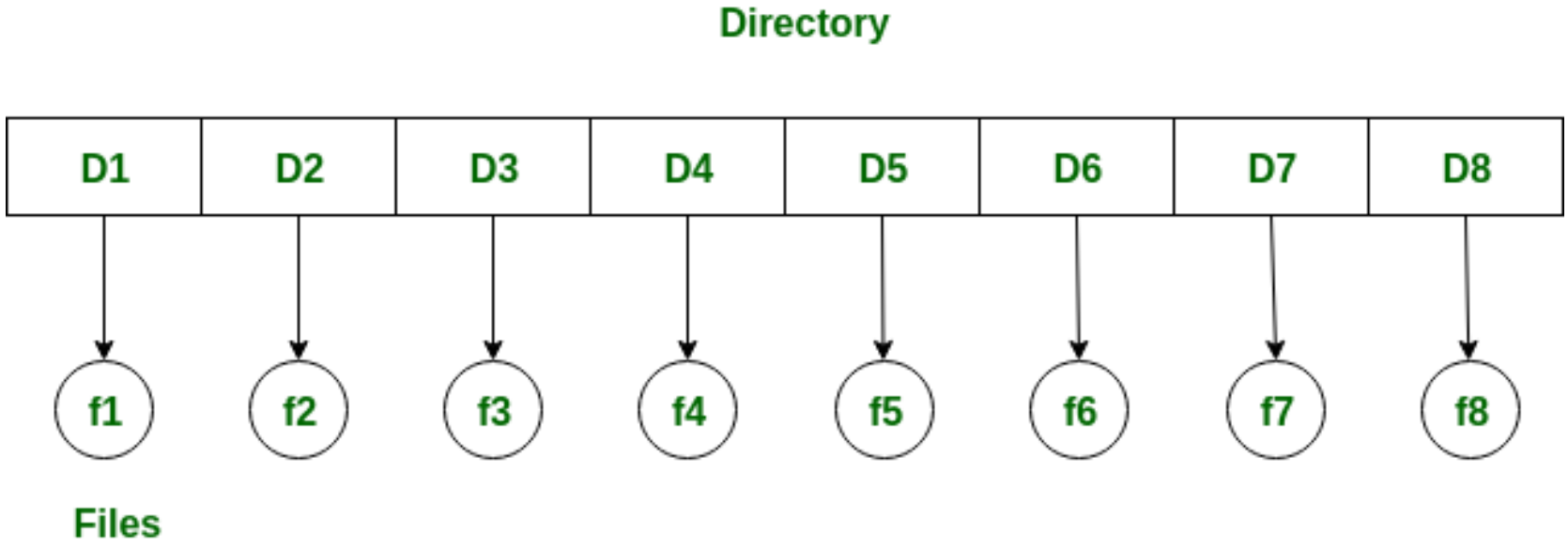
# Directory structure

- A **directory** is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.
- There are several logical structures of a directory



# Single-level directory

- Single level directory is simplest directory structure.
- In it all files are contained in same directory which make it easy to support and understand.
- A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user.
- Since all the files are in the same directory, they must have the unique name


Directory
D1
D2
D3
D4
D5
D6
D7
D8
f1
f2
f3
f4
f5
f6
f7
f8
Files

**Advantages:**

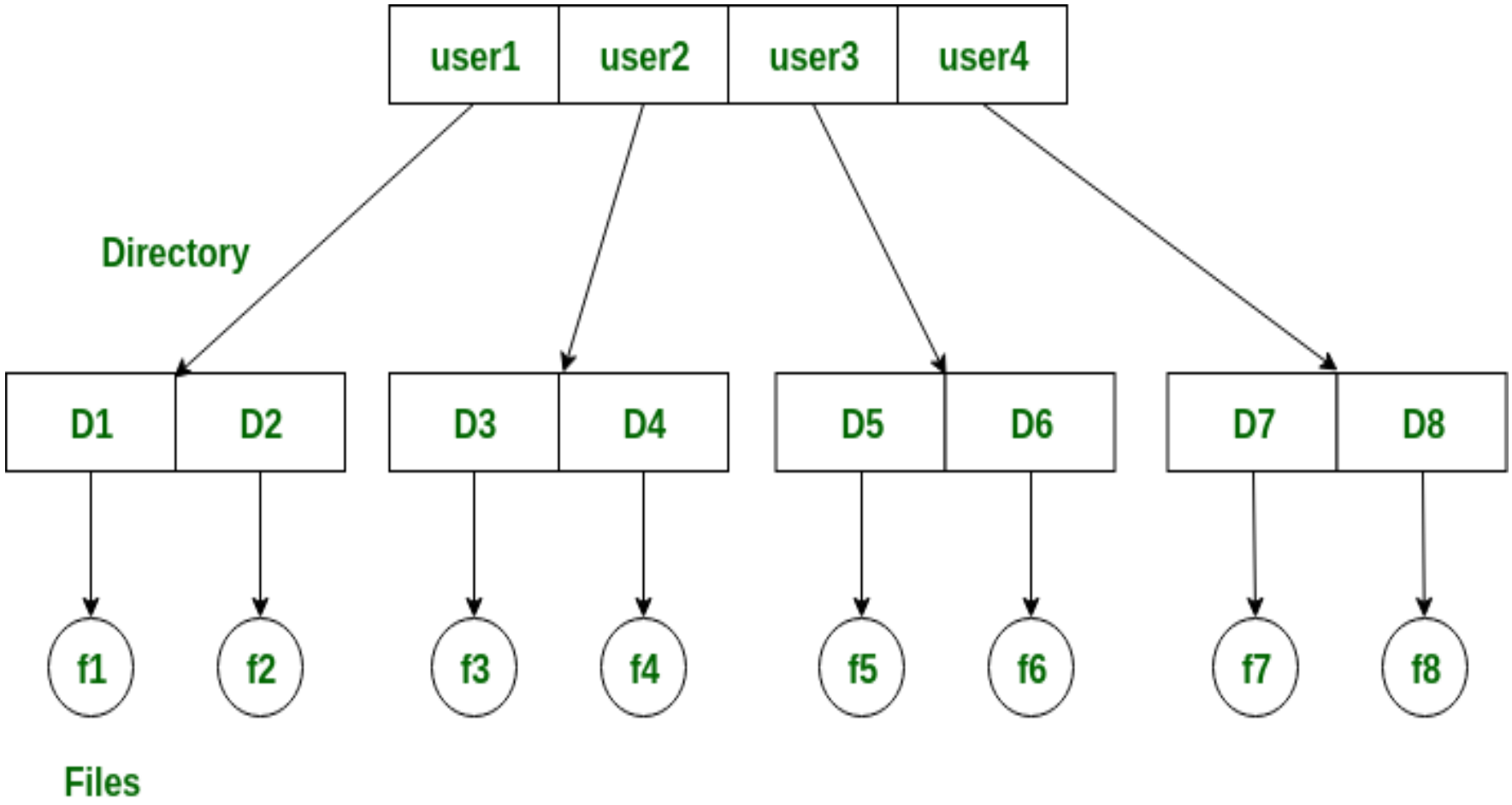
- Since it is a single directory, so its implementation is very easy.
- If files are smaller in size, searching will faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

**Disadvantages:**

- There may chance of name collision because two files can not have the same name.
- Searching will become time taking if directory will large.
- In this can not group the same type of files together.

# Two-level directory

- As we have seen, a single level directory often leads to confusion of files names among different users.
- the solution to this problem is to create a separate directory for each user.
- In the two-level directory structure, each user has there own *user files directory (UFD)*.
- The UFDs has similar structures, but each lists only the files of a single user.
- system's *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

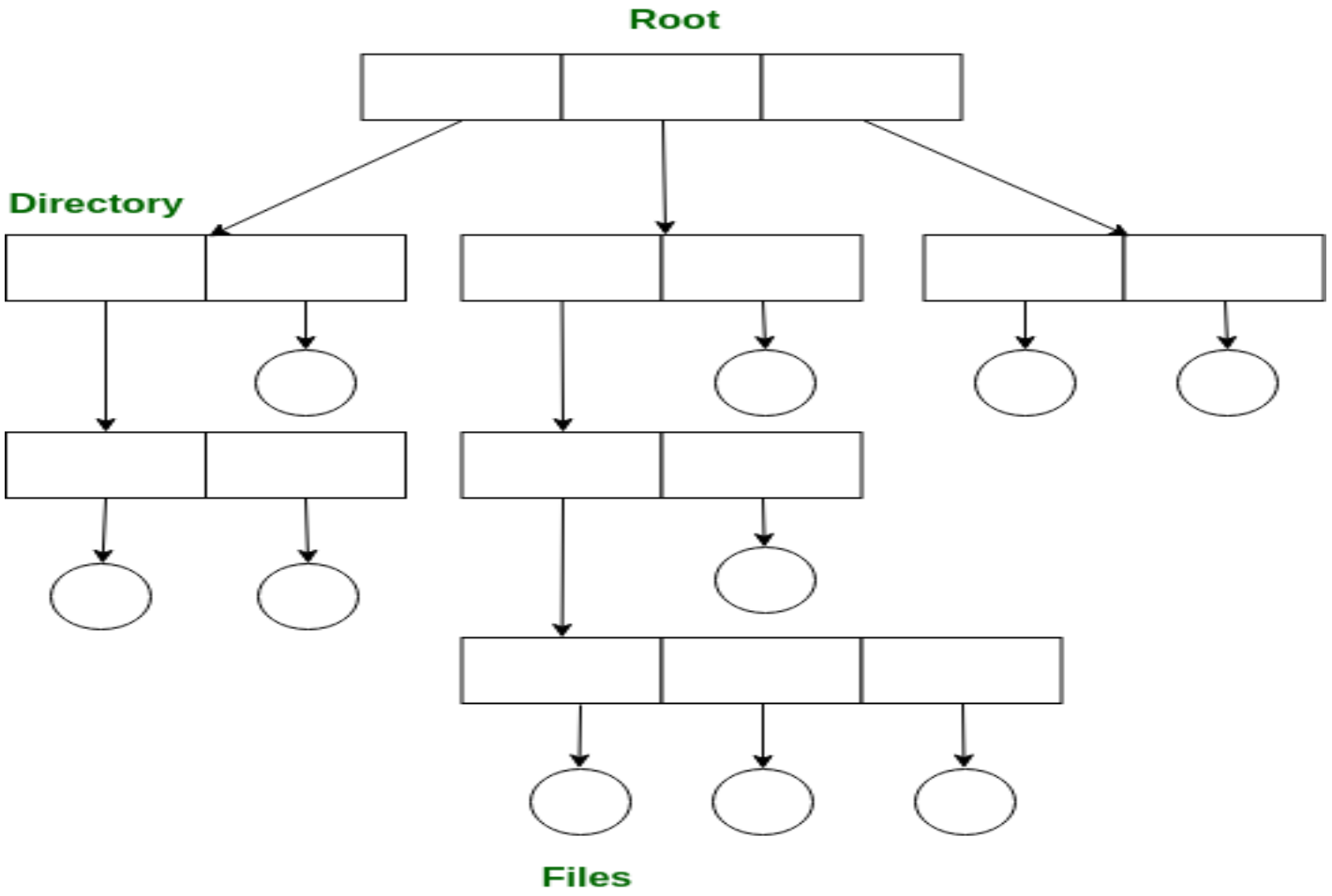

user1
user2
user3
user4
Directory
D1
D2
D3
D4
D5
D6
D7
D8
f1
f2
f3
f4
f5
f6
f7
f8
Files

- **Advantages:**
- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

# Tree-structured directory

- Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.

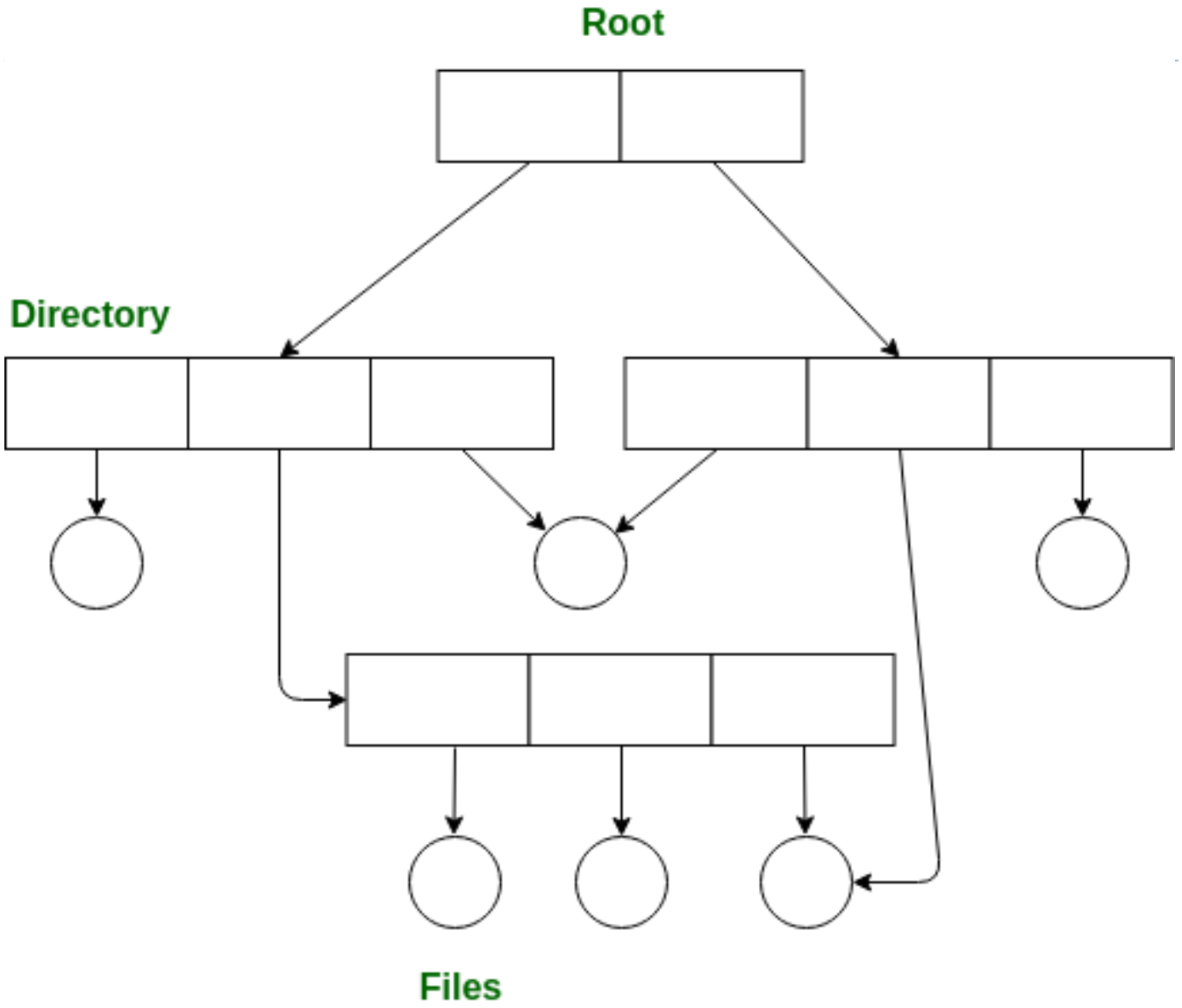
- This generalization allows the user to create there own subdirectories and to organize on their files accordingly.


Root
Directory
Files

- **Advantages:**
- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.
- **Disadvantages:**
- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.

# Acyclic graph directory

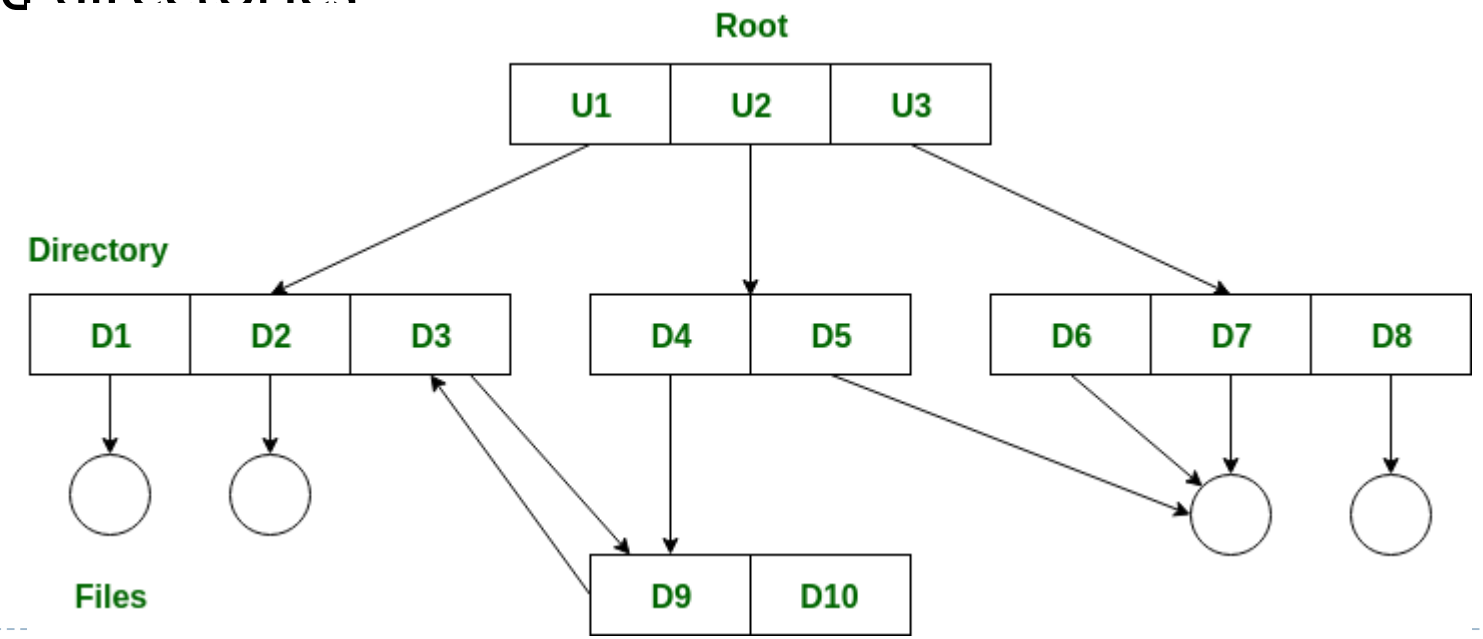
- An acyclic graph is a graph with no cycle and allows to share subdirectories and files.
- The same file or subdirectories may be in two different directories.
- It is a natural generalization of the tree-structured directory.
- It is used in the situation like when two programmers are working on a joint project and they need to access files


Root
Directory
Files

- **Advantages**
- We can share files.
- Searching is easy due to different-different paths.
- **Disadvantages:**
- We share the files via linking, in case of deleting it may create the problem,
- If the link is softlink then after deleting the file we left with a dangling pointer.
- **ages:**

# General graph directory structure

- In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.
- The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories



- **Advantages:**
- It allows cycles.
- It is more flexible than other directories structure.
- **Disadvantages:**
- It is more costly than others.
- It needs garbage collection.

# Operations Performed on Directory

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

# File System Mounting

- **Mounting** is a process by which the operating system makes files and directories on a storage device (such as hard drive, CD-ROM, or network share) available for users to access via the computer's file system

- Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system.

- More specifically, the directory structure may be built out of multiple volumes, which must be mounted to make them available within the file-system name space.

- The mount procedure is straightforward. The operating system is given the name of the device and the **mount point—the location within the file structure** where the file system is to be attached.

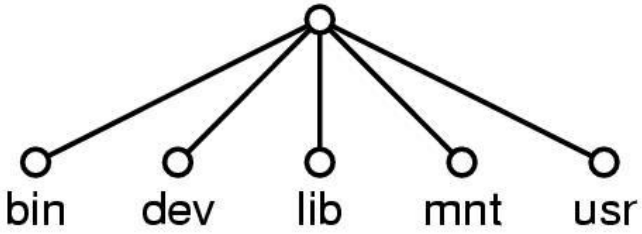
- An opposite process of mounting is called **unmounting**, in which the operating system cuts off all user access to files and directories on the mount point, writes the remaining queue of user data to the storage device, refreshes file system metadata, then relinquishes access to the device; making the storage device safe for removal.

- Typically, a mount point is an empty directory. For instance, on a UNIX system, a file system containing a user's home directories might be mounted as /home; then, to access the directory structure within that file system, we could precede the directory names with /home, as in /home/jane.

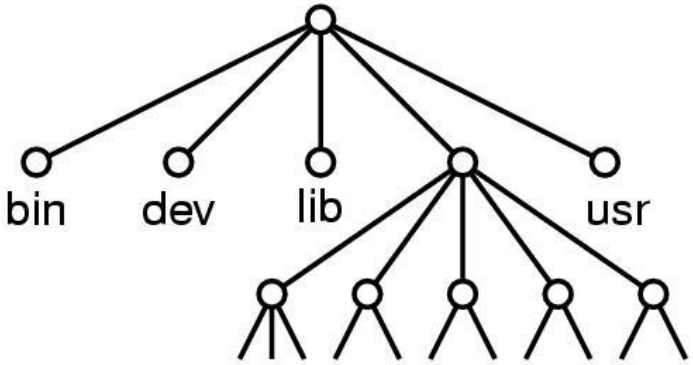
- Mounting that file system under /users would result in the path name /users/jane, which we could use to reach the same directory

- Mount allows two FSes to be merged into one
  - For example you insert your floppy into the root FS:

```
mount("/dev/fd0", "/mnt", 0)
```




(a)




(b)

# Mounting of File Systems

- There can be many file systems in an OS
- Each file system is constituted on a *logical disk*
  - i.e., on a partition of a disk
- Files can be accessed only when file system is *mounted*

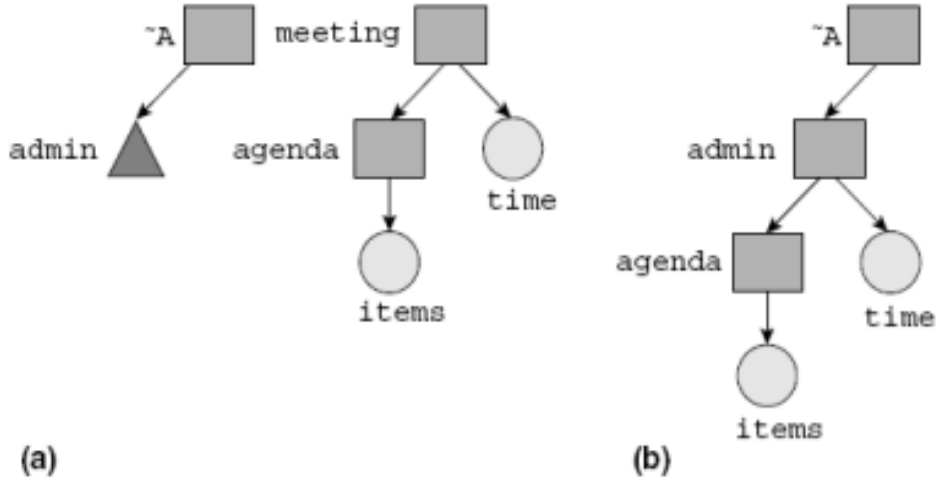



**Figure 13.11** Directory structures (a) before a mount command; (b) after a mount command.

# Protection

- When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).

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

- Protection can be provided in many ways. For a single-user laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet.

- In a larger multiuser system, however, other mechanisms are needed

# Categories of Users

- Individual user
  - Log in establishes a user-id
  - Might be just local on the computer or could be through interaction with a network service
- Groups to which the user belongs
  - For example, "nahum" is in "w4118"
  - Again could just be automatic or could involve talking to a service that might assign, say, a temporary cryptographic key

# File Sharing

- File sharing is the practice of sharing or offering access to digital information or resources, including documents, multimedia (audio/video), graphics, computer programs, images and e-books. It is the private or public distribution of data or resources in a network with different levels of sharing privileges.

- 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

- Operating systems also provide file-sharing methods, such as network file sharing (NFS). Most file-sharing tasks use two basic sets of network criteria, as follows

- P2P file sharing allows users to directly access, download and edit files. Some third-party software facilitates P2P sharing by collecting and segmenting large files into smaller pieces.
- File Hosting Services: This P2P file-sharing alternative provides a broad selection of popular online material. These services are quite often used with Internet collaboration methods, including email, blogs, forums, or other mediums, where direct download links from the file hosting services can be included. These service websites usually host files to enable users to download them

# File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

# File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
    - Using FTP under the covers
- **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
  - LDAP / Active Directory becoming industry standard -> **Secure Single Sign-on**
  - IP addresses can be **spoofed**
  - Protect remote access via firewalls
- Open file request to remote server first checked for client-to-server permissions, then user-id checked for access permissions, then file handle returned
  - Client process then uses file handle as it would for a local file

# File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
  - Data or **metadata** loss or corruption
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security
  - But **stateless** protocols can lack features, so NFS V4 and CIFS are both **state-ful**

# File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
  - Similar to Ch 7 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems)
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed
  - Easier to implement is **immutable shared files**
    - Once file is declared "shared", can't be renamed or modified

# Protection

- File owner/creator should be able to manage **controlled access**:
  - What can be done
  - By whom
  - But never forget physical security

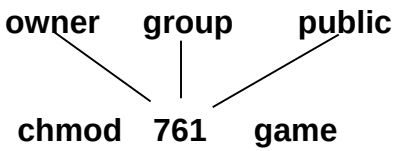
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**
  - Others can include renaming, copying, editing, etc
  - System calls then check for valid rights before allowing operations
    - Another reason for `open()`
  - Many solutions proposed and implemented

# Access Lists and Groups

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

| | | | 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.



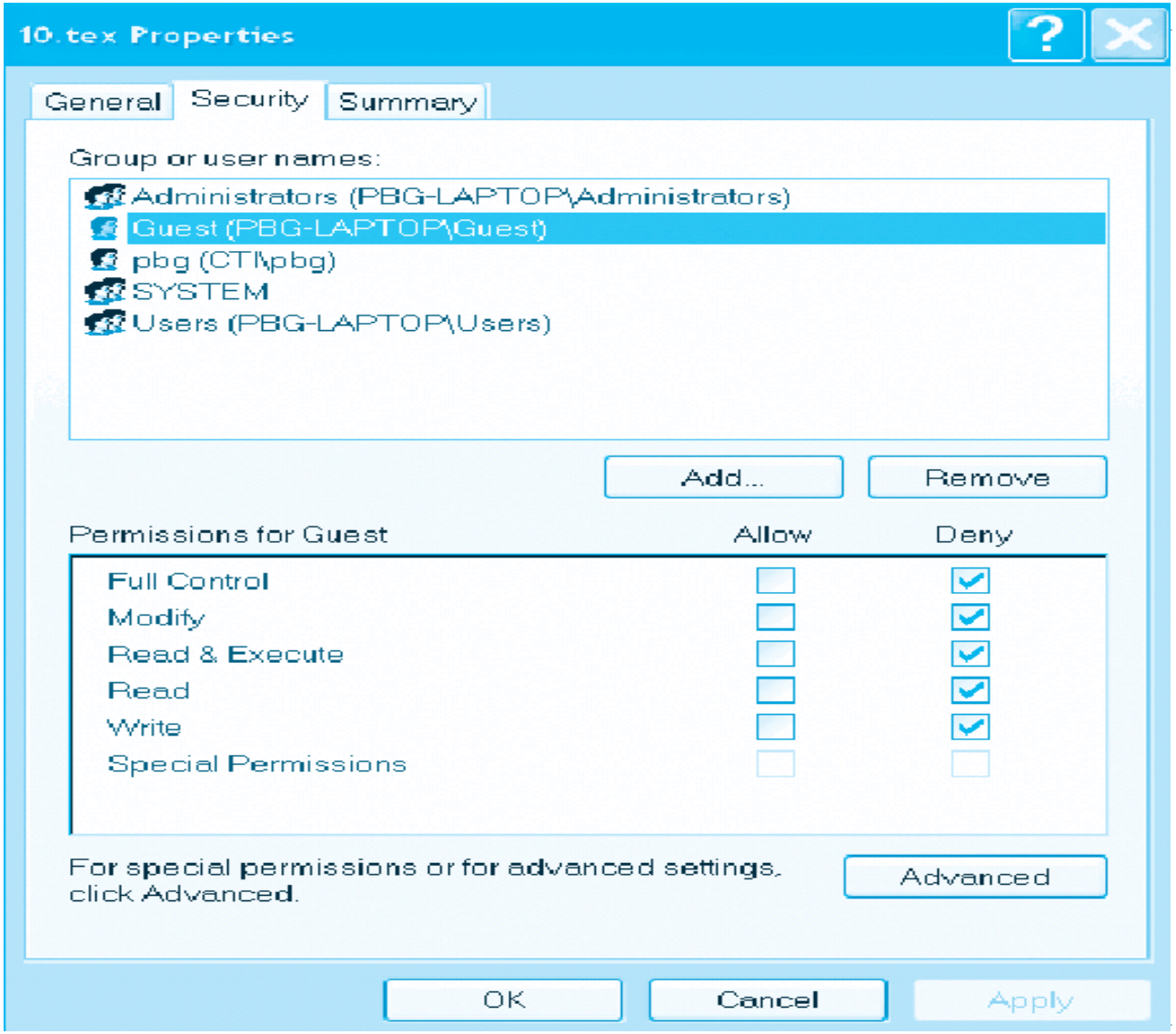

Attach a group to a file

chgrp G game

# Access Control

- More generally solved via **access control lists**
  - For a given entity, keep list of user-ids allowed to access and what access methods
  - Constructing such as list can be tedious and unrewarding
  - Data structure must be stored somewhere
    - Variable size

# Windows XP Access-Control List Management



# A Sample UNIX Directory Listing

| | | | | | |
|---|---|---|---|---|---|
| -rw-rw-r-- | 1 pbg | staff | 31200 | Sep 3 08:30 | intro.ps |
| drwx------ | 5 pbg | staff | 512 | Jul 8 09.33 | private/ |
| drwxrwxr-x | 2 pbg | staff | 512 | Jul 8 09:35 | doc/ |
| drwxrwx--- | 2 pbg | student | 512 | Aug 3 14:13 | student-proj/ |
| -rw-r--r-- | 1 pbg | staff | 9423 | Feb 24 2003 | program.c |
| -rwxr-xr-x | 1 pbg | staff | 20471 | Feb 24 2003 | program |
| drwx--x--x | 4 pbg | faculty | 512 | Jul 31 10:31 | lib/ |
| drwx------ | 3 pbg | staff | 1024 | Aug 29 06:52 | mail/ |
| drwxrwxrwx | 3 pbg | staff | 512 | Jul 8 09:35 | test/ |

# File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through the **protection** scheme
- On distributed systems, files may be shared across a network
- Use a distributed file system such as Network File System (NFS)
- Use distributed naming systems such as NIS, LDAP, Active Directory for network-wide user information

# Remote File System Mounting

- Same idea, but file system is actually on some other machine
- Implementation uses remote procedure call
  - Package up the user's file system operation
  - Send it to the remote machine where it gets executed like a local request
  - Remote sends back the answer
- Very common in modern systems
  - E.g., the CLIC lab, compute nodes, etc.

# File Sharing – Client-Server Model

- **Client-server** model allows clients to mount remote file systems from servers
- Server can serve multiple clients
- **NFS** is a standard UNIX client-server file sharing protocol
  - Multiple versions; V4 is standard now
  - We use it on CLIC, compute nodes
- **CIFS** is standard Windows protocol
- Standard operating system file calls are translated into remote procedure calls
  - Package up the user's file system operation
  - Send it to the remote machine where it gets executed like a local request
  - Remote sends back the answer

End