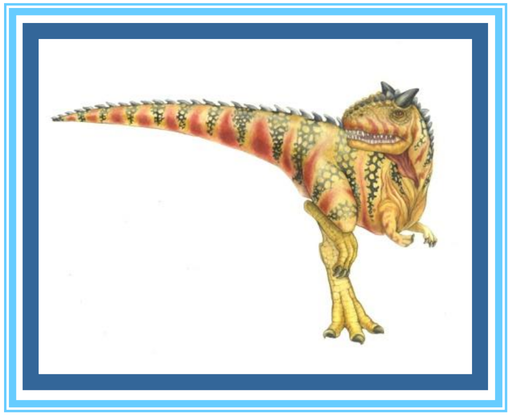
Chapter 13: File-System Interface

Section 13.1-13.3



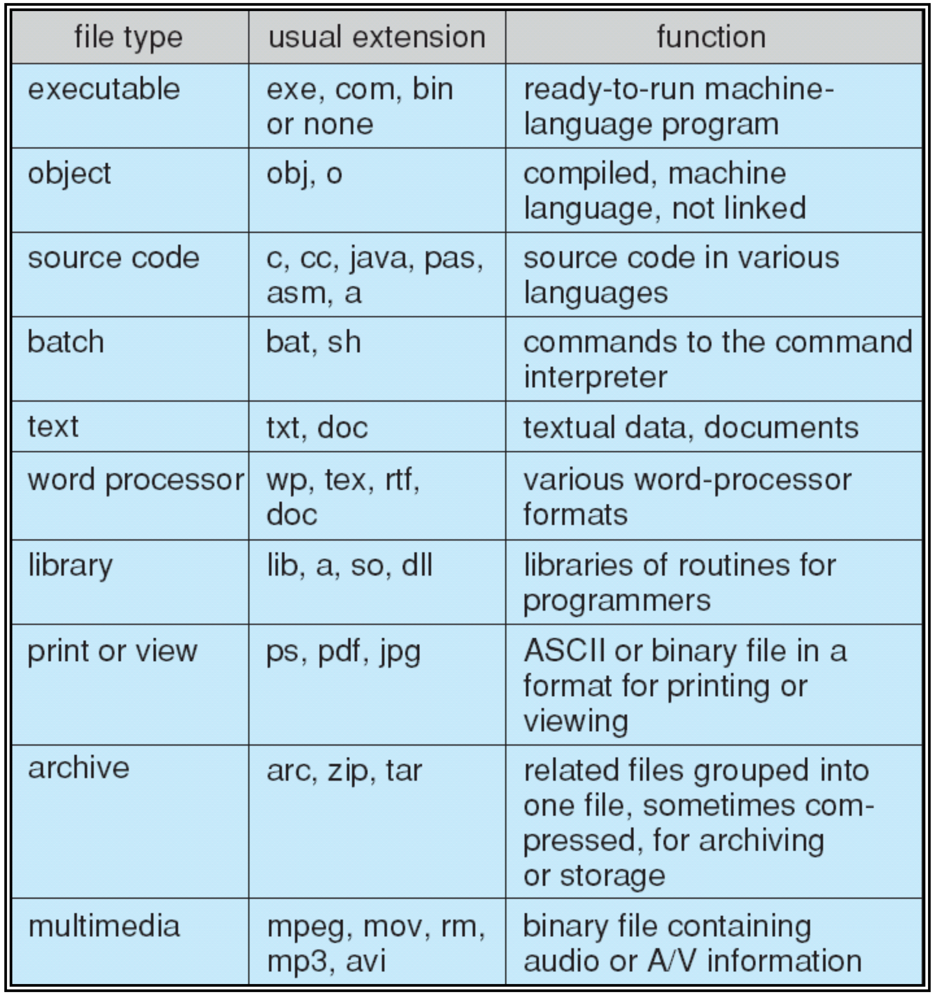
What is a File System?

- The file system provides the mechanism for on-line storage of and access to data and programs of the OS and the users

- File system consists of two parts
  - A collection of files
  - A directory structure providing information about all the files in the system

- A file is a named collection of related information that is recoded on secondary storage
  - Many types: text, photos, music, etc.

File Types – Name, Extension



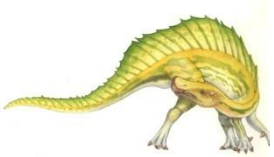
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual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

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
- **Timestamps and user identification** – info kept for creation, last modification, and last use, useful for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on secondary storage

File Operations

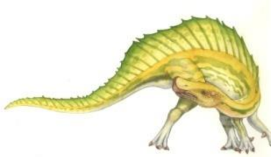
- **Create** – allocate space and create a directory entry for the file
- **Write –** system keeps a **write pointer** to location in file where next write is to take place; write pointer is updated after each write
- **Read –** system keeps a **read pointer** to location in file where next read is to take place; read pointer is updated after each read
  - Usually system keeps a per-process **file pointer** used by both read and write operations
- **Reposition within file** – the file pointer is repositioned to a given value (**seek**)
- **Delete** – release file space and erase the directory entry
- **Truncate –** release file space, set file size to 0, keep all other file attributes unchanged



Open-File Tables

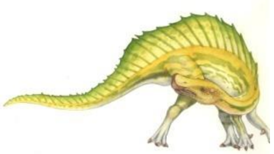
- Most systems require programmers to open a file with the **open()** system call before the file can be used

- Several processes may open the same file simultaneously → OS uses two levels of **open-file tables** to track open files
  - **System-wide open-file table**: stores process-independent info, e.g., file location on disk, access time, file size
    - Also stores an **open count** to indicate how many processes have the file open
  - **Per-process open-file table**: stores file pointer (to next read/write location), access mode (read-only, write-only, read-write)



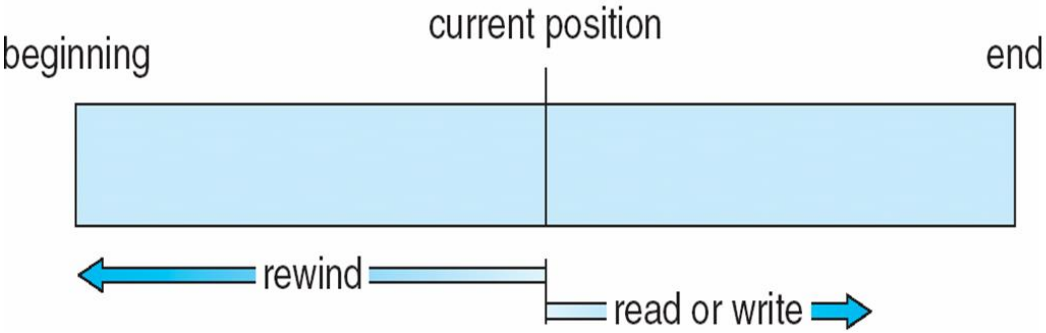
Open and Close Operations

- **Open(F)**
  - Search the directory structure for entry **F**, and copy the content of entry to system-wide open-file table
  - Creates an entry in the per-process open-file table, make it point to the entry in system-wide open-file table
  - Returns a pointer to the entry in the per-process open-file table, pointer is used in all subsequent file operations
- **Close (F)**
  - Remove file’s entry from per-process open-file table
  - Decrease open count in system-wide open-file table
  - If open count reaches 0, copy the content of entry **F** from system-wide open-file table to directory structure on disk and remove entry from the system-wide open-file table



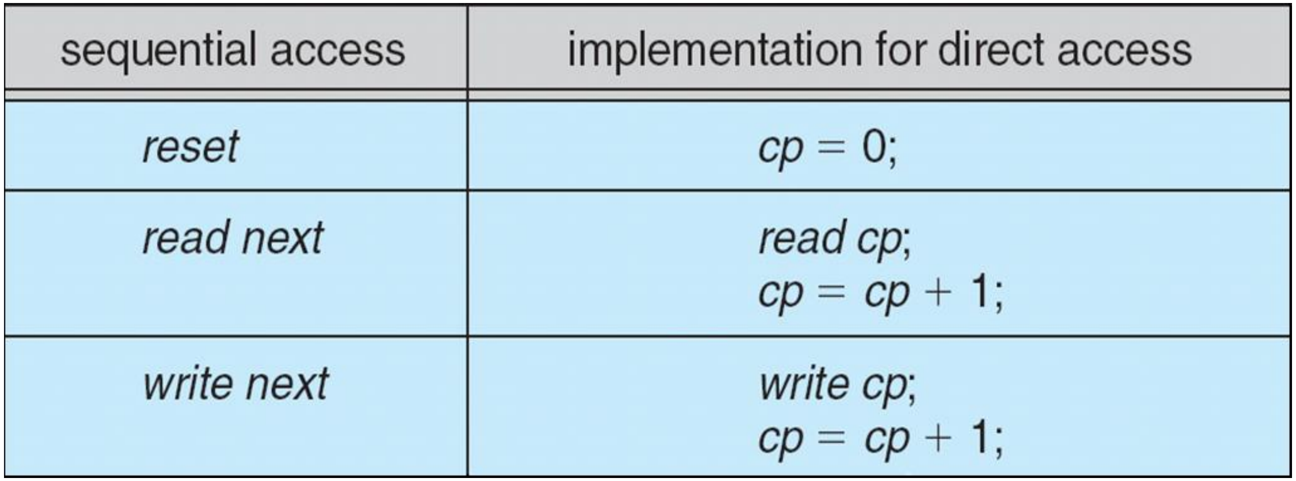
Access Methods

- **Sequential Access** – Information in the file is processed in order, one record after the other
  - Operations: **`read_next, write_next, reset`**



- **Direct Access –** File is viewed as a sequence of blocks, programs can read and write blocks in no particular order
  - Operations: **`read(n), write(n),`** n is a relative block number

Simulation of Sequential Access on Direct-access File

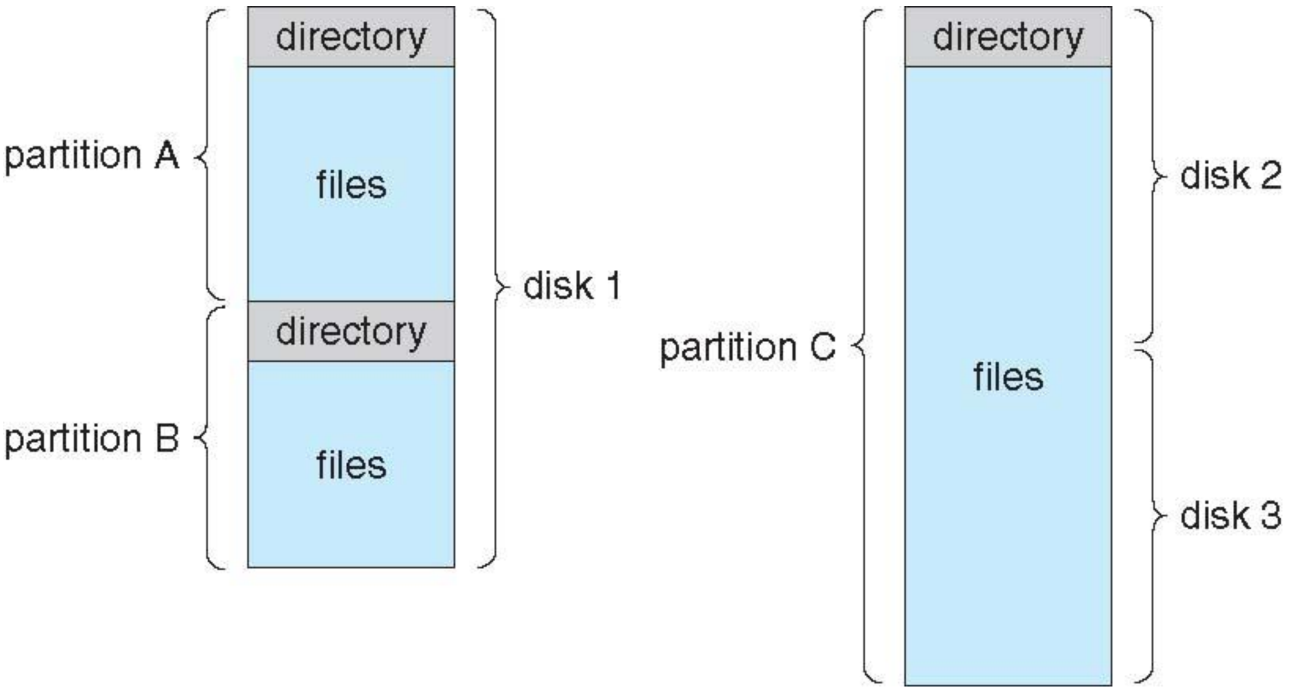


sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

cp defines current position in file

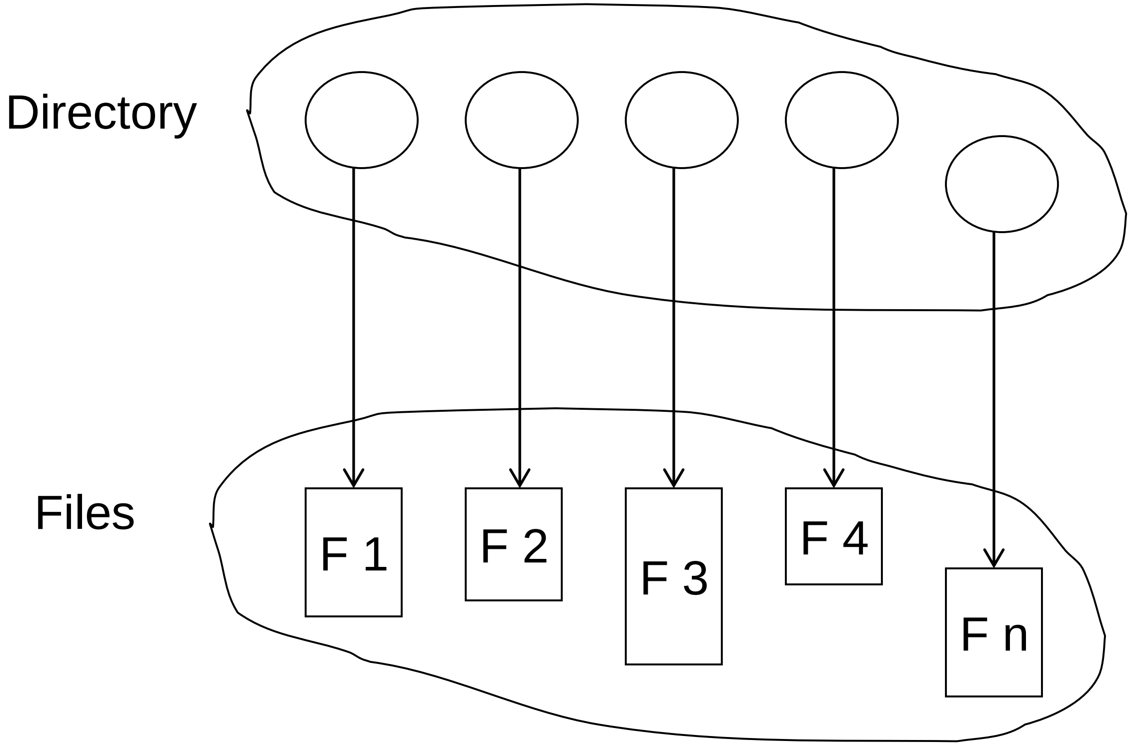
Disk Structure

- Disk can be subdivided into **partitions;** A partition may span multiple disks
- A partition can be **formatted** with a file system
- A file system consists of a collection of files and a directory structure



Directory Structure

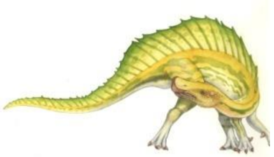
- A directory consists of a collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Directory Operations

- **Search for a file**
- **Create a file** – new file need to be added to directory
- **Delete a file** – need to remove the file from the directory
- **List a directory** – list the files in a directory and the contents of the directory entry for each file in the list
- **Rename a file** – change the name of the file
- **Traverse the file system** – access every directory and every file in a directory structure



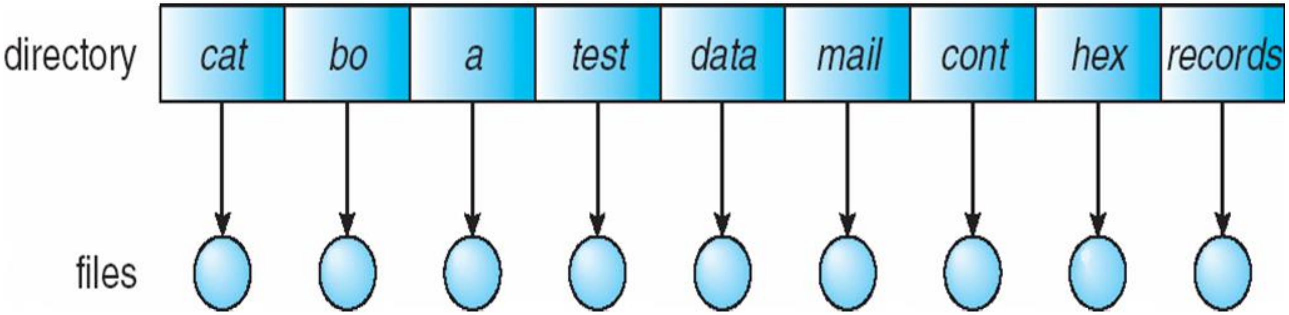
Directory Organization

The directory is organized 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

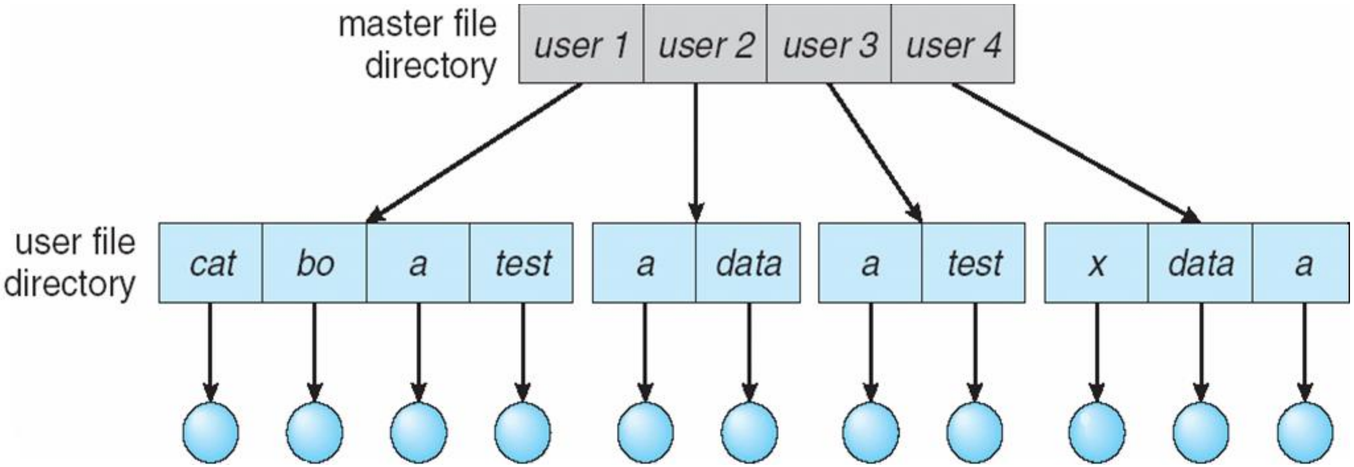
- A single directory for all users



- Naming problem – all files must have unique names
- Grouping problem

Two-Level Directory

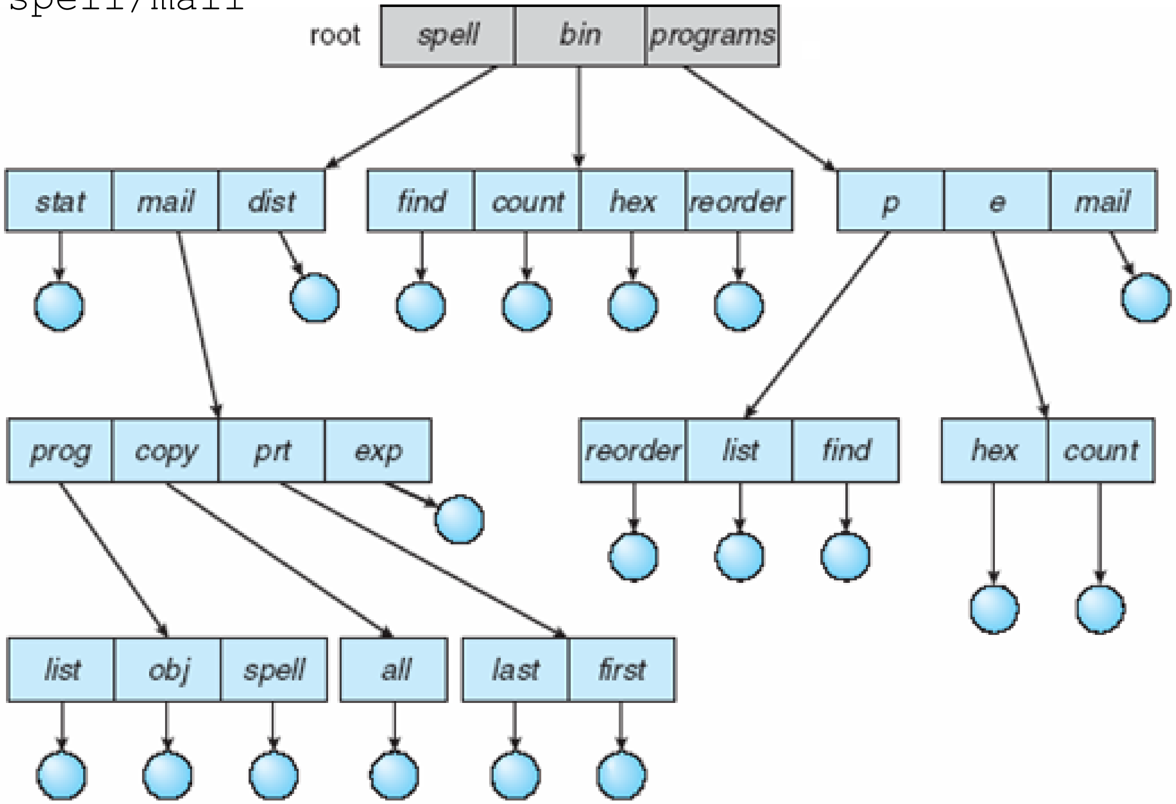
- Separate directory for each user



- Can have the same file name for different user
- Every file has a unique path name consisting of a user name and a file name
    - A user can use path names to refer to files of other users
- Efficient searching – search is done within a user's own directory
- No grouping capability

Tree-Structured Directories

- Provides grouping capability – users can create their own subdirectories to group their files
- Each process has a **current directory**, which can be changed using `cd` command
- Each file has a unique path name, which can be of two types
  - **Absolute path name** begins at the root, e.g., `/spell/mail/copy/all`
  - **Relative path name** begins at the current directory, e.g., `copy/all` with current directory `/spell/mail`



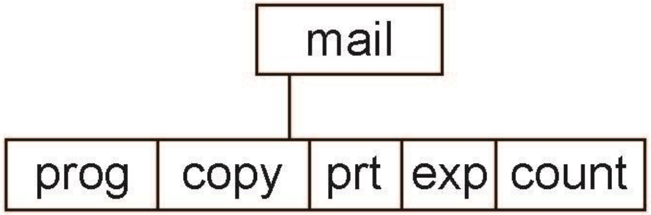
Tree-Structured Directories (Cont)

- Efficient searching: current directory is searched when reference is made to a file
- Creating a new file is done in current directory
- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory `/mail`

```
mkdir count
```



mail

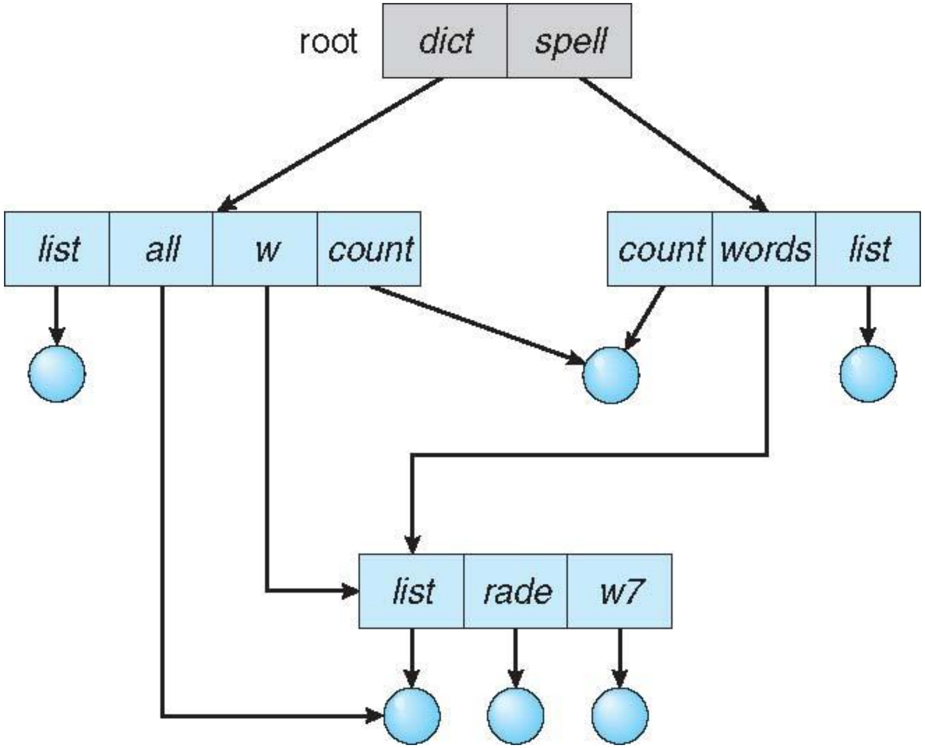
prog copy prt exp count

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

- A user can access a file of another user by specifying an absolute or relative path name
  - E.g., if current directory is user1’s directory, then path name ../user2/cs352/homework1.pdf and /user2/cs352/homework1.pdf refer to same file of user2

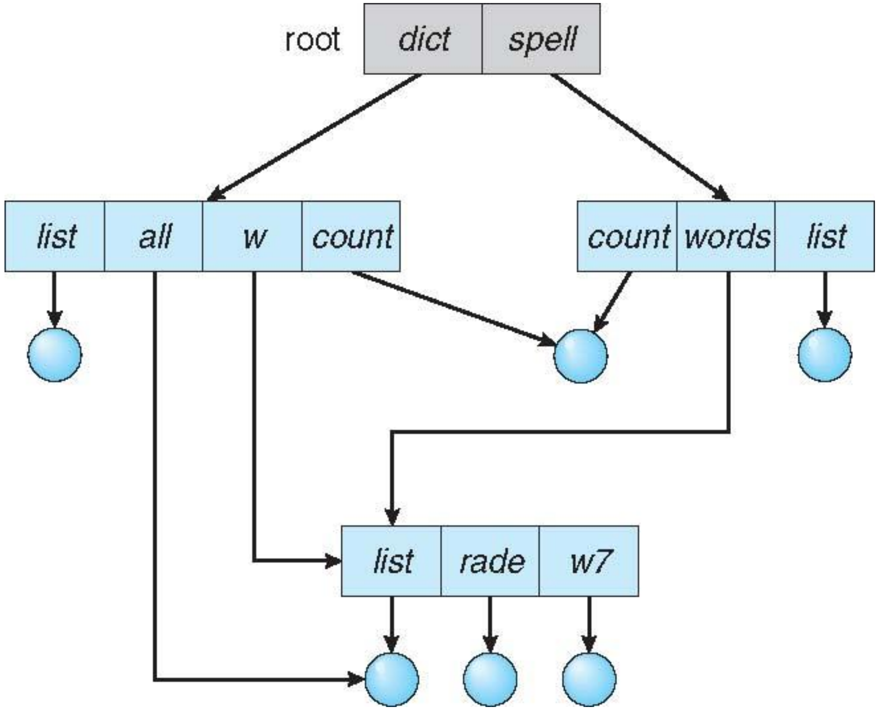
Acyclic-Graph Directories

- In a tree-structured directory, a file or subdirectory can be contained in only one directory
- Acyclic-graph directories allow shared files and subdirectories to be contained in more than one directory
  - A shared file/subdirectory has more than one name, but only one copy of the shared file/subdirectory exists
  - Can be implemented using **links** – a link is a pointer to an existing file or directory



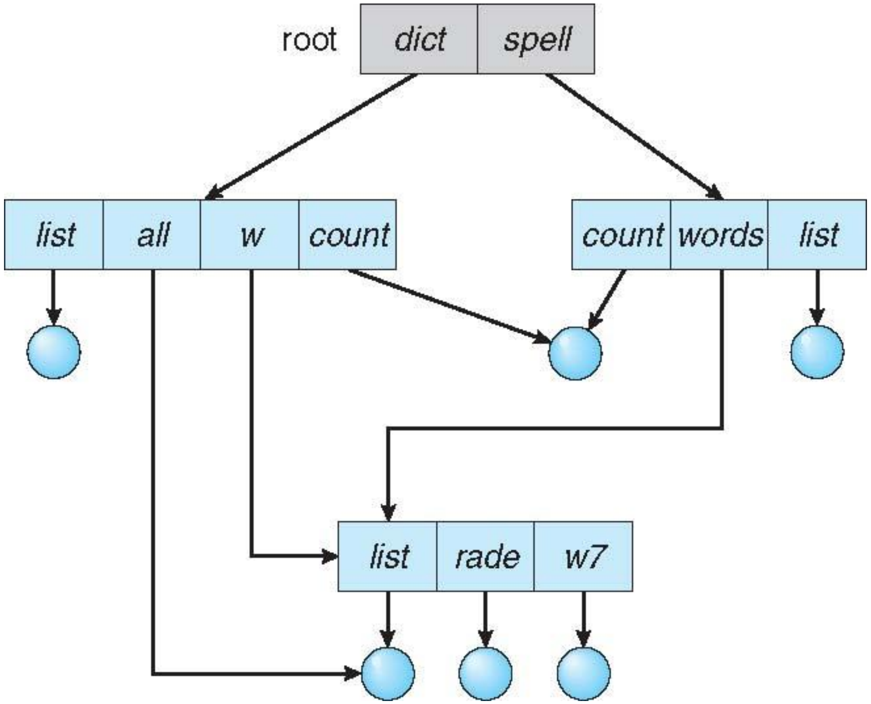
Hard Links

- Some file systems allow multiple absolute path names to refer to the same file, these absolute path names are called **hard links** to the file
  - E.g., the command ln /spell/count /dict/count creates a hard link named /dict/count to a file named /spell/count
  - /dict/count and /spell/count refer to the same physical file location. You can edit a file using any one of its names



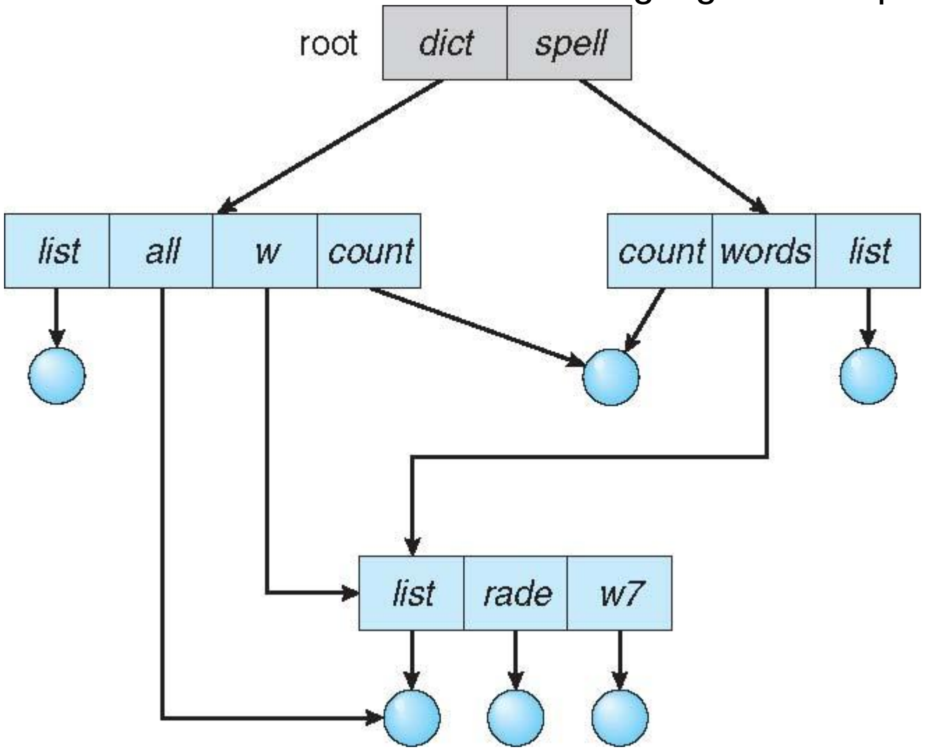
Hard Links (Cont.)

- ***count*** in ***spell*** is deleted ⇒ dangling pointer
- Solution:
  - Keep a count of number of references to a shared file
  - Adding a new link or directory entry increments the count
  - Deleting a link or directory entry decrements the count
  - File can be deleted when count is 0



Symbolic/Soft Links

- A **symbolic/soft link** is a special type of file that contains a reference to another file or directory in the form of an absolute path name
  - E.g., the command ln –s /spell/count /dict/count creates a symbolic link named /dict/count to a file named /spell/count
  - Soft link contains the path for the original file
  - When you edit /dict/count, /spell/count will get updated
  - Removing a soft link does not affect anything
  - Removing the original file causes the link to become dangling link that points to nonexistent file


root
dict
spell
list
all
w
count
count
words
list
list
rade
w7