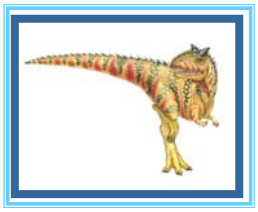
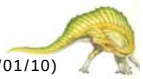
Chapter 10:
File-System Interface



(slides improved by R. Doemer, 06/01/10)

Chapter 10: File-System Interface

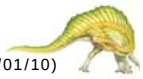
- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection



(slide modified by R. Doemer, 06/01/10)

Objectives

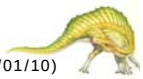
- To explain the function of file systems
- To describe the interface to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection



(slide modified by R. Doemer, 06/01/10)

File Concept

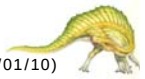
- File
  - Logical storage unit (on secondary storage)
  - Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
    - binary
    - script



(slide modified by R. Doemer, 06/01/10)

File Structure

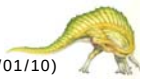
- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
 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



(slide modified by R. Doemer, 06/01/10)

File Attributes

- **File Attributes** are kept in a **directory structure**, which is maintained on the disk
  - **Name** – the 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 (r), writing (w), executing (x)
  - **Time, date, and user identification** – data for protection, security, and usage monitoring



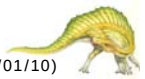
(slide modified by R. Doemer, 06/01/10)

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 Operations

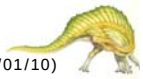
- **Operations** on a file (as an **abstract data type)**
  - **Create** **(w)**
  - **Write** **(w)**
  - **Read** **(r)**
  - **Reposition within file** **(r)**
  - **Delete** **(w)**
  - **Truncate** **(w)**
  - **Execute** **(x)**
- *Open(F_i)* –
  search the directory structure on disk for entry 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



(slide modified by R. Doemer, 06/01/10)

Open Files

- To **manage open files**, several pieces of data are needed :
  - **File pointer**:
    pointer to last read/write location;
    one per process that has the file opened
  - **File-open count**:
    counter of number of times a file is open –
    to allow removal of data from open-file table when last process closes it
  - **Disk location of the file**:
    cache of data access information
  - **Access rights**:
    per-process access mode information (r, w, x)
- **Open File Locking**: mediates access to a file
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

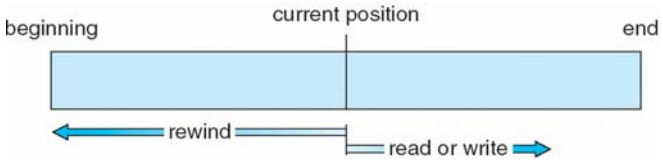


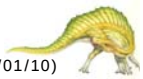
(slide modified by R. Doemer, 06/01/10)

File Access Methods

- **Sequential Access**

  read next
  write next
  reset / rewind
  (rewrite)





(slide modified by R. Doemer, 06/01/10)

File Access Methods

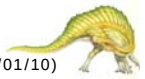
- **Direct Access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  n = relative block number

- Simulation of Sequential Access on Direct-access File

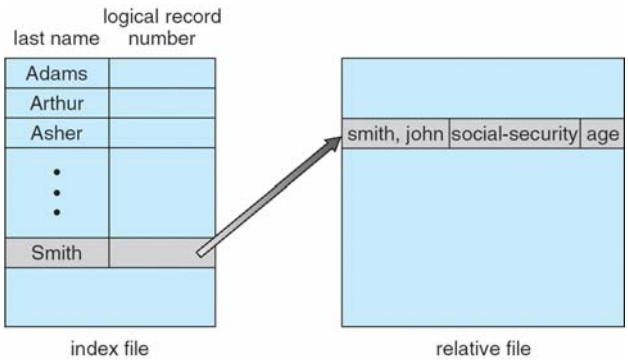
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

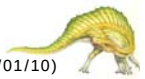


(slide modified by R. Doemer, 06/01/10)

File Access Methods

- Other Access Methods
  - Typically implemented on top of direct-access
  - Example of **Index and Relative Files**

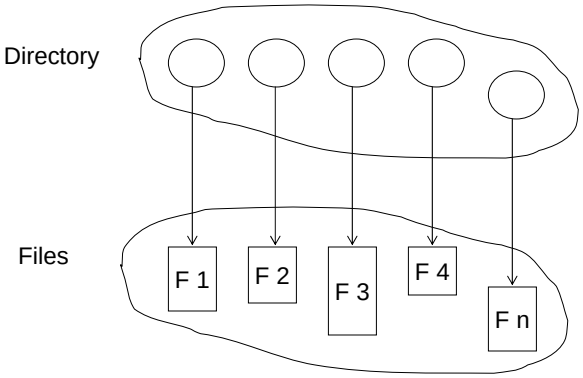




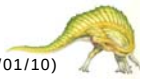
(slide modified by R. Doemer, 06/01/10)

Directory Structure

- **Directory** (aka. **Folder**)
  collection of nodes containing information about a set of files

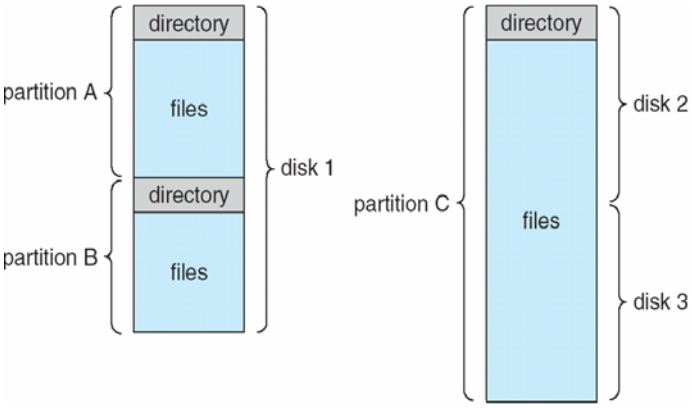


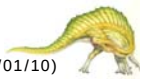
Both the directory structure and the files reside on disk.



(slide modified by R. Doemer, 06/01/10)

A Typical File-System Organization

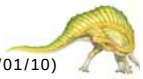




(slide modified by R. Doemer, 06/01/10)

Operations Performed on a Directory

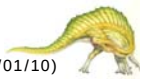
- List a directory (r)
- Search for a file (x)
- Create a file (w)
- Delete a file (w)
- Rename a file (w)
- Traverse the file system (x)



(slide modified by R. Doemer, 06/01/10)

Directory Structure

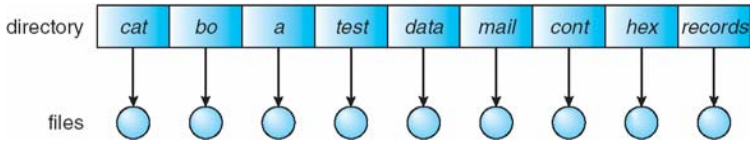
- Organize the directory logically to obtain:
  - **Efficiency** – locating a file quickly
  - **Naming** – convenient to users
    - Two users can have same name for different files
    - The same file can have several different names
  - **Grouping** – logical grouping of files by properties
    - e.g., all Java programs, all games, …



(slide modified by R. Doemer, 06/01/10)

Single-Level Directory

- A single directory for all users



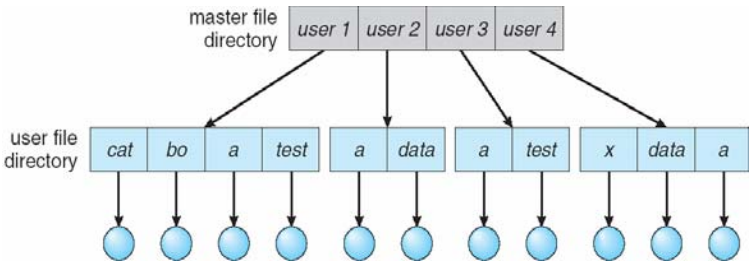
Naming problem

Grouping problem

(slide modified by R. Doemer, 06/01/10)

Two-Level Directory

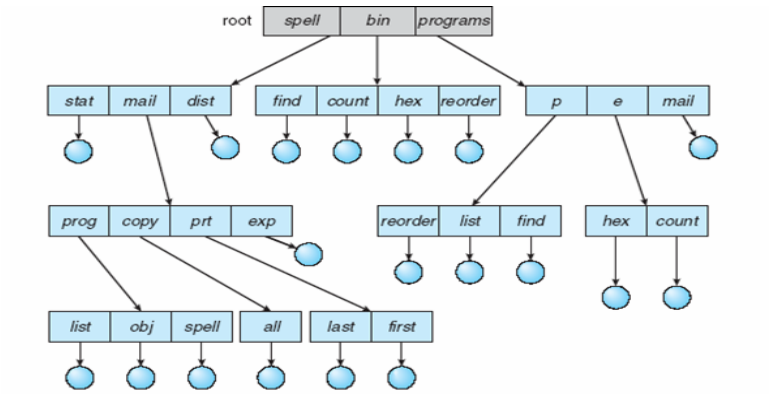
- Separate directory for each user



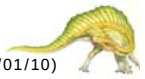
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

(slide modified by R. Doemer, 06/01/10)

Tree-Structured Directories



- Efficient searching
- Grouping Capability
- **Current directory** (working directory)
  - cd /spell/mail/prog
  - type list



(slide modified by R. Doemer, 06/01/10)

Tree-Structured Directories

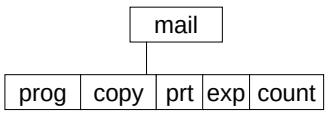
- **Absolute** or **relative path** name
- Creating a new file is done in current directory
- Delete (remove) a file

  `rm filename`
- Creating a new subdirectory is done in current directory

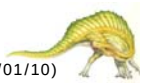
  `mkdir dirname`

  Example: if in current directory `/mail`

  `mkdir count`



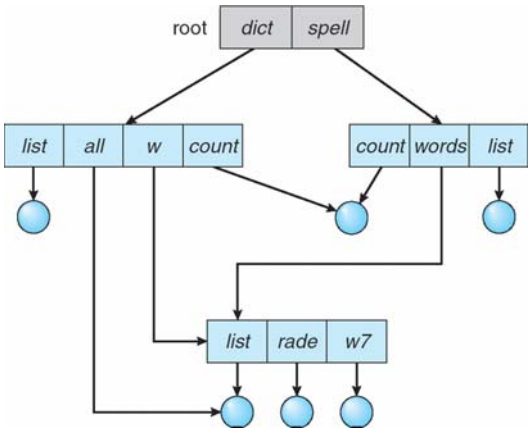
Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

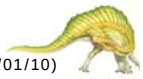


(slide modified by R. Doemer, 06/01/10)

Acyclic-Graph Directories

- Allows to have shared sub-directories and files



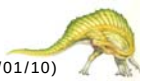


(slide modified by R. Doemer, 06/01/10)

Acyclic-Graph Directories

- Same file can have two different (path) names (aliasing)
  - **Link** – another name (pointer) to an existing file
    - e.g. Unix *hard link*: `ln filename linkname`
  - **Resolve the link** – follow pointer to locate the file
    - e.g. Unix *soft link*: `ln -s dirname linkname`
  - **Avoid cycles** – disallow hard links to directories (Unix)
- Deleting a file = removing the link to the file (*unlink* in Unix)
- When deleting files, *dangling pointers* must be avoided

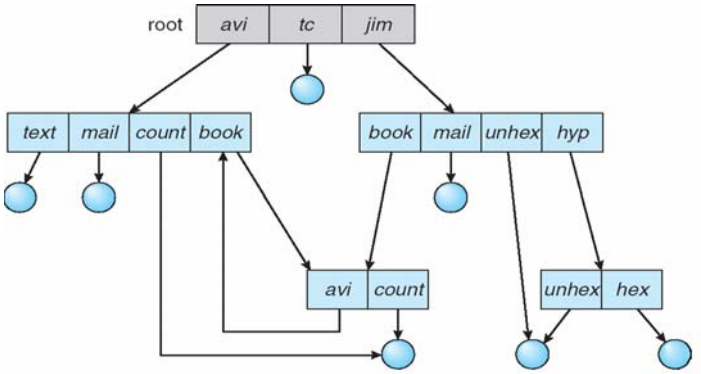
  Possible solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution (Unix)



(slide modified by R. Doemer, 06/01/10)

General Graph Directory

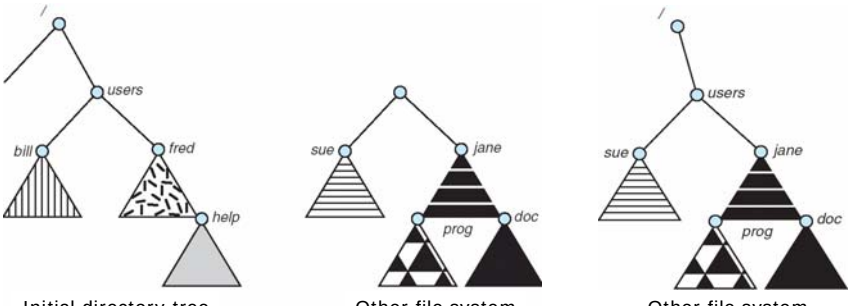
- Allows arbitrary directory and file structures
  - Difficult to traverse and maintain due to cycles…



(slide modified by R. Doemer, 06/01/10)

File System Mounting

- A file system must be **mounted** before it can be accessed
- An unmounted file system is mounted at a **mount point**
- In Unix/Linux, any **directory** can serve as a mount point
  - Unless empty, previous contents are not accessable after mounting

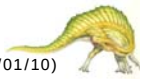


Initial directory tree | Other file system | Other file system mounted at **/users**

(slide modified by R. Doemer, 06/01/10)

File Sharing

- **Sharing** of files on multi-user systems is desirable
  - File **owner** (creator) should be able to control
    - what can be done
    - by whom
- Sharing is typically controlled through a **protection** scheme
  - **Permissions** (r, w, x) per user, and/or per group
  - Multiple users: User IDs identify users
  - Multiple groups: Group IDs allow users to be in groups
- On distributed systems, files may be shared across a network
  - **Network File System** (**NFS**) is a common distributed file-sharing method



(slide modified by R. Doemer, 06/01/10)

File Protection in Unix

- 3 modes of access: read (r), write (w), execute (x)
- 3 classes of users

			r	w	x
a) **owner access**	7	⇒	1	1	1
b) **group access**	5	⇒	1	0	1
c) **public access**	0	⇒	0	0	0

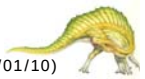
- To share a file (or subdirectory)
  - Ask system administrator to create a group with a unique name, and add appropriate users to the group
  - For the particular file (or subdirectory), define appropriate access

owner group public

```
chmod 750 hangman
```

  - Change the group of the file

```
chgrp games hangman
```



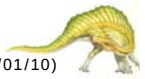
(slide modified by R. Doemer, 06/01/10)

A Sample UNIX Directory Listing

```
% ls -l
```

-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/

type access #links owner group size date name



(slide modified by R. Doemer, 06/01/10)

End of Chapter 10