

# 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

- Logical units of information on secondary storage
- Named collection of related info on secondary storage
- Abstracts out the secondary storage details by presenting a common logical storage view

# 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

- Create
- Write
- Read
- Reposition within file
- Delete
- Truncate
- $Open(F_i)$  – 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: records the current position in the file, for the next read or write access.

- File open count:

- How many times has the current file been opened ( simultaneously by different processes ) and not yet closed?

- When this counter reaches zero the file can be removed from the table.

- Disk location of the file: cache of data access information


- Access rights: per-process access mode information

# File Operations

- Some systems provide support for file locking.
  - A shared lock is for reading only.
  - A exclusive lock is for writing as well as reading.
  - An advisory lock is informational only, and not enforced.
  - A mandatory lock is enforced.
- UNIX used advisory locks, and Windows uses mandatory locks.



# 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 com- pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

# File Structure

- Some files contain an internal structure, which may or may not be known to the OS.
- For the OS to support particular file formats increases the size and complexity of the OS.
- UNIX treats all files as sequences of bytes, with no further consideration of the internal structure.
- Macintosh files have two forks - a resource fork, and a data fork.
- The resource fork contains information relating to the UI, such as icons and button images, and can be modified independently of the data fork, which contains the code or data as appropriate.

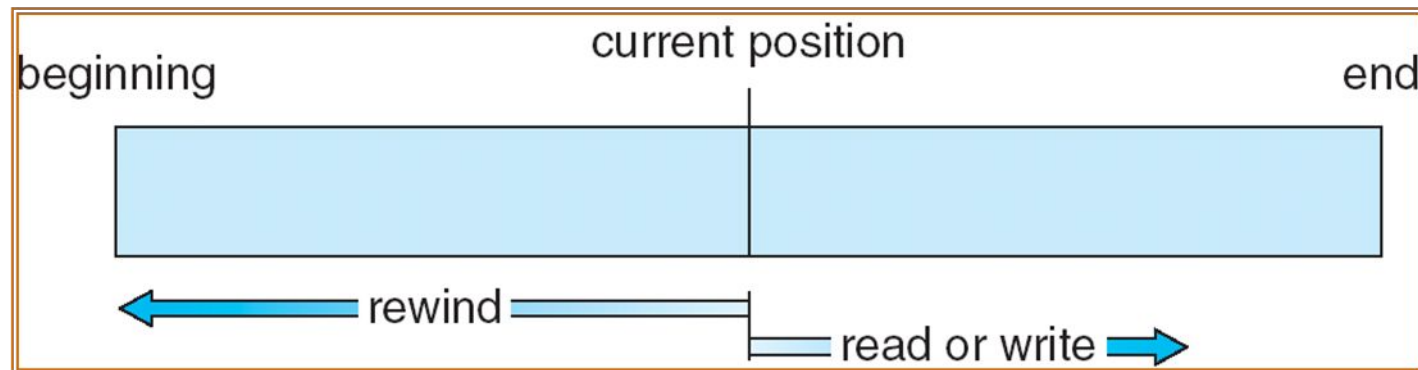
# Internal File Structure

- Disk files are accessed in units of physical blocks, typically 512 bytes or some power-of-two multiple thereof.
  - Larger physical disks use larger block sizes, to keep the range of block numbers within the range of a 32-bit integer.
- Internally files are organized in units of logical units, which may be as small as a single byte, or may be a larger size corresponding to some data record or structure size.
- The number of logical units which fit into one physical block determines its packing, and has an impact on the amount of internal fragmentation ( wasted space ) that occurs.
- As a general rule, half a physical block is wasted for each file, and the larger the block sizes the more space is lost to internal fragmentation.

# Access Methods

• Sequential Access- A sequential access file emulates magnetic tape operation, and generally supports a few operations:

- read next - read a record and advance the tape to the next position.
- write next - write a record and advance the tape to the next position.
- rewind
- skip n records - May or may not be supported. N may be limited to positive numbers, or may be limited to +/- 1.



# Direct Access

- Jump to any record and read that record. Operations supported include:
  - read n - read record number n. ( Note an argument is now required. )
  - write n - write record number n. ( Note an argument is now required. )
  - jump to record n - could be 0 or the end of file.
  - Query current record - used to return back to this record later.
  - Sequential access can be easily emulated using direct access. The inverse is complicated and inefficient.

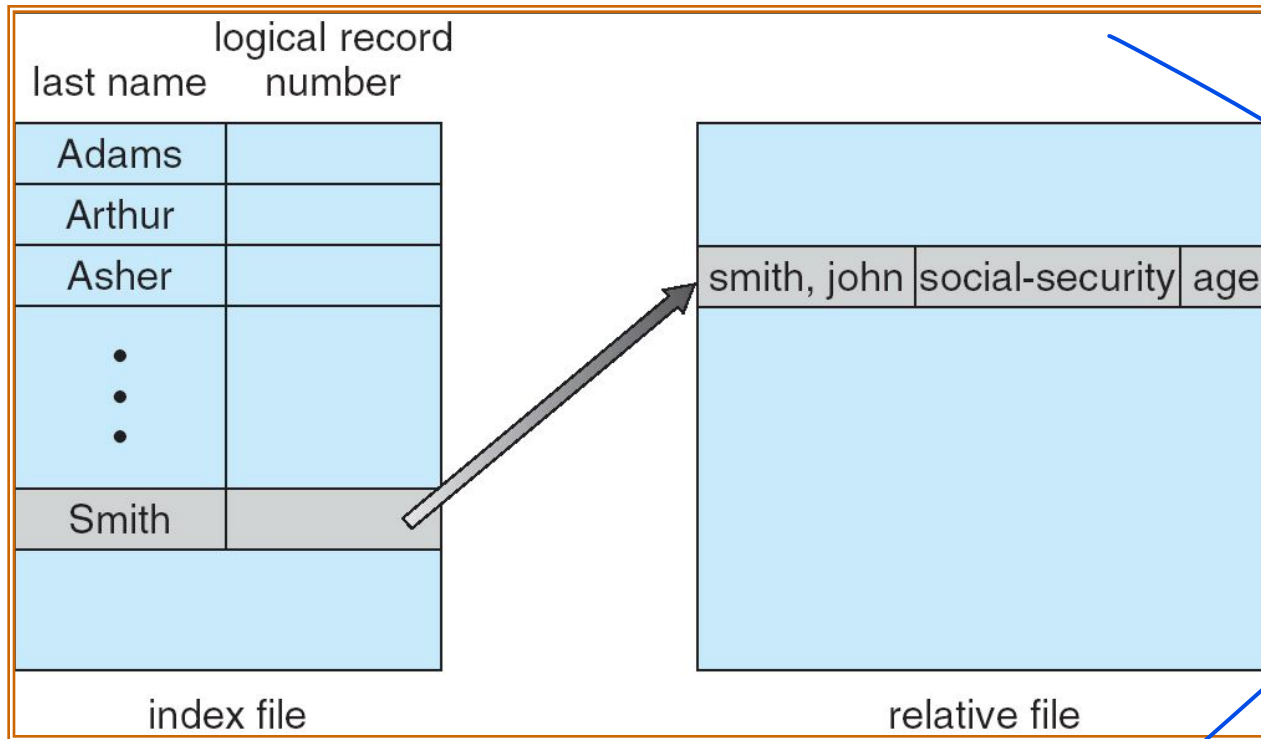
# Simulation of Sequential Access on a Direct-access File

sequential access	implementation for direct access
<i>reset</i>	<i>cp</i> = 0;
<i>read next</i>	<i>read cp</i> ; <i>cp</i> = <i>cp</i> + 1;
<i>write next</i>	<i>write cp</i> ; <i>cp</i> = <i>cp</i> + 1;

## Other Access Methods

- An indexed access scheme can be easily built on top of a direct access system.
- Very large files may require a multi-tiered indexing scheme, i.e. indexes of indexes.

# Example of Index and Relative Files



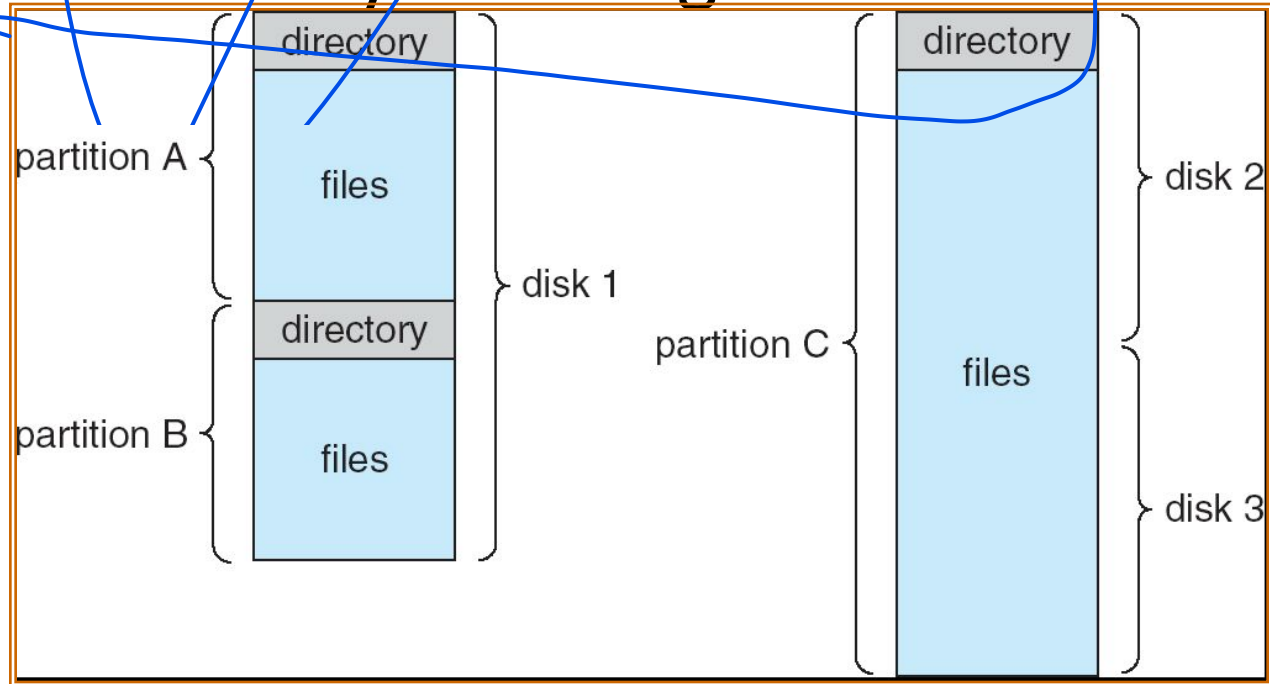


# Directory Structure

- **Storage Structure**

- A disk can be used in its entirety for a file system.
- Alternatively a physical disk can be broken up into multiple partitions, slices, or mini-disks, each of which becomes a virtual disk and can have its own filesystem.
- Multiple physical disks can be combined into one volume, i.e. a larger virtual disk, with its own filesystem spanning the physical disks.

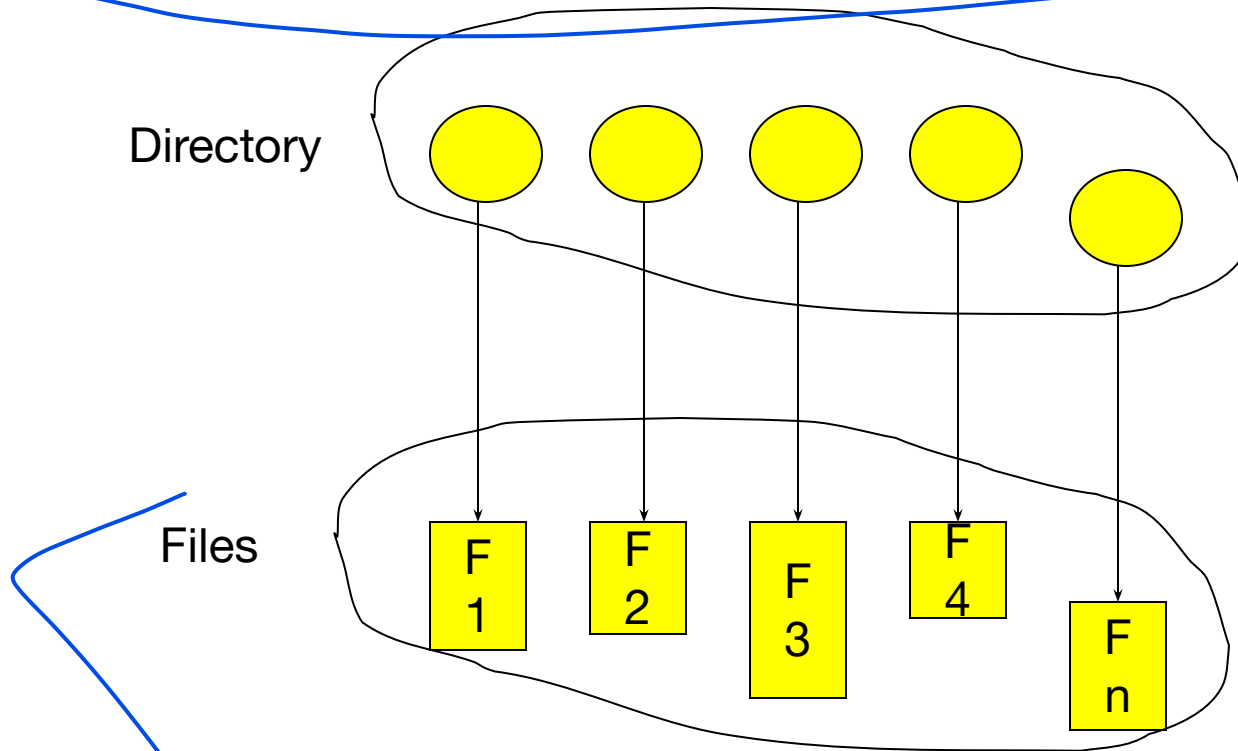
# A Typical File-system Organization



- 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*

# Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

# Directory Overview

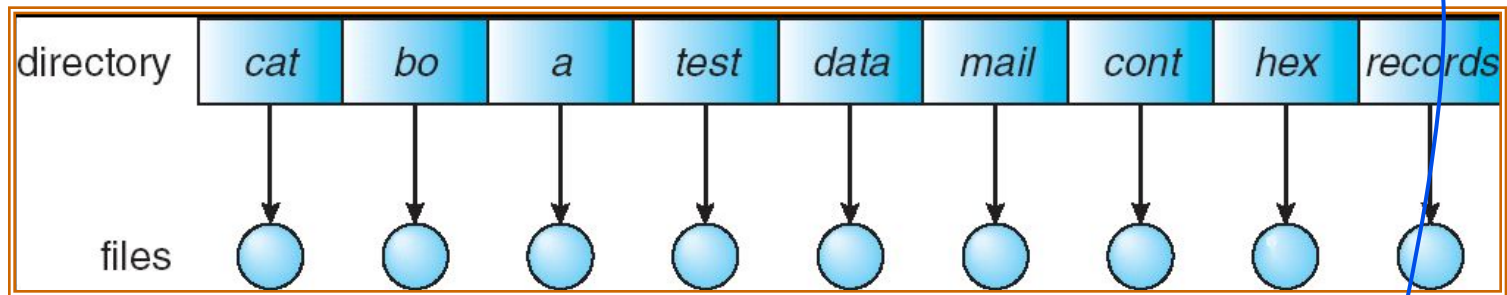
- Directory operations to be supported include:
  - Search for a file
  - Create a file - add to the directory
  - Delete a file - erase from the directory
  - List a directory - possibly ordered in different ways.
  - Rename a file - may change sorting order
  - Traverse the file system.

## 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, ...)

# Single-Level Directory

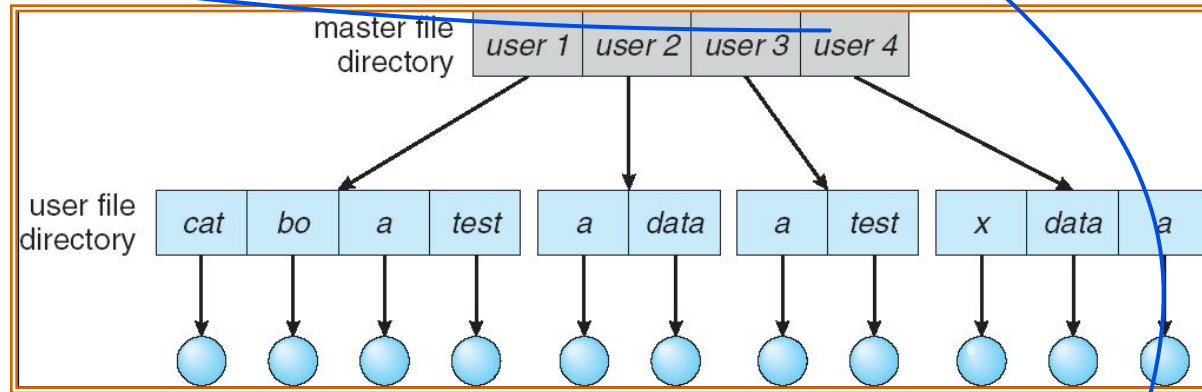
- A single directory for all users
- Called the root directory



- Pros: Simple, easy to quickly locate files
- Cons: inconvenient naming (uniqueness), no grouping

# Two-Level Directory

- Separate directory for each user



- Introduces the notion of a path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

# Two-Level Directory

- Advantages

- Solves the name-collision problem.
- Isolates users from one another ! a form of protection.
- Efficient searching.

- Disadvantages

- Restricts user cooperation.
- No logical grouping capability (other than by user)



## Path Name

- If a user can access another user's files, the concept of path name path name path name is needed.
- In two-level directory, this tree structure has MFD as root of path through UFD to user file name at leaf.
- Path name :: username + filename
- Standard syntax -- /user/file.ext

## Add Partitions

- Additional syntax needed to specify partition
  - e.g. in MS-DOS C:\user\file.ext file.ext

## System files

- Dotted files in Unix

# Path Name

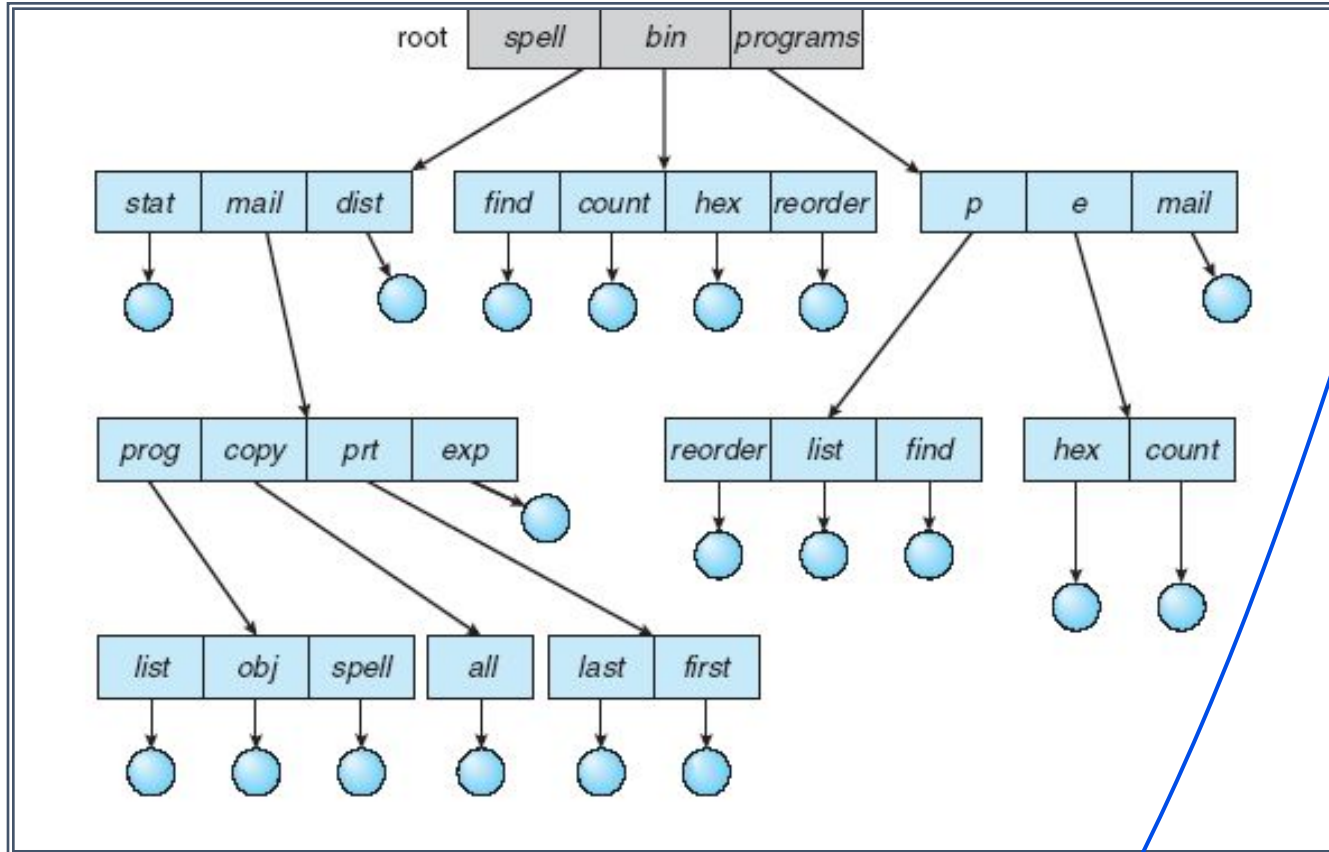
## System File Issues

- Those programs provided as part of the system (e.g. loaders, compilers, utility routines)
- e.g., Dotted files in Unix
- Another tradeoff issue
  - ◆ Copy all system files into each UFD OR
  - ◆ Create special user file directory that contains the system files.
    - ✓ Note: This complicates the file search procedure.
    - ✓ Default is to search local UFD, and then special UFD.
- To override this default search scheme, the user specifies a specific sequence of directories to be searched when a file is named – the search path.

# Tree-Structured Directories

- This generalization to a directory tree structure of arbitrary height allows users to create their own subdirectories and organize their files accordingly
- Directory
  - Becomes simply another file.
  - Contains a set of files or subdirectories.
  - All directories have the same internal format.
  - One bit in directory entry defines entry as file or directory.
  - Special commands are used to create and delete directories.

# Tree-Structured Directories



- Directories can now contain files and subdirectories
- Efficient searching, allows grouping

# Tree-Structured Directories

- Advantages

- Efficient searching

- Grouping Capability

- Each user has a current directory (working directory)

- cd /spell/mail/prog

- type list

## Path Names

- To access a file, the user should either:
  - Go to the directory where file resides, or
  - Specify the **path** where the file is
- Path names are either absolute or relative
  - Absolute: path of file from the root directory
  - Relative: path from the current working directory
- Most OSes have two special entries in each directory:
  - "." for current directory and ".." for parent

# Path Names

- **Absolute** or **relative** path name

- Creating a new file is done in current directory.

- Delete a file

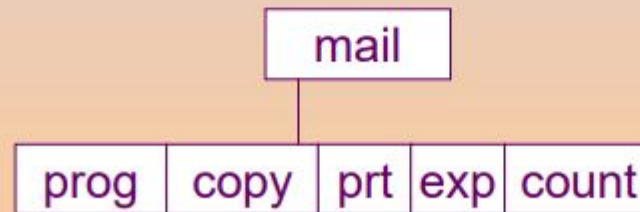
**rm** <file-name>

- Creating a new subdirectory is done in current directory.

**mkdir** <dir-name>

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

**mkdir** count

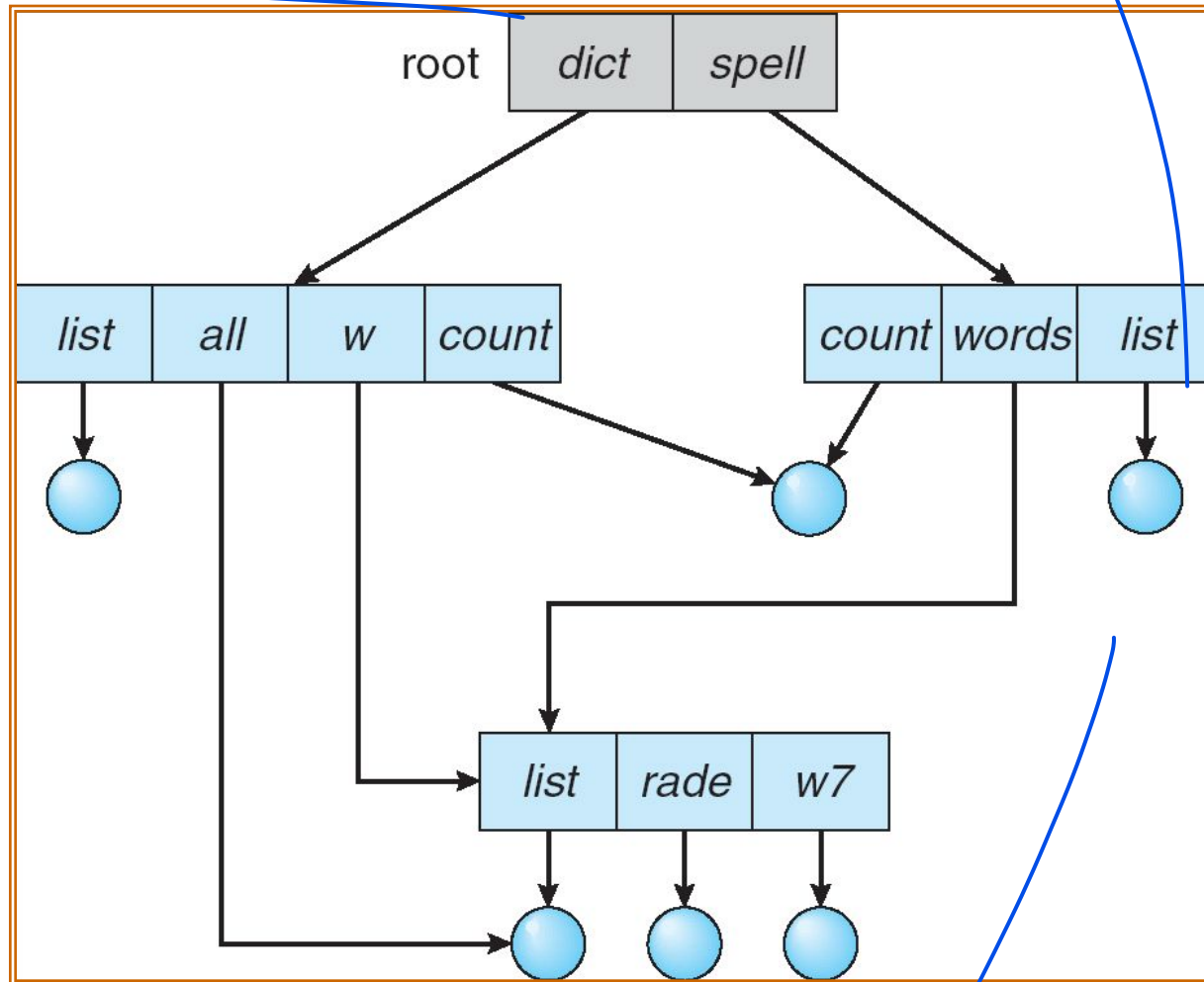


Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”.



# Acyclic-Graph Directories

- Allow sharing of subdirectories and files





# Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure
  - e.g. because they are being shared by more than one user / process
- it can be useful to provide an acyclic-graph structure.
- UNIX provides two types of links for implementing the acyclic-graph structure.
  - A hard link involves multiple directory entries that both refer to the same file.
    - Hard links are only valid for ordinary files in the same filesystem.
  - A symbolic link, that involves a special file, containing information about where to find the linked file.
    - Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.

## Acyclic-Graph Directories (Cont.)

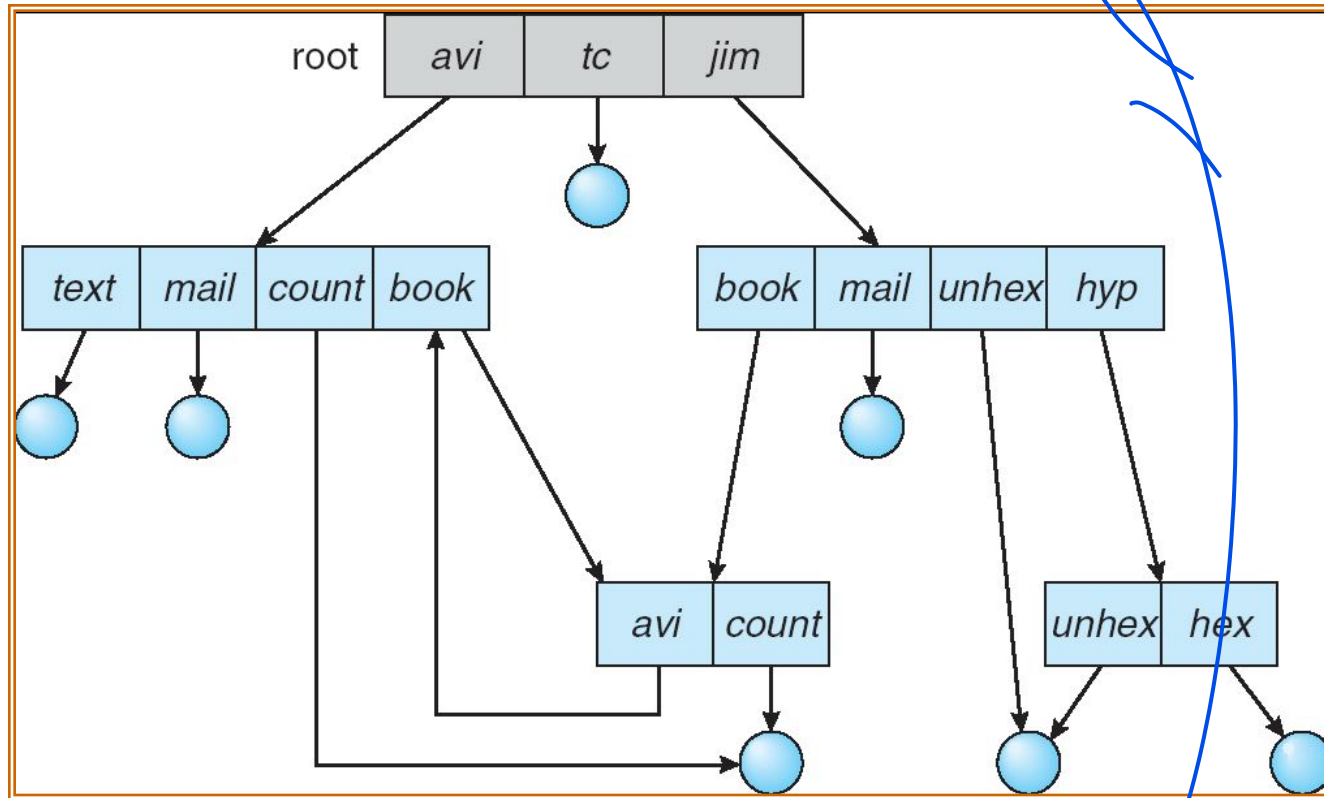
- Hard links require a reference count, or link count for each file
  - keeping track of how many directory entries are currently referring to this file.
- Whenever one of the references is removed the link count is reduced, and when it reaches zero
  - the disk space can be reclaimed.

# Acyclic-Graph Directories

---

- *Upon traversal of file system, do not want to traverse shared structures more than once (e.g., doing backups or accumulating file statistics).*
- *On deletion, which action to take?*
  - ◆ *Option1: remove file when anyone issues delete → possible dangling pointer to non-existent file.*
  - ◆ *Option2: [UNIX] use symbolic links → links are left when file is deleted and user has to “realize” that original file is gone.*
  - ◆ *Option3: maintain a file reference list containing one entry for each reference to the file {disadvantages – variable and large list}.*
  - ◆ *Option4: keep a count of the number of references. When count=0, file is deleted.*

# General Graph Directory



# General Graph Directory

---

- If cycles are allowed in the graphs, then several problems can arise:

- Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms.

- not to follow symbolic links

- to only allow symbolic links to refer to directories

- Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero.

- Periodic garbage collection is required to detect and resolve this problem.

- chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system.

- Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.

## General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to files 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

- Basic idea behind mounting file systems is to combine multiple file systems into one large tree structure.
- mount command is given a filesystem to mount and a mount point ( directory ) on which to attach it.
- Once a file system is mounted onto a mount point, any further references to that directory actually refer to the root of the mounted file system.
- Any files ( or sub-directories ) that had been stored in the mount point directory prior to mounting the new filesystem are now hidden by the mounted filesystem, and are no longer available.
  - For this reason some systems only allow mounting onto empty directories.

# File-System Mounting

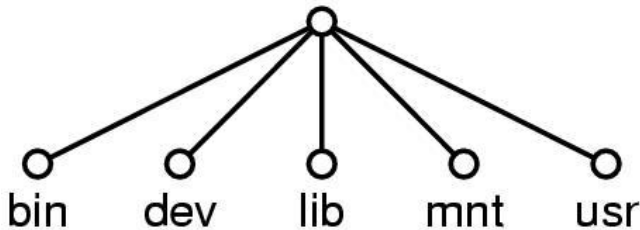
- Filesystems can only be mounted by root
  - E.g. root may allow users to mount floppy filesystems to /mnt
- Anyone can run the mount command to see what filesystems are currently mounted.
- Filesystems may be mounted read-only, or have other restrictions imposed.



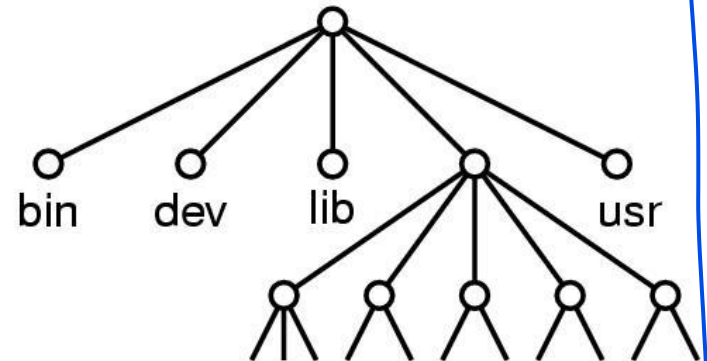
# File System Mounting

- 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)

# File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
  - **Group IDs** allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

# File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - 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 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

# File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called metadata
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

# Consistency Semantics

- Consistency Semantics deals with the consistency between the views of shared files on a networked system.
- When one user changes the file, when do other users see the changes?

# UNIX Semantics

- The UNIX file system uses the following semantics:
  - Writes to an open file are immediately visible to any other user who has the file open.
  - One implementation uses a shared location pointer, which is adjusted for all sharing users.
- The file is associated with a single exclusive physical resource, which may delay some accesses.

# Session Semantics

---

- The Andrew File System, AFS uses the following semantics:
  - Writes to an open file are not immediately visible to other users.
  - When a file is closed, any changes made become available only to users who open the file at a later time.
- According to these semantics, a file can be associated with multiple views.
- Almost no constraints are imposed on scheduling accesses.
- No user is delayed in reading or writing their personal copy of the file.
- AFS file systems may be accessible by systems around the world.
- Access control is maintained through complicated access control lists, which may grant access to the entire world or to specifically named users accessing the files from specifically named remote environments.

# Immutable-Shared-Files Semantics

- Under this system, when a file is declared as shared by its creator,
  - it becomes immutable and the name cannot be re-used for any other resource.
  - Hence it becomes read-only, and shared access is simple.



# Protection

---

- Files must be kept safe for reliability ( against accidental damage ), and protection ( against deliberate malicious access. )
    - The former is usually managed with backup copies.
  - One simple protection scheme is to remove all access to a file.
    - However this makes the file unusable, so some sort of controlled access must be arranged.
-

# Types of Access

- The following low-level operations are often controlled:

- Read - View the contents of the file
  - Write - Change the contents of the file.
  - Execute - Load the file onto the CPU and follow the instructions contained therein.
  - Append - Add to the end of an existing file.
  - Delete - Remove a file from the system.
  - List -View the name and other attributes of files on the system.
- Higher-level operations, such as copy, can generally be performed through combinations of the above.

# Access Control

- One approach is to have complicated Access Control Lists, ACL, which specify exactly what access is allowed or denied for specific users or groups.
  - AFS uses this system for distributed access.
  - Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown.
- UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others.

# 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

# UNIX Access Rights

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

RWX

a) **owner access** 7  $\Rightarrow$  1 1 1

RWX

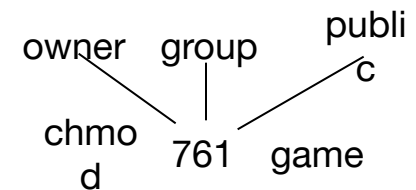
b) **group access** 6  $\Rightarrow$  1 1 0

RWX

c) **public access** 1  $\Rightarrow$  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`



# UNIX Access Rights

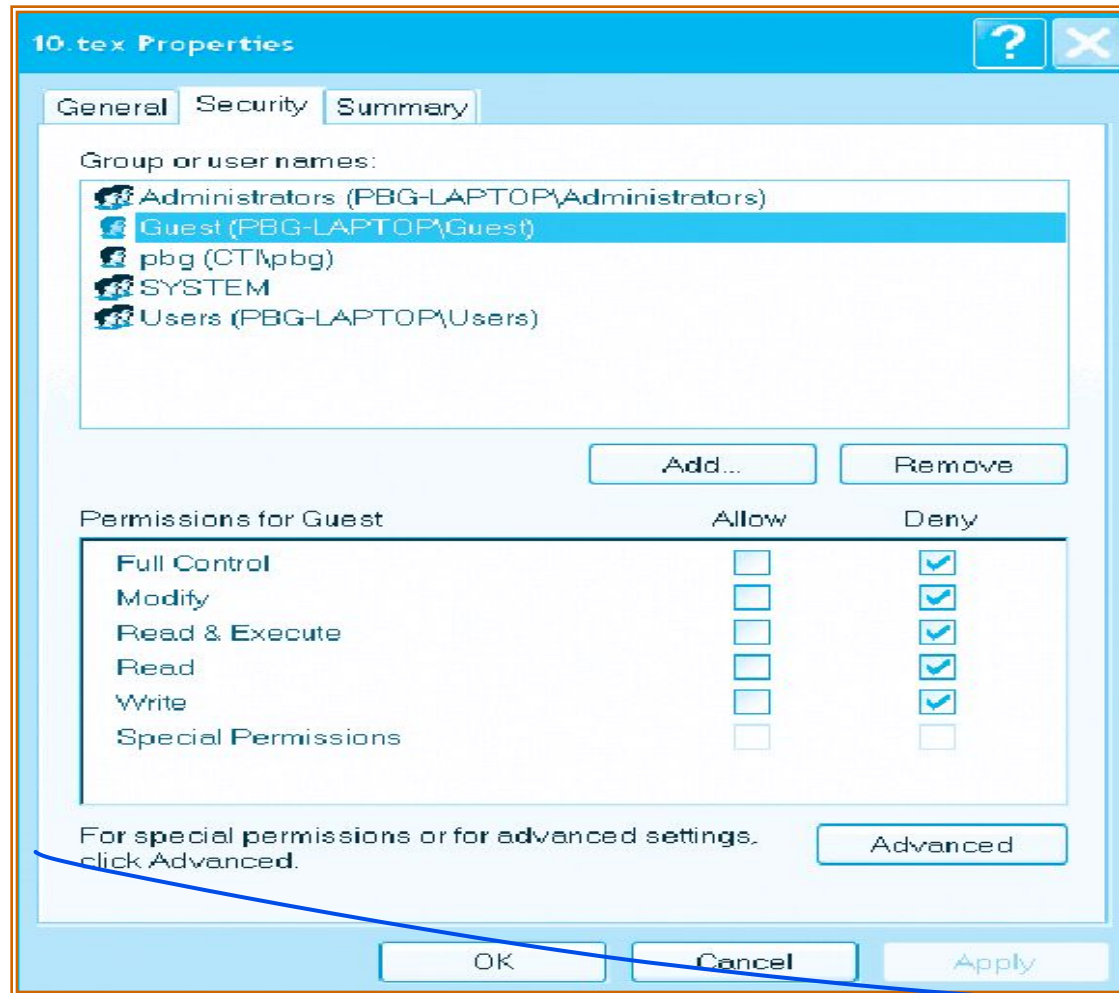
- In addition there are some special bits that can also be applied:
  - The set user ID ( SUID ) bit and/or the set group ID ( SGID ) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program.
  - The sticky bit on a directory modifies write permission, allowing users to only delete files for which they are the owner. This allows everyone to create files in /tmp, for example, but to only delete files which they have created, and not anyone else's.
  - The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for execute permission for the user, group, and others, respectively.
  - If the letter is lower case, ( s, s, t ), then the corresponding execute permission is not also given. If it is upper case, ( S, S, T ), then the corresponding execute permission IS given.

# Issues with UNIX Access Rights

---

- Just a single owner, a single group and the public
    - Pro: Compact enough to fit in just a few bytes
    - Con: Not very expressive
  - *Access Control List*: This is a per-file list that tells who can access that file
    - Pro: Highly expressive
    - Con: Harder to represent in a compact way
-

# Windows XP Access-control List Management





# End of Chapter 10

---