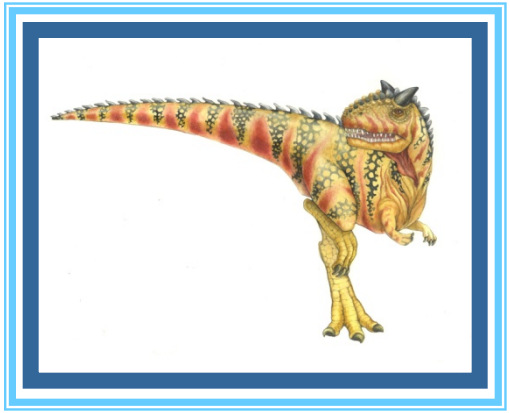
Understanding File-System Interface



File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- Protection

File Concept

- Named collection of related information – smallest allotment of logical secondary usage
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, object file, executable file**
    - A file often has a **structure**, e.g., sequence of characters

File Concept

```
// At this point, I'd like to take a moment to speak to you about the Adobe PSD format.
// PSD is not a good format. PSD is not even a bad format. Calling it such would be an
// insult to other bad formats, such as PCX or JPEG. No, PSD is an abysmal format. Having
// worked on this code for several weeks now, my hate for PSD has grown to a raging fire
// that burns with the fierce passion of a million suns.
// If there are two different ways of doing something, PSD will do both, in different
// places. It will then make up three more ways no sane human would think of, and do those
// too. PSD makes inconsistency an art form. Why, for instance, did it suddenly decide
```

https://github.com/kjk/xee/blob/master/XeePhotoshopLoader.m

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 (persistent)
- Many variations, including extended file attributes such as file checksum
- Extra information also kept in the directory structure

File info on Linux

```
  File: notes.txt
  Size: 39              Blocks: 2          IO Block: 1048576 regular file
Device: 5dh/93d Inode: 64392       Links: 1
Access: (0644/-rw-r--r--)  Uid: (21115/ mainack)   Gid: (21000/ns-science)
Access: 2020-04-26 20:31:20.812781321 +0200
Modify: 2020-04-26 20:31:20.814449217 +0200
Change: 2020-04-26 20:31:20.818990853 +0200
 Birth: -
```

- stat notes.txt
- file notes.txt
- ls –l notes.txt

RECAP

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 Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

Moving forward …

File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- Protection

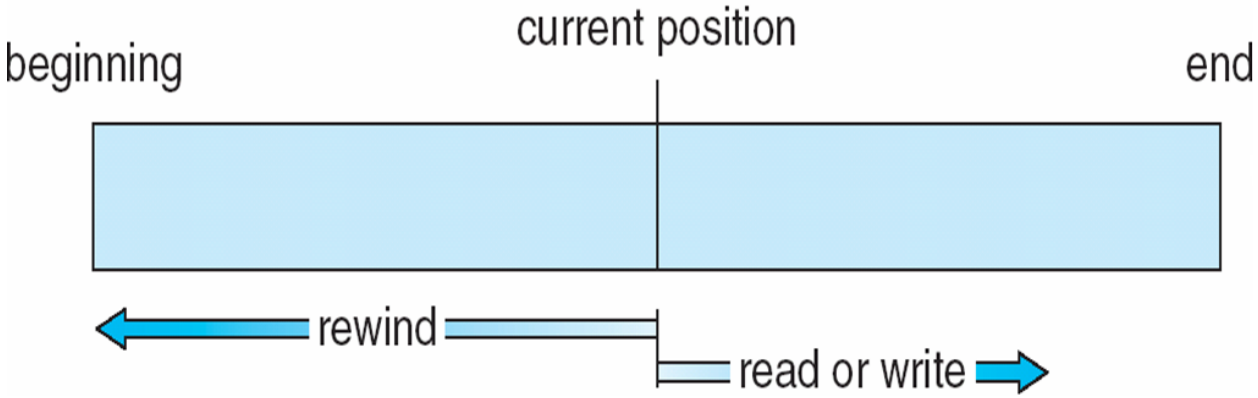
File Operations

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate (… > notes.txt** or **truncate –s <size> filename)** why?
- **Copying a file?**

Open Files

- ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory, e.g., open('notes.txt', 'r')
- ***Close (F_i)*** – remove the content of entry ***F_i*** from memory
- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files (per process or system?)
  - 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: cache of data access information
  - Access rights: per-process access mode information

Sequential-access File



Access Methods

- **Sequential Access**

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

- **Direct Access –** file is fixed length logical records

```
read n
write n
position to n
        read next
        write next
rewrite n
```

n = relative block number

- Relative block numbers allow OS to decide where file should be placed

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;

Other Access Methods

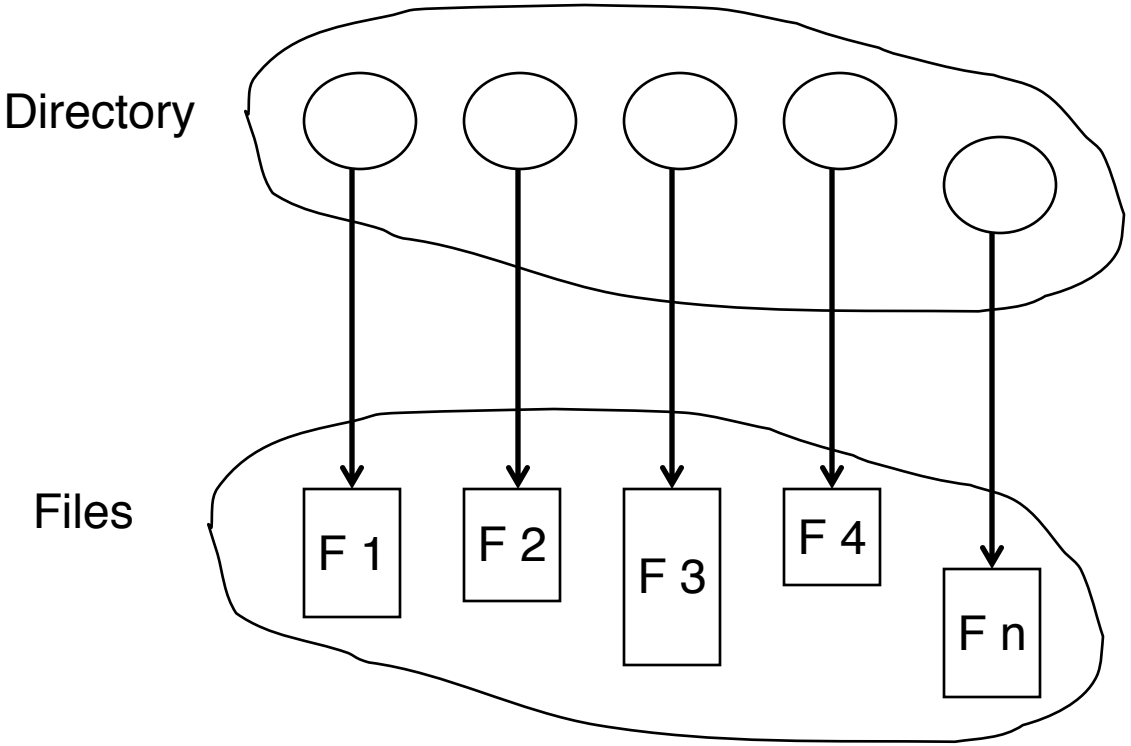
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on
  - Consider a set of records of grocery items
  - Each record = 10 digit product id + 6 digit price = 16 digits = 16 bytes
  - Each block = 1024 bytes => 1024/16 = 64 records
  - If 320,000 records => 320,000/64 = 5000 blocks
  - If the file is sorted by product id, we just need to store the first product id for each block
  - Then can do binary search on this smaller index to find a particular product
- If too large, index (in memory) of the index (on disk)

File-System Interface

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

Directory Structure

- A collection of nodes containing information about all files



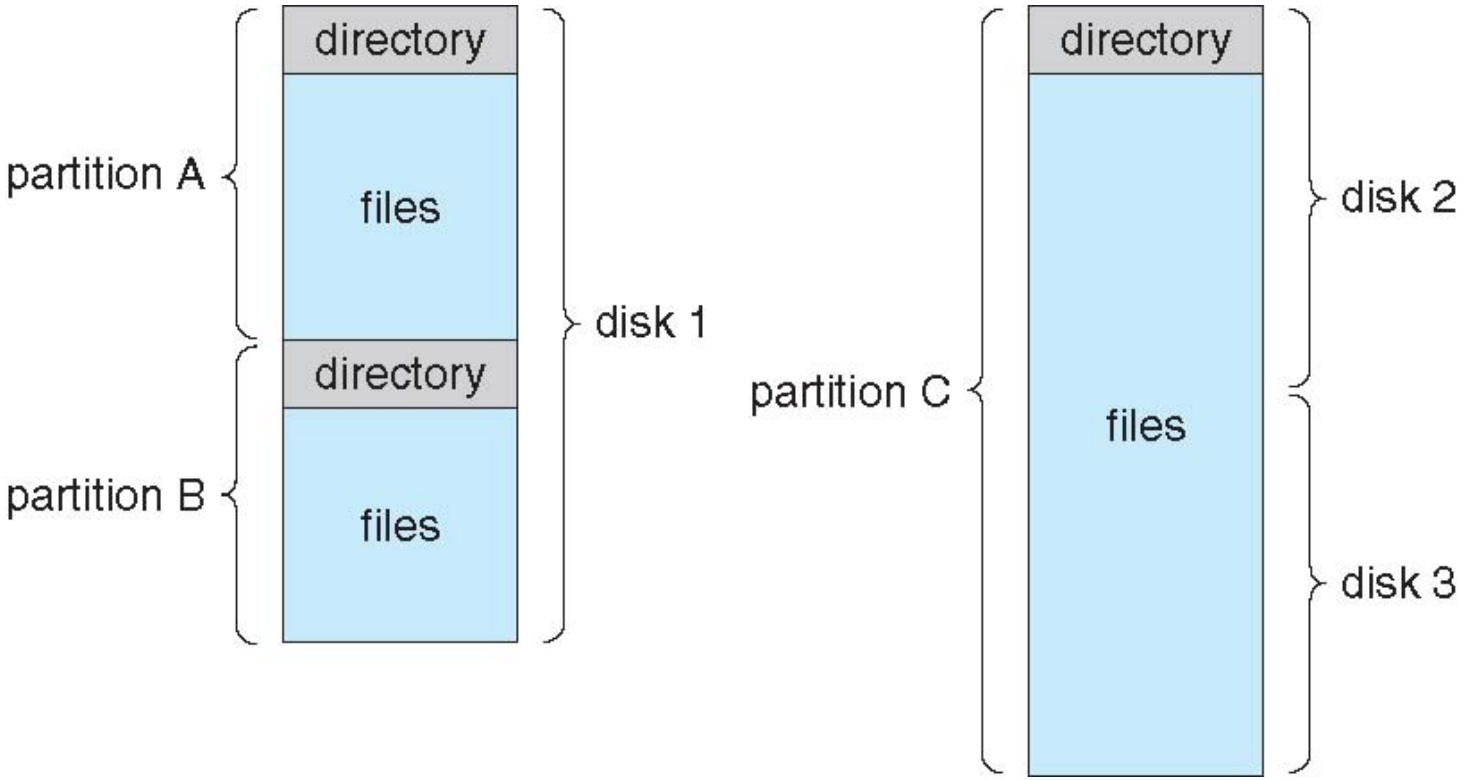
Both the directory structure and the files reside on disk

Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- 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 that file system's info 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

RECAP

A Typical File-system Organization



File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure (contd)
- File-System Mounting
- File Sharing
- Protection

Operations Performed on Directory

- Search for a file (command: find)
- Create a file (command: mkdir, touch)
- Delete a file (command: rm –rf)
- List a directory (command: ls)
- Rename a file (command: rename)
- Traverse the file system (command: rsync avz)

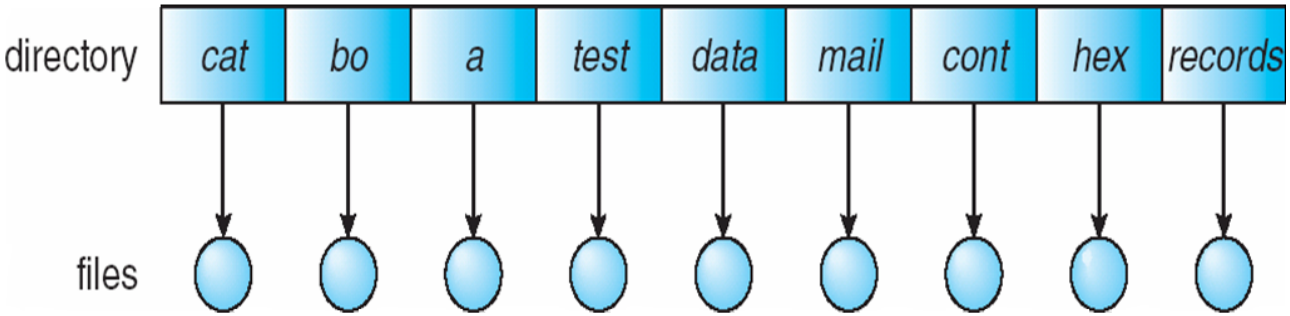
Directory Organization

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, all pdf files, …)

1. Single-Level Directory

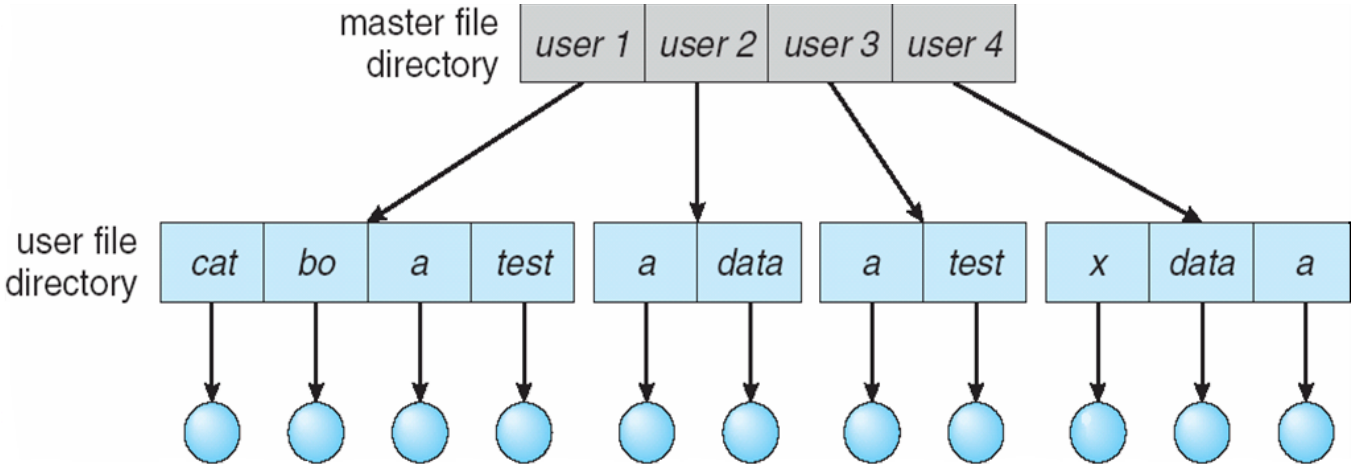
- A single directory for all users



- Naming problem
- Grouping problem

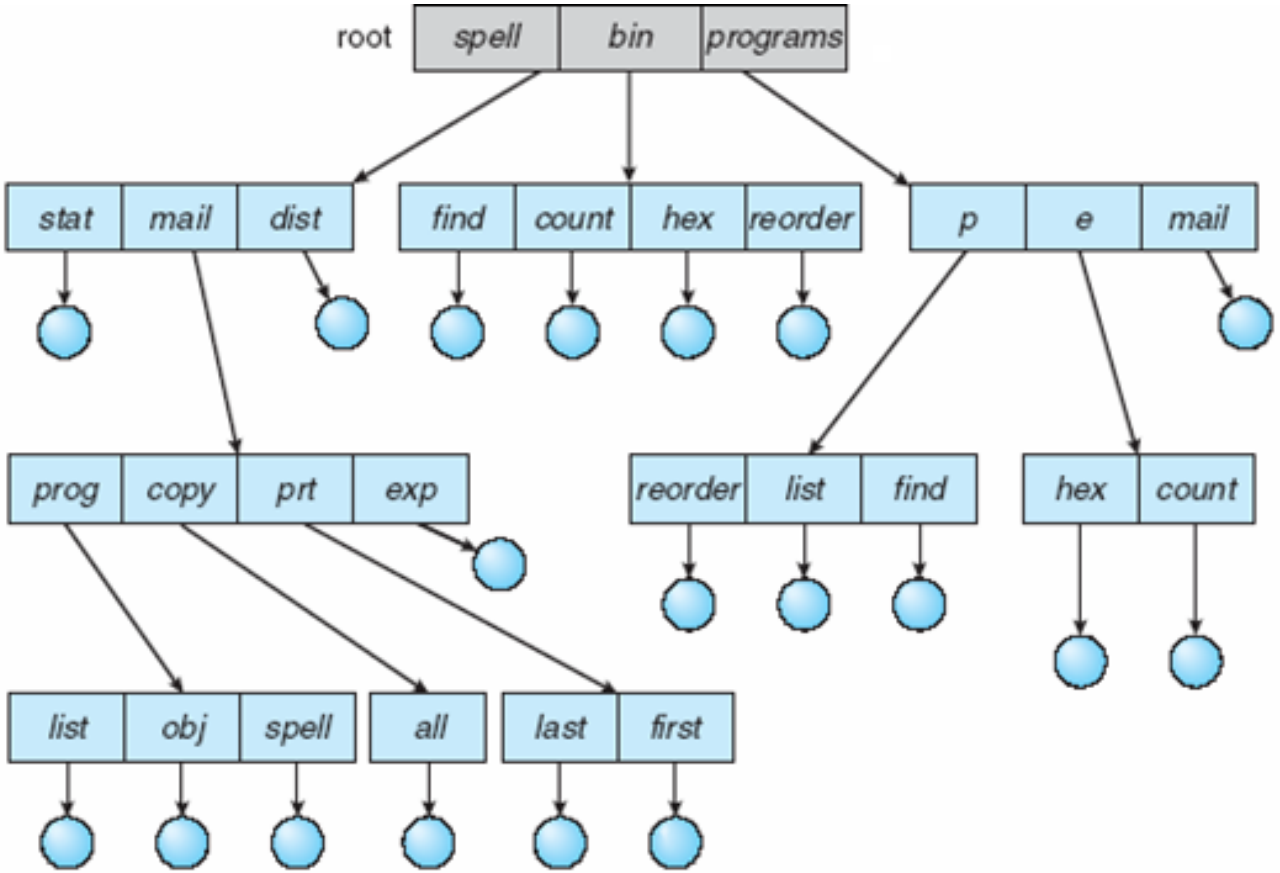
2. Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

3. Tree-Structured Directories



root/spell/mail/prog/obj/filename

4. Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory) [command: pwd]
  - `cd /spell/mail/prog` [change directory]

4. Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory [command: mkdir]
- Delete a file

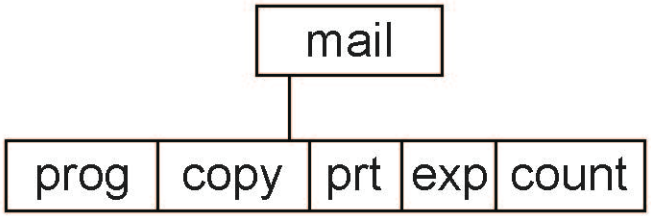
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

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

Example: if in current directory `/mail`

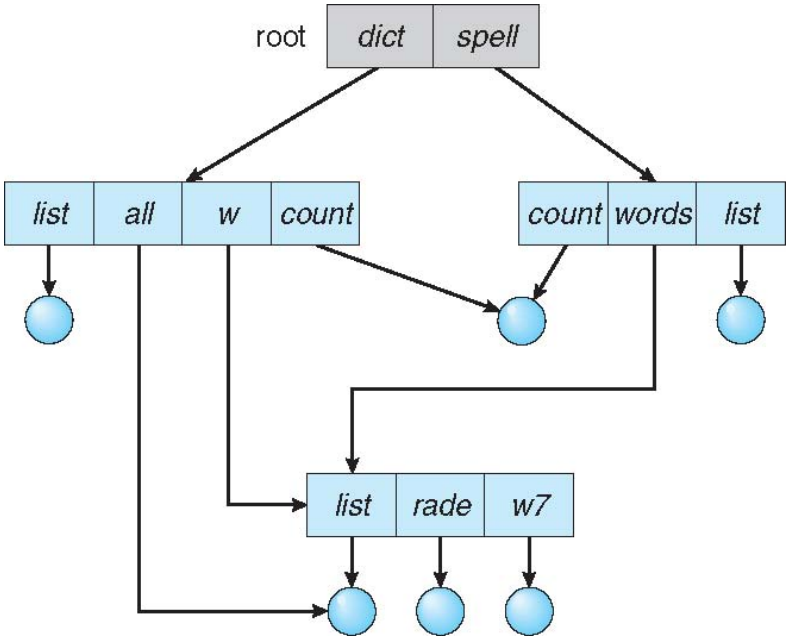
```
mkdir count
```



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

5. Acyclic-Graph Directories

- Have shared subdirectories and files



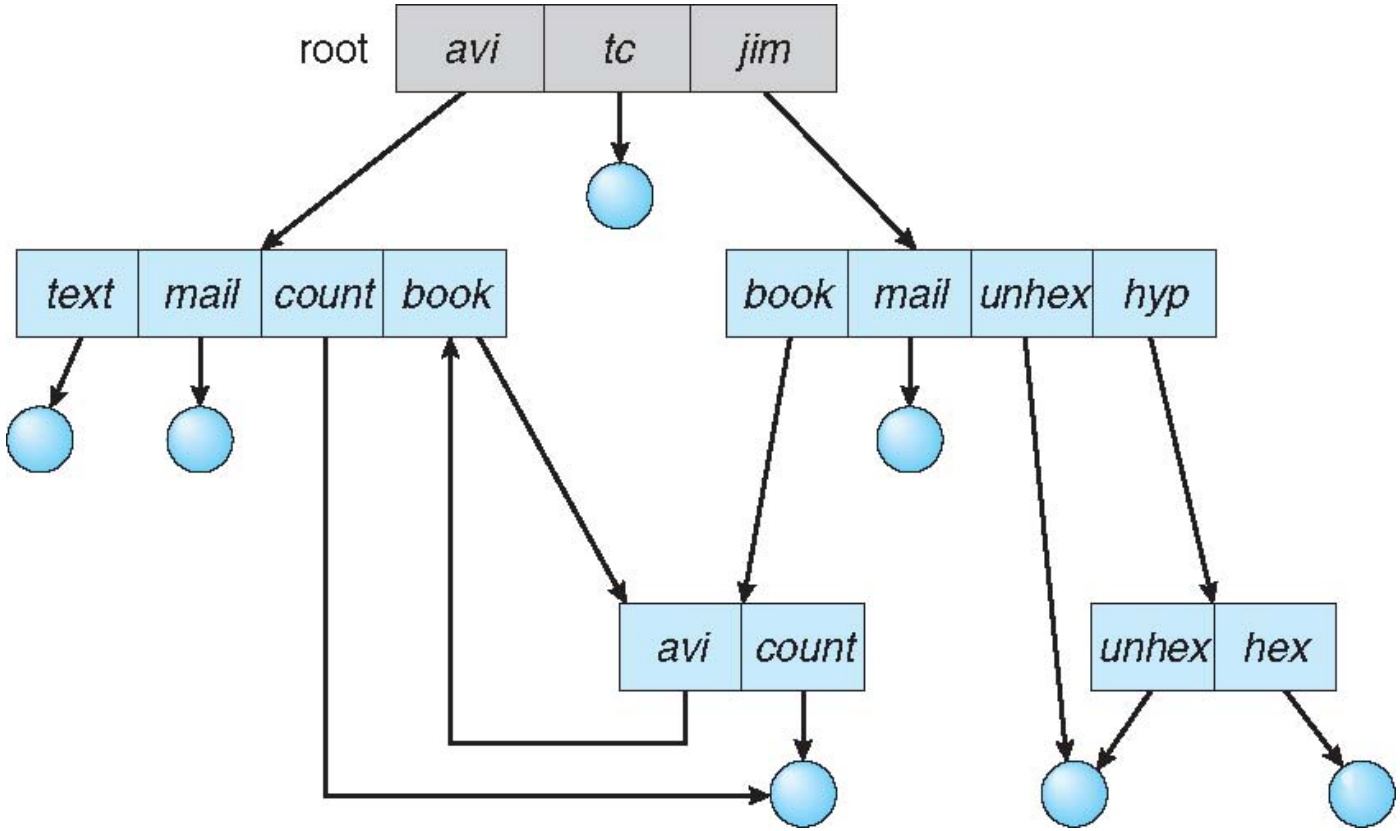
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

5. Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- Deletion might cause dangling pointer
  - e.g., *dict* deletes *count* in previous slide

  Solutions:
  - Keep backpointers of links for each file
  - Leave the link, and delete only when accessed later
  - Keep reference count of each file

6. General Graph Directory



6. General Graph Directory (Cont.)

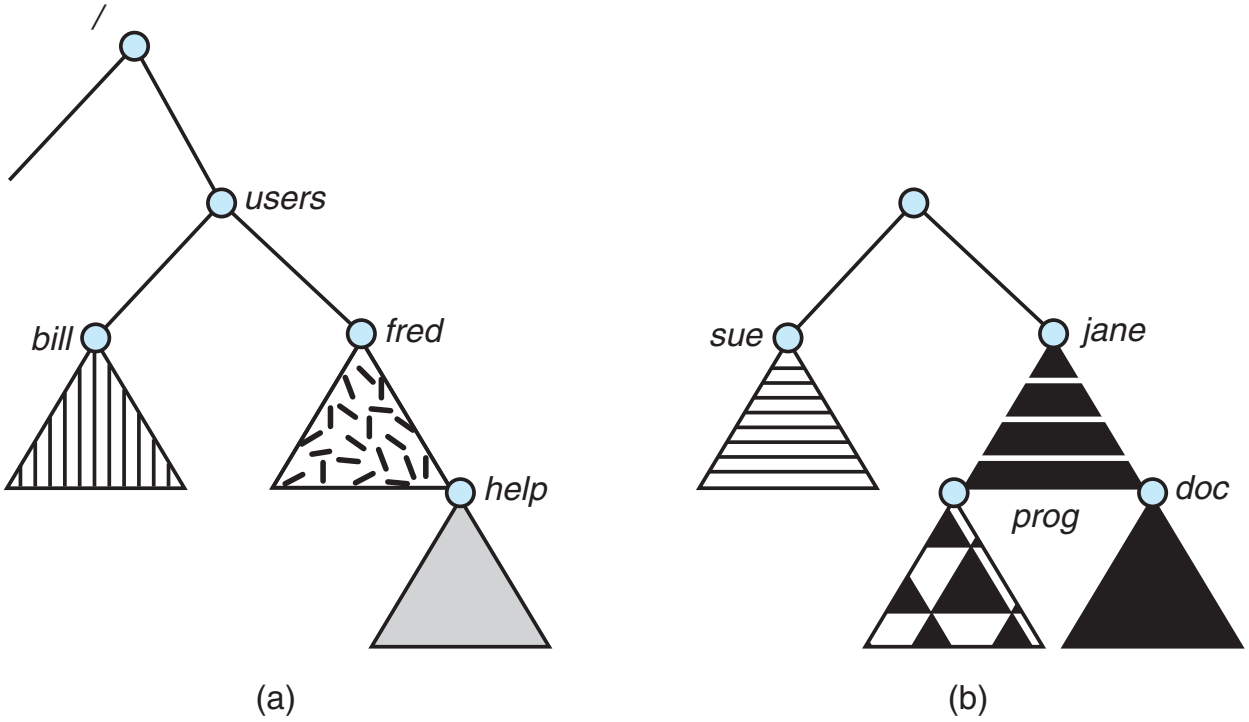
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK
- What is actually done?
  - Bypass links during directory traversal

File-System Interface

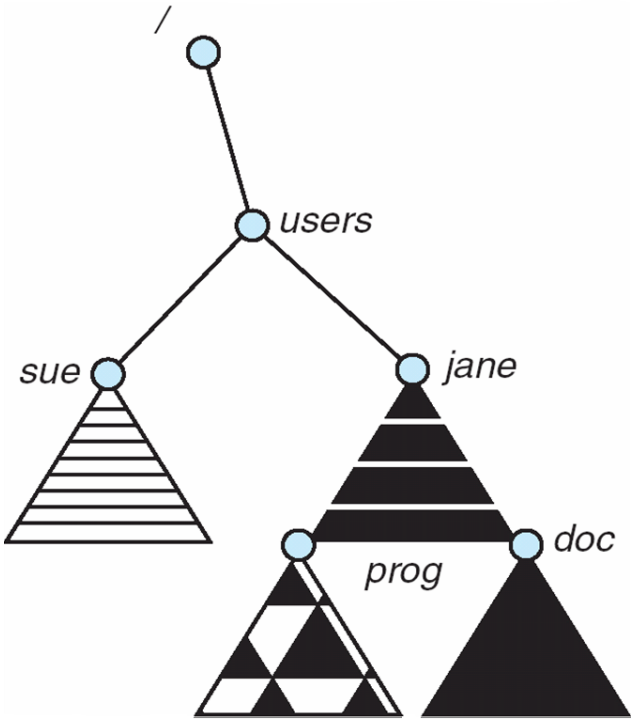
- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- Protection

File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. below) is mounted at a **mount point**



Mount Point



File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- Protection

Protection

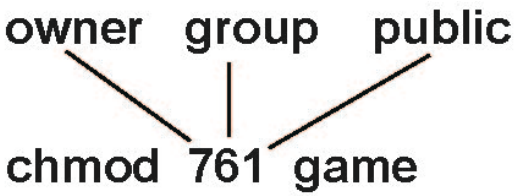
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group 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/

- Question: What is the meaning of chmod 777, chmod 644, chmod 700
- Check : http://www.filepermissions.com/articles/what-are-file-permissions-in-linux-and-mac