Chapter 9: File System Interface

File System Interface

File Concept

- Computers store information on different [physical] media
  - Flash Drives, Magnetic disk, Optical Disks, Magnetic Tapes
- OS provides a uniform view of stored info [for convenience] – How?
  - DOS directory, Windows directory, UNIX directory
- The OS takes an abstraction of the physical storage media & defines a logical storage unit: File
  - Files are held in persistent storage on the physical device
    - Sequence of bits, bytes, lines or records
  - The OS maps the File to the Physical device
    - A logical representation of how files are stored on the physical device
      - Mapping varies from one OS to another

File is an abstraction

- To define the file, we need to examine/agree on some basic characteristics;
- Attributes, Structure, Operations on File, Types, etc…

File Concept

Common File Attributes

File Attributes are dependent on the OS

- **Name** – Symbolic file name is only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
  - Used to locates the trailing file attributes
- **Type** – needed for systems that support different types of files
- **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
  - **Structure specifies entries for the name and identifier**

File Concept

File **Operations**

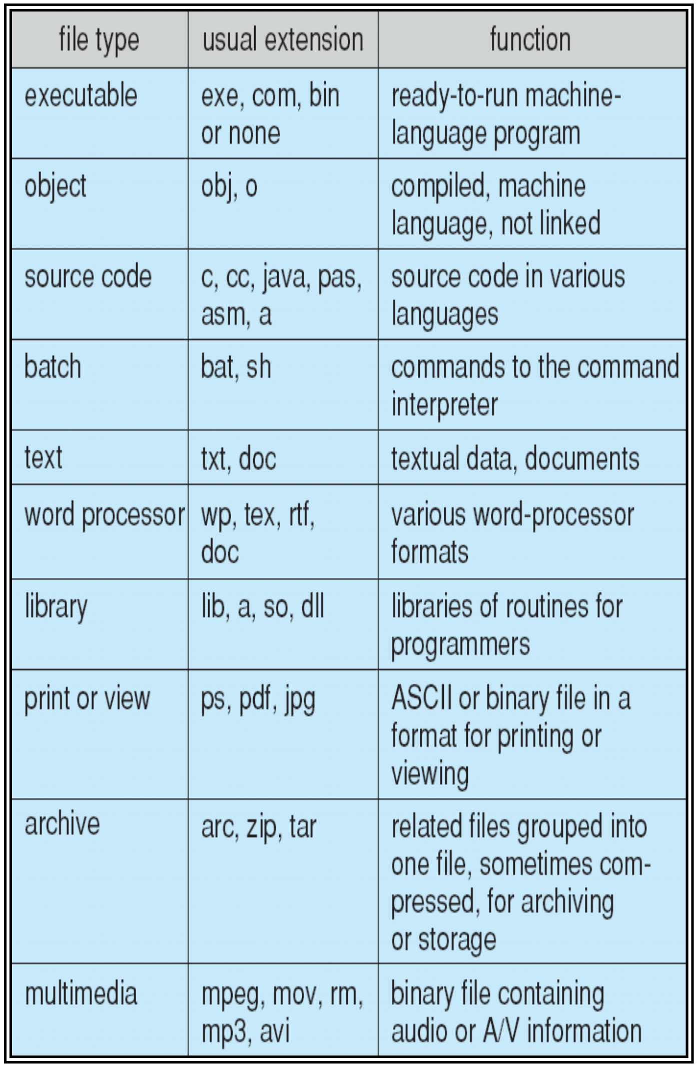
- OS provides system calls to perform Basic Operations:
  - Create
  - Write
  - Read
  - Reposition within file (seek)
    - Reposition to current file position pointer to a given value
  - Delete
  - Truncate
    - Attributes are unchanged

- *Open(F_i)* – search the directory structure on disk for entry F_i, and move the content of entry to memory
  - Updates a small table of open files; *open-file-table*
    - Minimizes overhead of searching entire file system
- *Close (F_i)* – move the content of entry F_i in memory to directory structure on disk
  - Remove entry from *open-file-table*

File Concept
File Type & File Structure

- ***File Types***
  - OS may need to recognize file types
  - To operate on files correctly
    - Try printing a binary file type
- File Type Specs:
  - Include file type as an extension of file name
    - *resume.doc, myprog.c*
  - *The OS uses the extensions to indicate to user the type of file and valid operations on the file*
- *File Structure*
  - *OS may require files to conform to structure understood by the OS*
    - *File extensions provide an indication of the internal structure of the file*

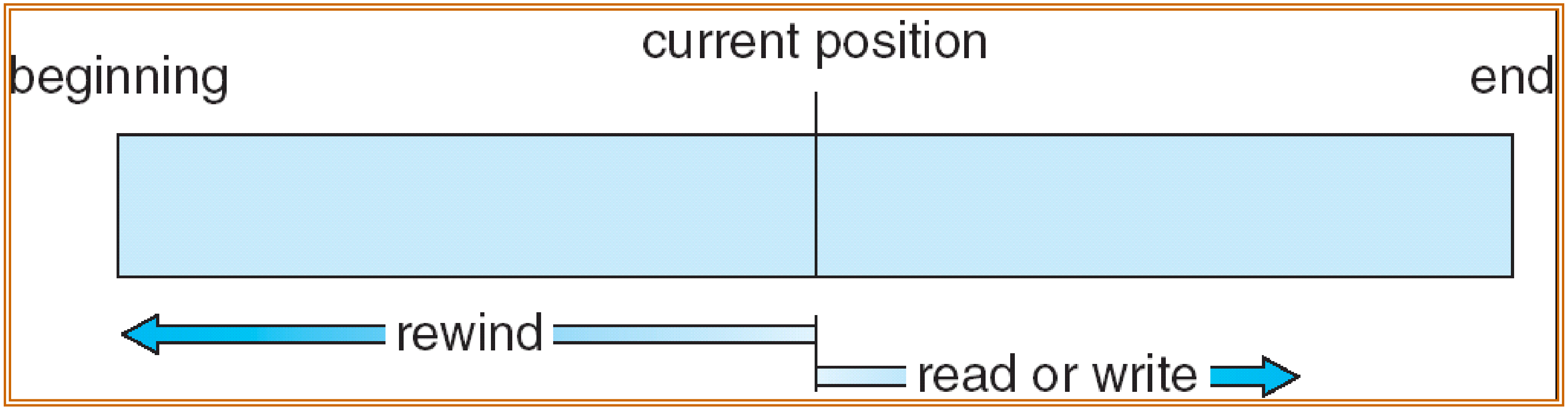
Common File Types



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
Sequential Access

- **Information stored in the file is processed serially: one record at a time:**
  - Read Operation (read next) – reads next portion of file and advances the file pointer
  - Write Operation (write next) – appends to end of file (eof) and advances file pointer to the new eof
  - File Pointer tracks I/O location
    - Allows File to be reset at the beginning



Access Methods
Direct Access

- **Information stored in file is processed in any random order**
  - **File is viewed as a numbered sequence of blocks (1, 2, …10,11,12..)**
    - Read Operation (read 12) – reads block 12
    - Write Operation (write 10) – write to block 10
  - Designed for access to large amounts of data items
    - Database query

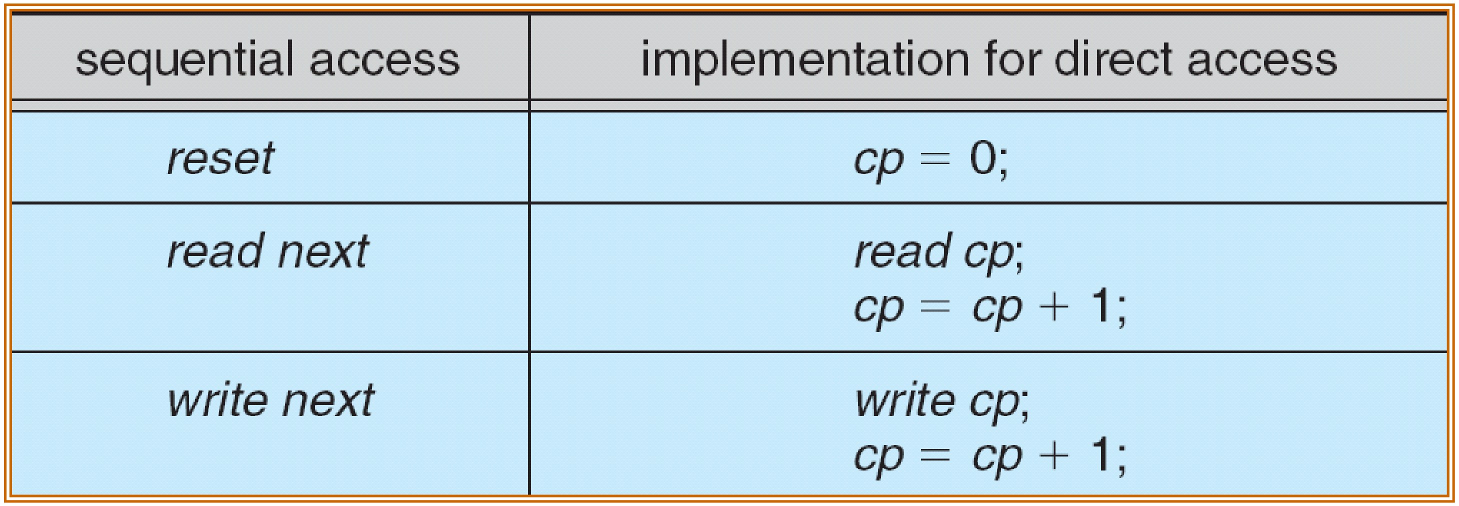
Direct Access:

```
            read n
            write n
Alternatively:
            position to n
                  read next
                  write next
```

n = relative block number to start of file

Access Methods
Simulation of Sequential Access on a Direct Access File

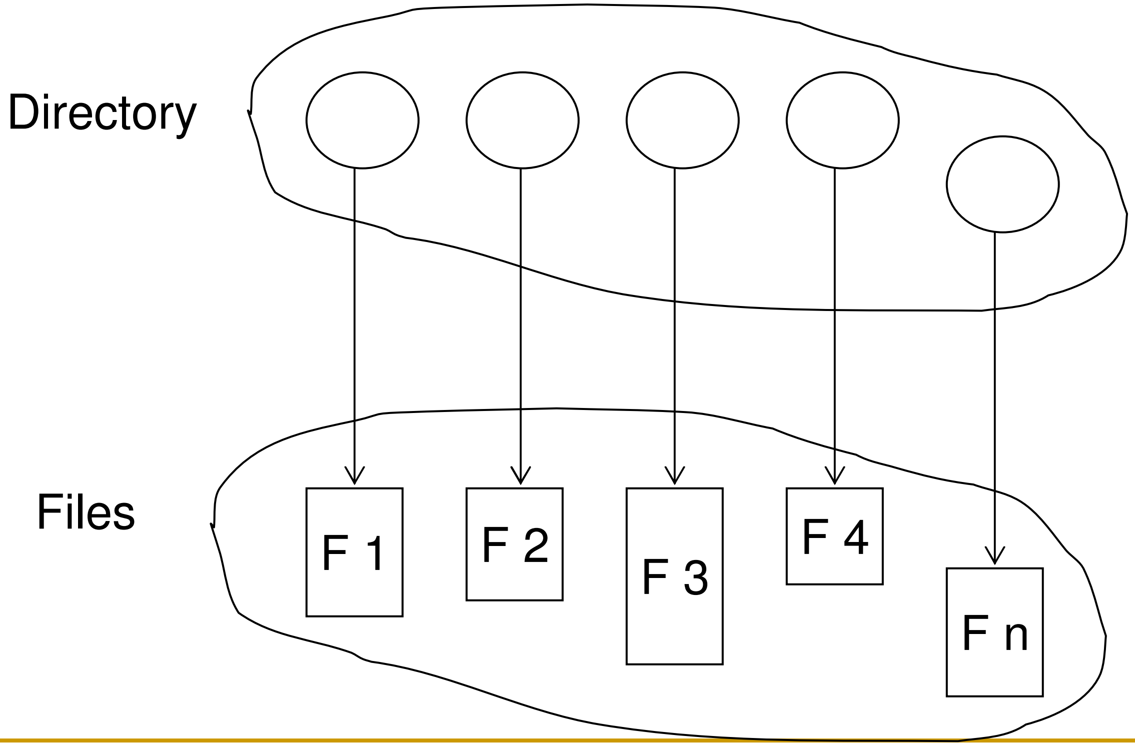
- Define Variable *cp* that holds current position (cp) of file pointer
  - Increment cp by 1 after each read/write operation



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

File System Interface
Directory Structure

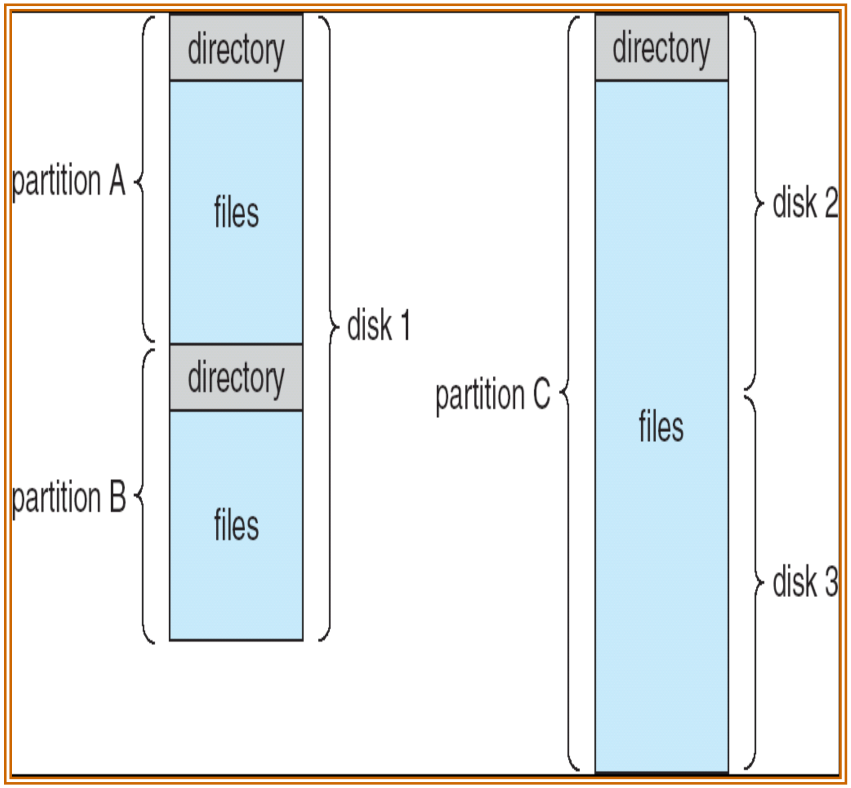
- File Systems may contain info up to a million terabytes and more on a disk
- Directory Structure is scheme to manage the file systems
  - A collection of nodes containing information about all files



File-System: Directory

Storage Structure

- File-systems are stored on a disk (storage)
- Partitions:
  - Multiple file-systems on different parts of a disk
- Volume:
  - Combinations of partitions to form a larger structure



DirectoryStructure

Operations

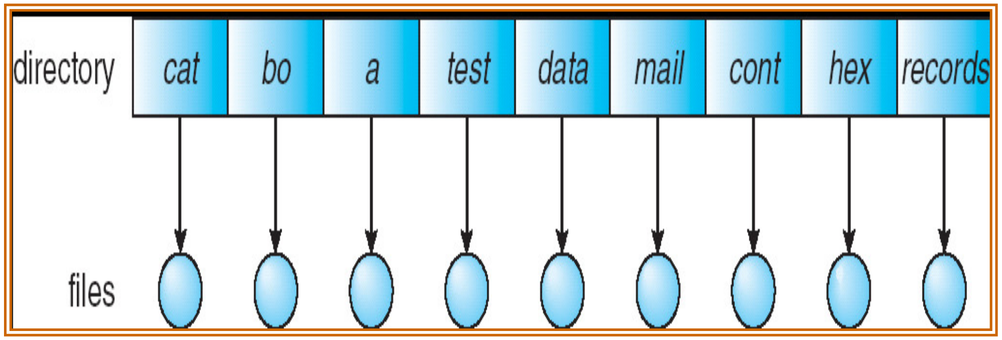
Let’s examine typical operations performed on directories (“Use Case Scenarios”)

- Search for a file
  - Search for files using pattern matching of names
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Structure

Single-Level Directory

- All files are in the same directory
- Limitations:
  - File names must be unique
  - Difficult to remember file names in a large file system
    - Grouping problem
  - Different users may encounter Name Collision Problem

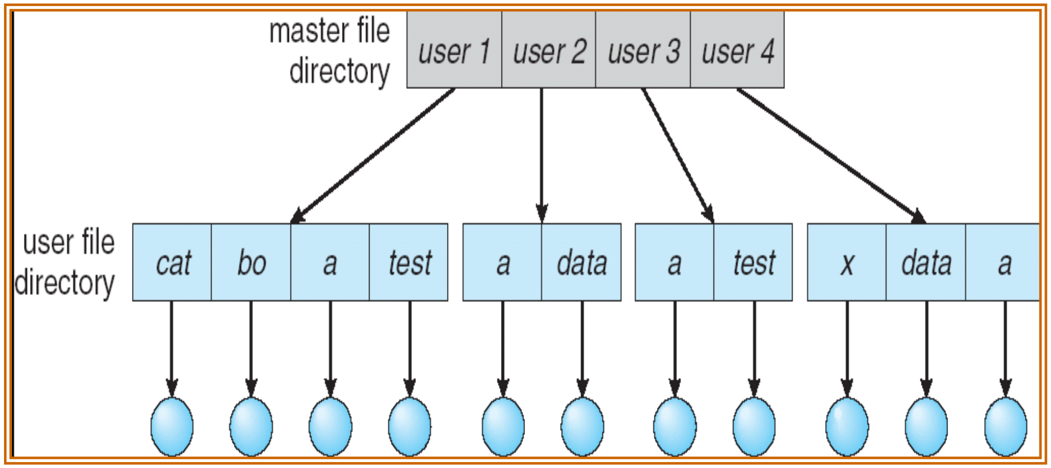


Create different dir for each user

Directory Structure

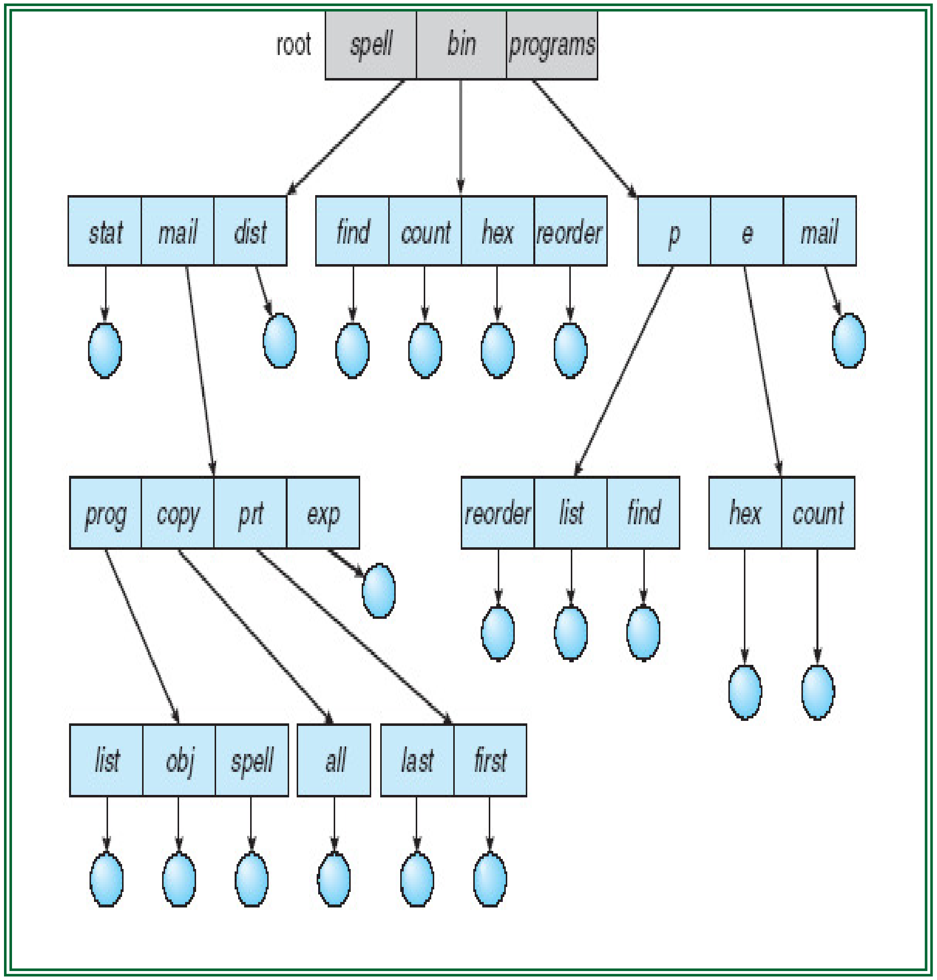
Two-Level Directory

- Each user has a User File Dir (UFD)
- UFDs have similar structure
- MFD contains the list of UFDs
- UFDs spawned at start-up via MFD
- Searches are local to UFDs
  - Resolves name-collision problem
- Limitations:
  - Users cannot cooperate across directories
  - Access permission across Dir requires designation of full path names of files
  - Grouping-problem not resolved



Directory Structure
Tree-structure Directories

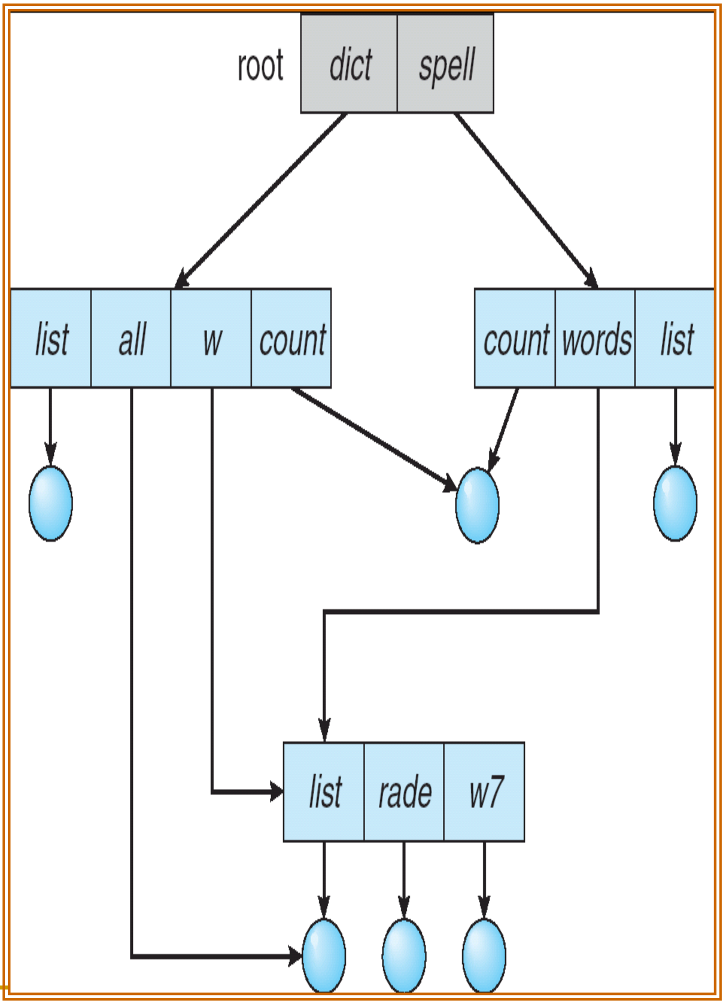
- Generalization of Two-level Dir
- Resolves grouping
  - Users can create subdirectories….
    - mkdir < dir-name>
    - rm <file name>
    - Efficient searching
- Tree has root dir
  - Unique path name for files



Directory Structure

Acyclic-Graph Directories

- Share Files and Share subdirectories
- Directory exist in two or more places at the same time
  - new directory (link) is created in each user dir.
    - Link then points to the sub dir or file
- More flexible than simple tree-structure



Directory Structure

Acyclic-Graph Directories

- 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

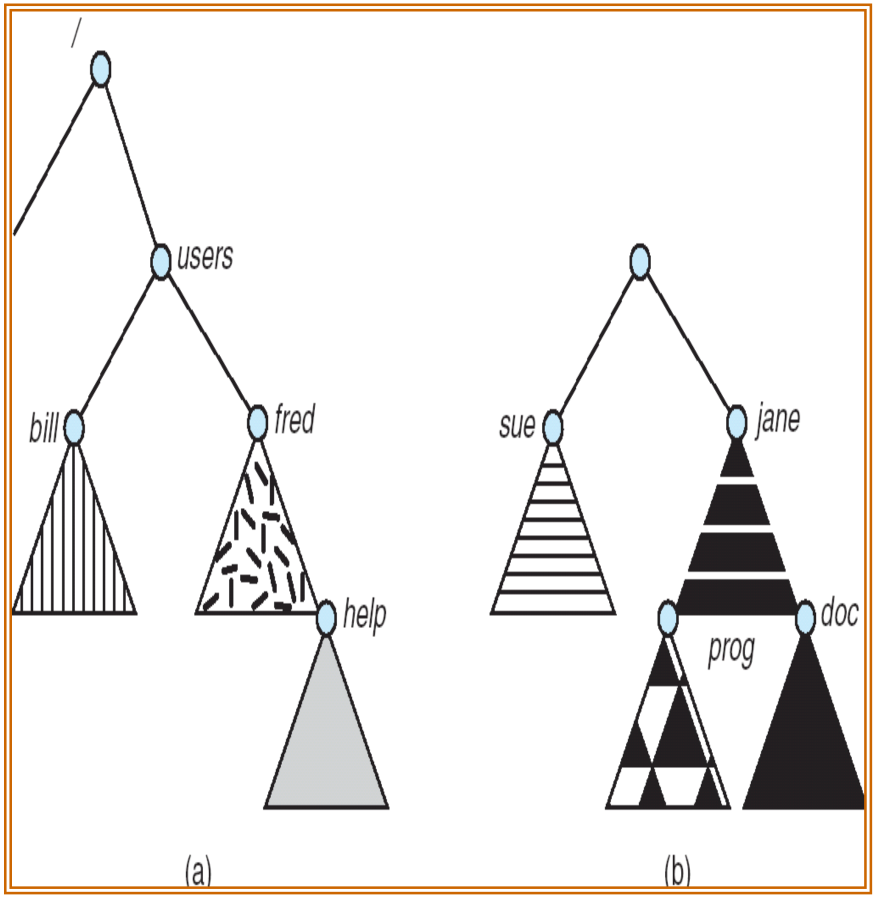
File System Mounting

- A file system must be **mounted** before it can be accessed
  - Directory structure must be mounted to make them available within the File system
- To mount a File system, OS needs:
  - Device name
  - Location within existing file structure where file system is to be mounted (mount point):
    - E.g., /home   /usr   /bin
    - Given a file system   /jane the dir structure is
      - /home/jane   if mounted under /home
      - /usr/jane   if mounted under /usr
      - /bin/jane   if mounted under /bin

File System Mounting

a) Existing File system structure
- With Mount Points:
- users, bill, fred and help

b) Unmounted Partition
- resides on a media device (disk)

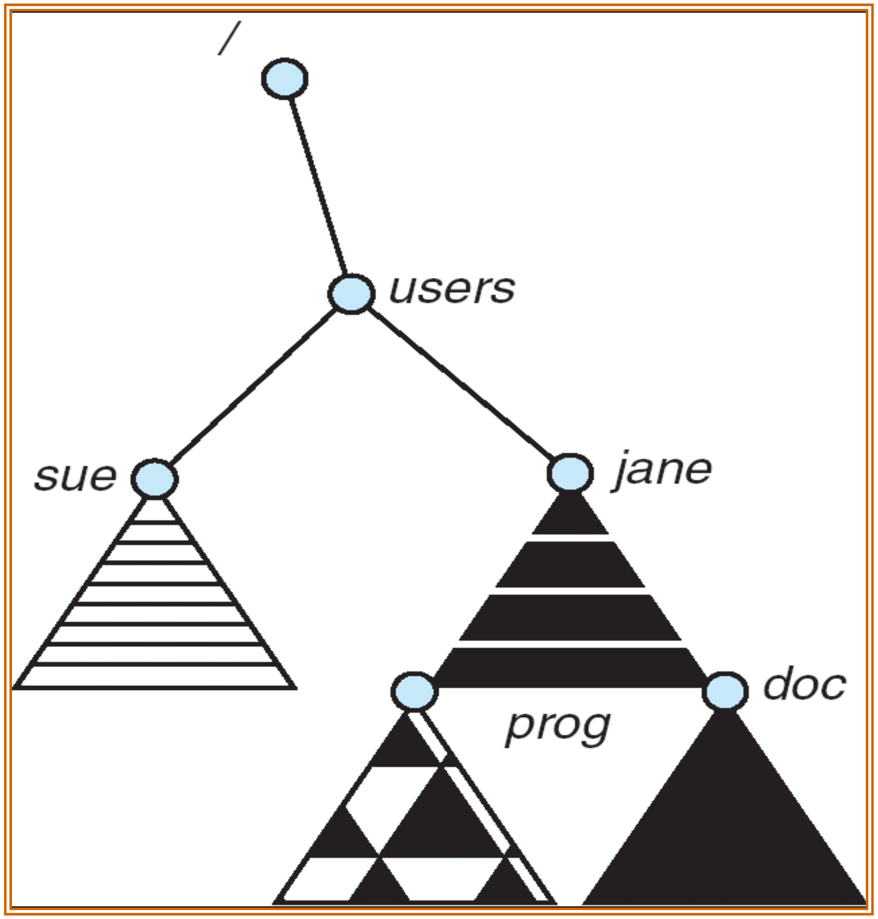


Currently, only files in existing file system structure (a) can be accessed

Mount Point

Mount Location within existing file structure is:

/users



Now files in directory: sue, jane, prog and doc can be accessed

File Sharing

- In Multi-User Systems
  - Files may be shared by users
  - OS mediates file sharing

- Sharing may be done through a **protection** scheme
  - OS maintains additional file/directory attributes:
    - Owner ID
    - Group ID
- On distributed systems, files may be shared across a network
  - ftp: anonymous and authenticated file exchange
  - www: anonymous file exchange for the most part
- Network File System (NFS) is a common distributed file-sharing method

File Sharing
Additional File/Directory Attributes

- **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**

- **Client-server** model allows clients to mount remote file systems from servers
  - Server is machine containing the files:
    - can serve multiple clients
  - Client is machine seeking access to the files
    - User-on-client identification is insecure or complicated
      - IP address (can be spoofed)
        - Encrypted keys for added security

  - **NFS** is standard UNIX client-server file sharing protocol:
    - User Id maintained on Client and Server must match

Remote File Systems

Sharing cont

- Distributed Information Systems:
  - **Provide unified access to remote computing**
    - **DNS: Host name to IP address**
    - **NIS: “yellow pages concept”- centralized repository for user attributes: E.g., User name + printer access on network +**

Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
  - Client and Server maintain knowledge of current activities and open files (states)
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

Protection

Mulituser File Systems

- We need Controlled Access to the File System
  - File owner/creator should be able to control:
    - what can be done
    - by whom
- Types of common (primitive) access control

Read	- Read from file
Write	- Write or re-write the file
Execute	- Load file into memory and execute
Append	- Write new information at end of file
Delete	- Delete file and free its pace
List	- List names and file attributes

How does the OS enforce control for: copying and renaming a file?

Protection

- Common Solution to Protecting File/Directory Systems:
  - Associate each file/directory with an Access-Control List (ACL)
    - UID + Types/Mode of allowed access
- OS uses ACL to process user requests
- Challenge:
  - Different users may have different types of access
    - Difficult to construct a comprehensive list of access without prior knowledge of user needs
      - Adopting Variable directory entry poses complex challenges
  - Adopt a condensed ACL
    - Owner: File owner
    - Group: Set of user that share the file and require similar access as owner
    - Universe – Other users

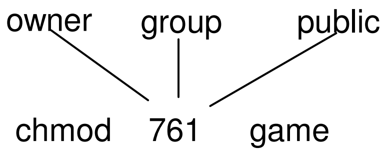
Access Lists and Groups

UNIX File-system

- 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

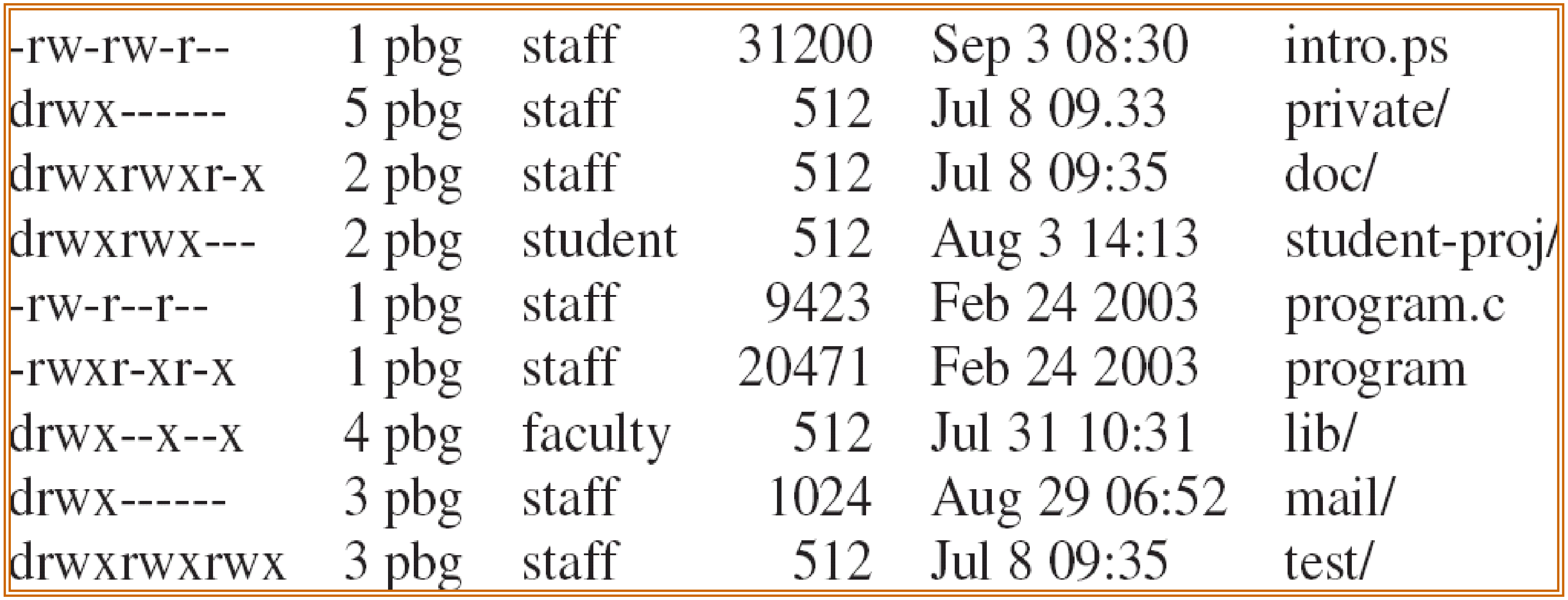
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



chmod 761 game

Attach a group (G) to a file: chgrp G game

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-system Interface

Let’s summarize

- The major task for OS is to map logical file concept onto physical storage device (magnetic tape, disks)
- Each device maintains a directory
  - List of locations on device
- Single-level directory causes naming collision problems for multiple users
- Two-level directory structure resolve naming collision, but introduces exclusivity
- Tree-structured directory allows users to create subdirectories
- Acyclic allows users to share files and directory, but searching and deletion gets complicated
- File protection can be provided by password, ACL…
  - Controlled access for read, write, execute (common attributes)
  - Combination of common and optional parameters (recent approach)