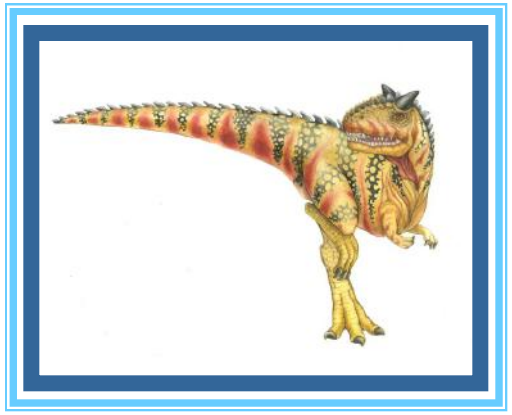
Chapter 13: File-System Interface



Outline

- File Concept
- Access Methods
- Disk and Directory Structure
- Protection
- Memory-Mapped Files

Objectives

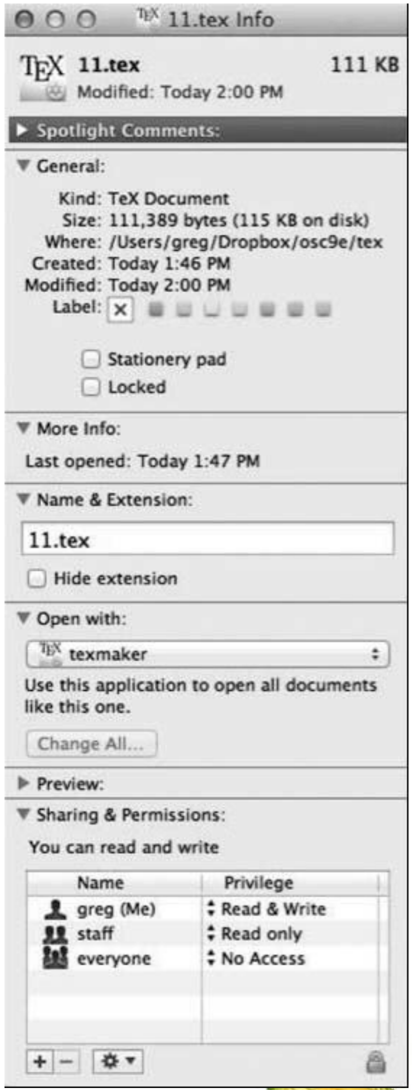
- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

File Concept

- Contiguous logical address space
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents defined by file's creator
  - Many types
    - **text file,**
    - **source file,**
    - **executable file**

File info Window on Mac OS X

- **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
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure



11.tex Info

11.tex 111 KB
Modified: Today 2:00 PM

▶ Spotlight Comments:

▼ General:

Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label:

☐ Stationery pad
☐ Locked

▼ More Info:

Last opened: Today 1:47 PM

▼ Name & Extension:

11.tex

☐ Hide extension

▼ Open with:

texmaker

Use this application to open all documents like this one.

Change All...

▶ Preview:

▼ Sharing & Permissions:

You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

File Operations

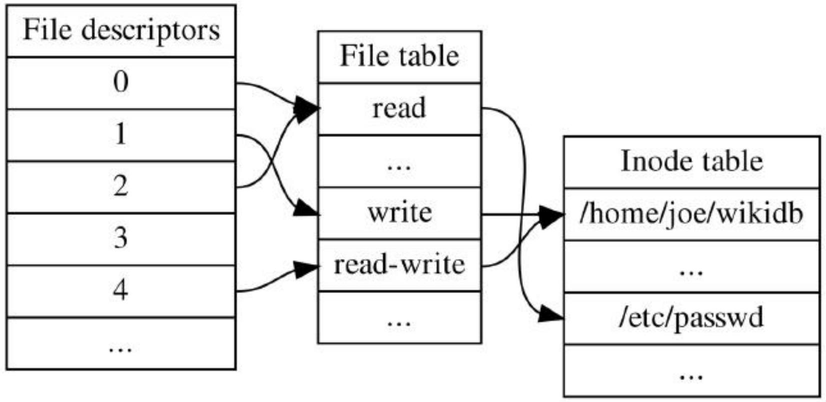
- **Create –** The operating system needs to: 1- find space in the file system to allocate the file; 2 –add an entry for the file in a directory
- **Open –** Returns a file handler that is used by other operation.
- **Write –** The OS keeps a **write pointer** with the location in the file where to perform the next writing operation in a sequential writing. This pointer is updated after every writing operation.
- **Read –** Like with **Write**, the OS keeps **read pointer** with the block to be read in a sequential operation, and it is updated after each reading.
- **Reposition within file – seek** change the **current-file-position pointer** to the position indicated as the input to the call.
- **Delete –** The OS locates the file in the file system, releasing the file space from the disk, and erase it from the directory entry. When using hard links for the same file, the file is only deleted after the last hard link is deleted.
- **Truncate –** Erase the contents of a file keeping the attributes.
- **Close –** removes from memory the information related to the file handler passed as an argument.

Additional File Operations

- **Append**
- **Rename**
- **Get File Attributes**
- **Set File Attributes**
- **Combine operations**
  - **Copy:** Create a new file, open the file to copy, read from the original file and write in the new file.

Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open. It is unique for each process operating the file
  - **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



File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
            RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
            // get the channel for the file
            FileChannel ch = raf.getChannel();
            // this locks the first half of the file - exclusive
            exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
            /** Now modify the data . . . */
            // release the lock
            exclusiveLock.release();
```

File Locking Example – Java API (Cont.)

```
                        // this locks the second half of the file - shared
                        sharedLock = ch.lock(raf.length()/2+1, raf.length(), SHARED);
                        /** Now read the data . . . */
                        // release the lock
                        sharedLock.release();
                } catch (java.io.IOException ioe) {
                        System.err.println(ioe);
                }finally {
                        if (exclusiveLock != null)
                        exclusiveLock.release();
                        if (sharedLock != null)
                        sharedLock.release();
                }
        }
}
```

File Types – Name, Extension

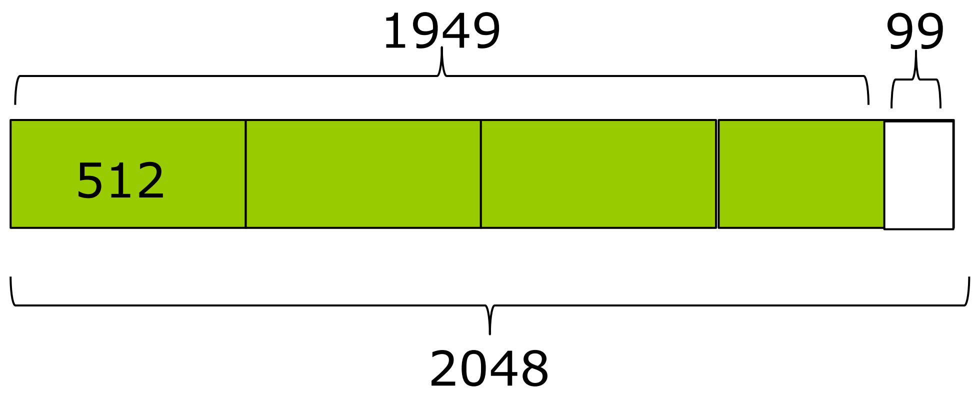
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual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

Internal File Structure

- Files packs logical records into physical blocks of certain size: 256B, 512B,1KB.
- Unix defines files as stream of bytes, where each byte is addressable by the offset from the beginning or end of the file, and has a logical record of 1 byte.
- The file system automatically pack and unpack logical records into a block.
- File operations work with blocks: when performing an operation over a file, the disk doesn't return a byte, but a block.
- This structure generates fragmentation. A file of 1949bytes needs 4 blocks when the block size is 512 bytes, wasing 99 blocks



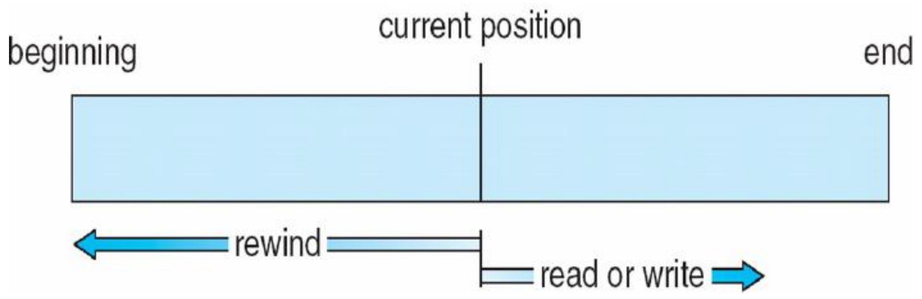
Access Methods

- A file is fixed length **logical records**
- **Sequential Access**
- **Direct Access**
- **Other Access Methods**

Sequential Access

- Operations
  - `read next`
  - `write next`
  - `Reset`
  - no read after last write (rewrite)

- Figure



Direct Access

- Operations
  - `read` *n*
  - `write` *n*
  - `position to` *n*
    - `read next`
    - `write next`
    - `rewrite` *n*

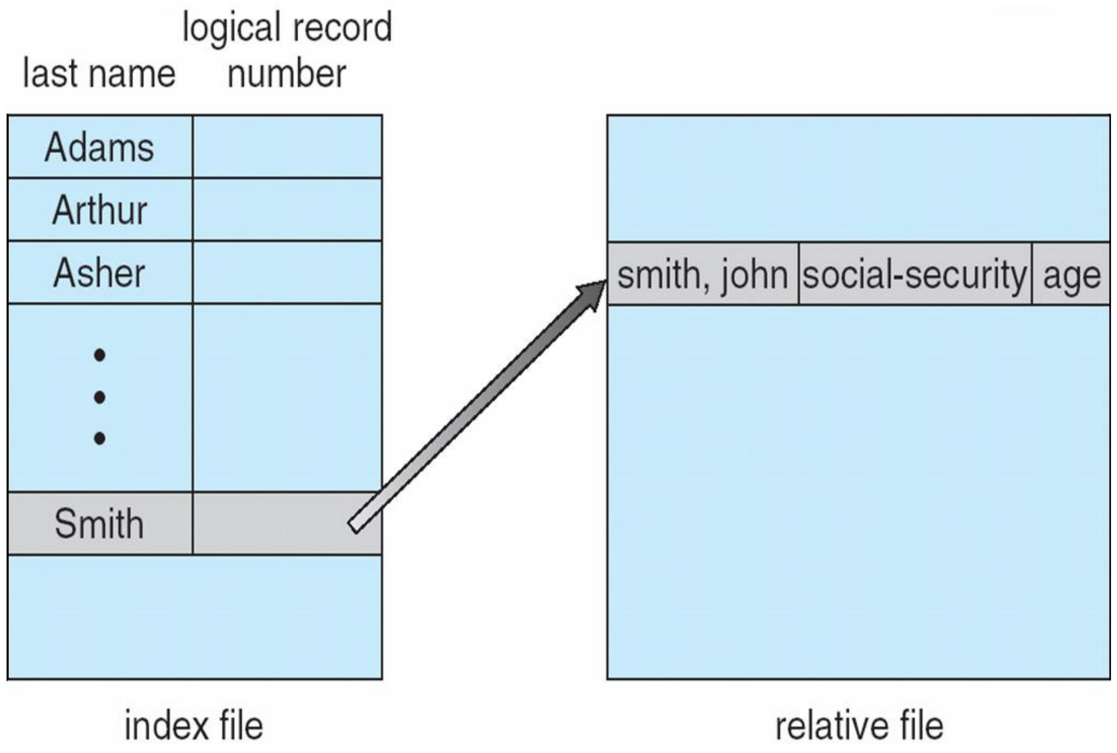
  n = **relative block number**

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

Simulation of Sequential Access on Direct-access File

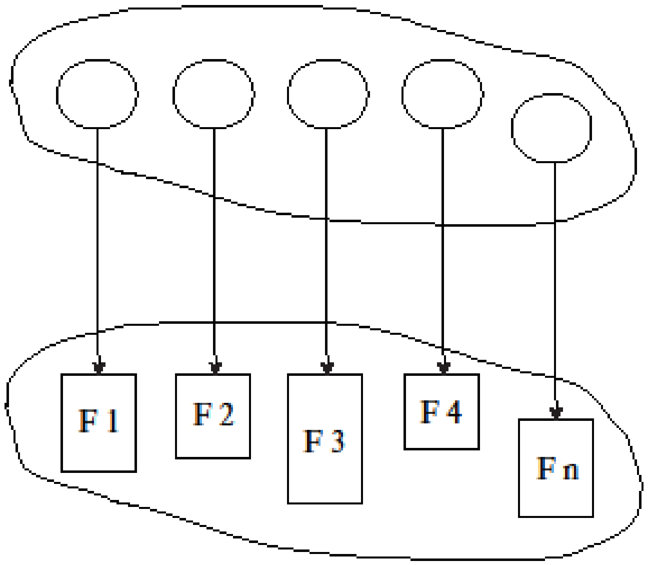
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Example of Index and Relative Files



Directory Structure

- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

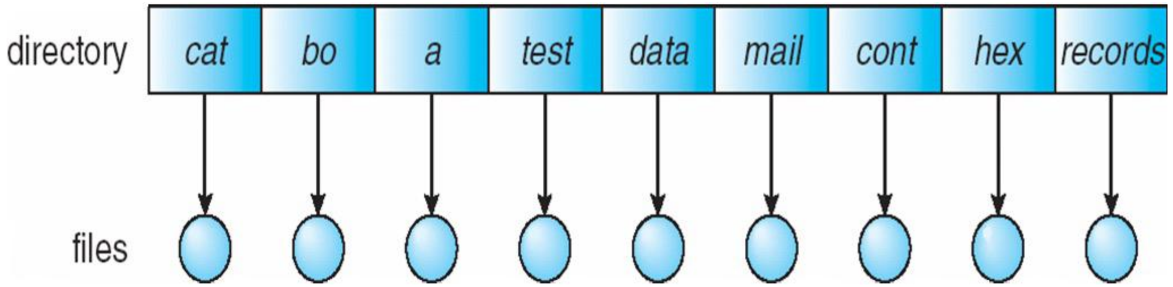
Directory Organization

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

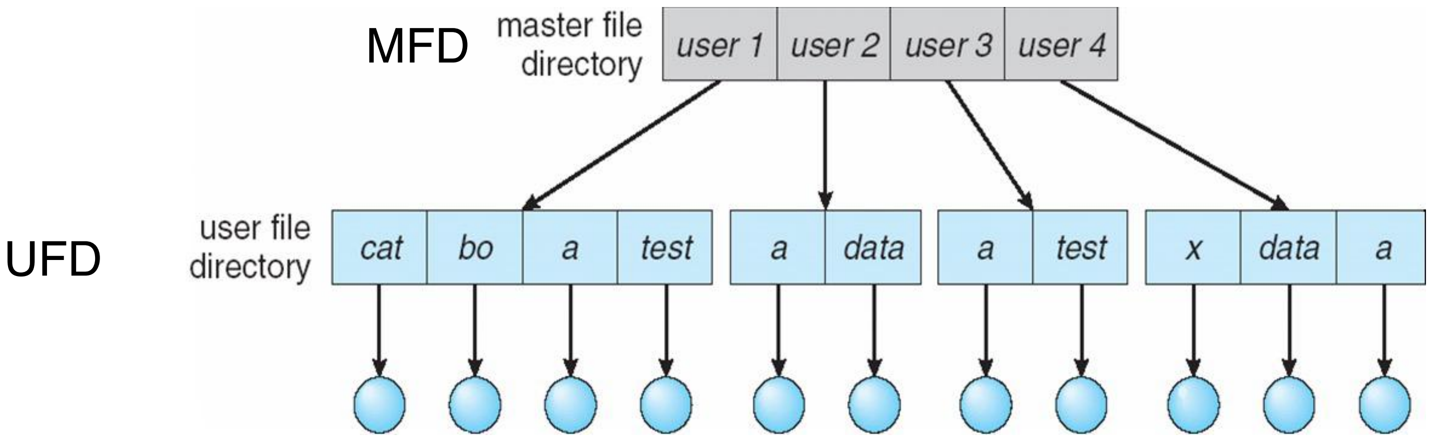
- A single directory for all users



- Naming problem. All the files need to have different names. In a multi-user environment, different users need to keep different names for their files.
- Organization problem: Since it is not possible to allocate files in different folders, a single-level directory structure becomes unmanageable.

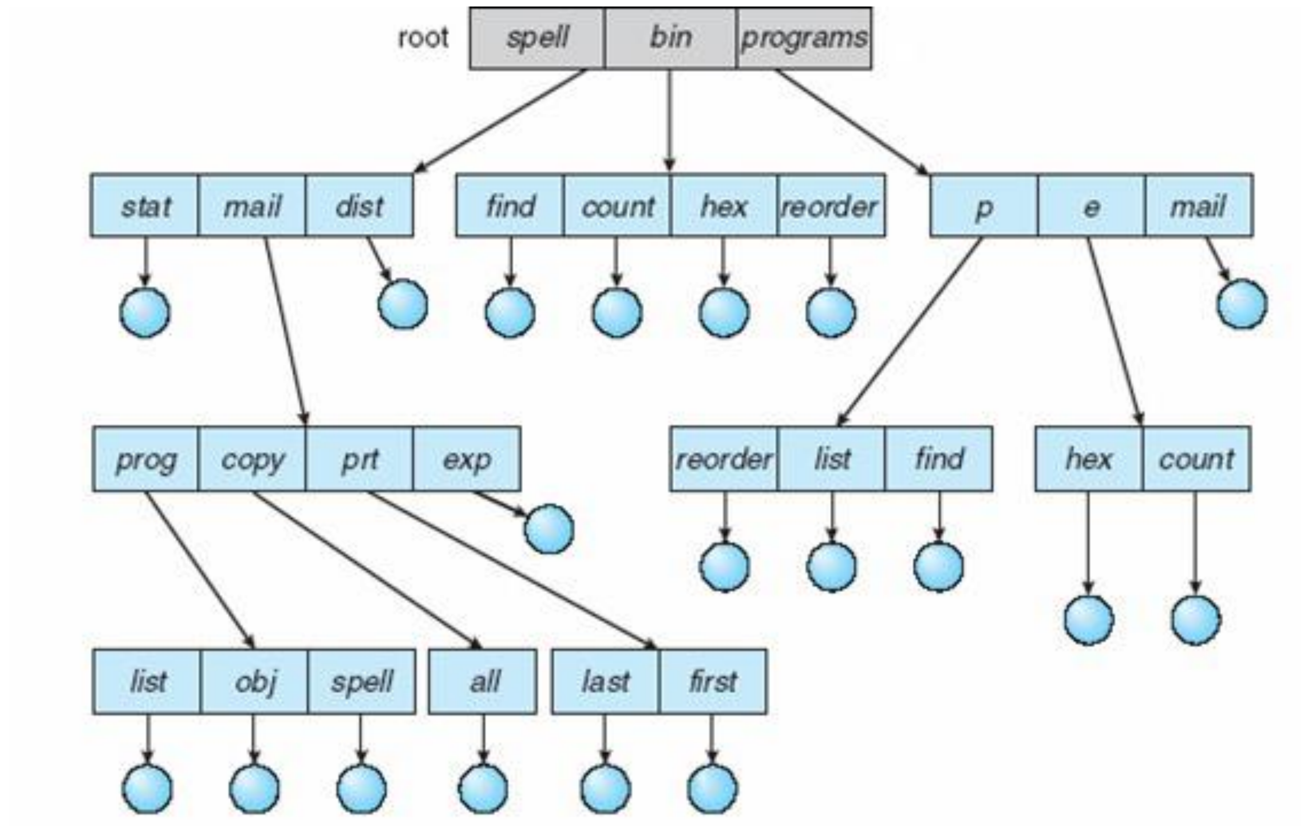
Two-Level Directory

- Separate directory for each user



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

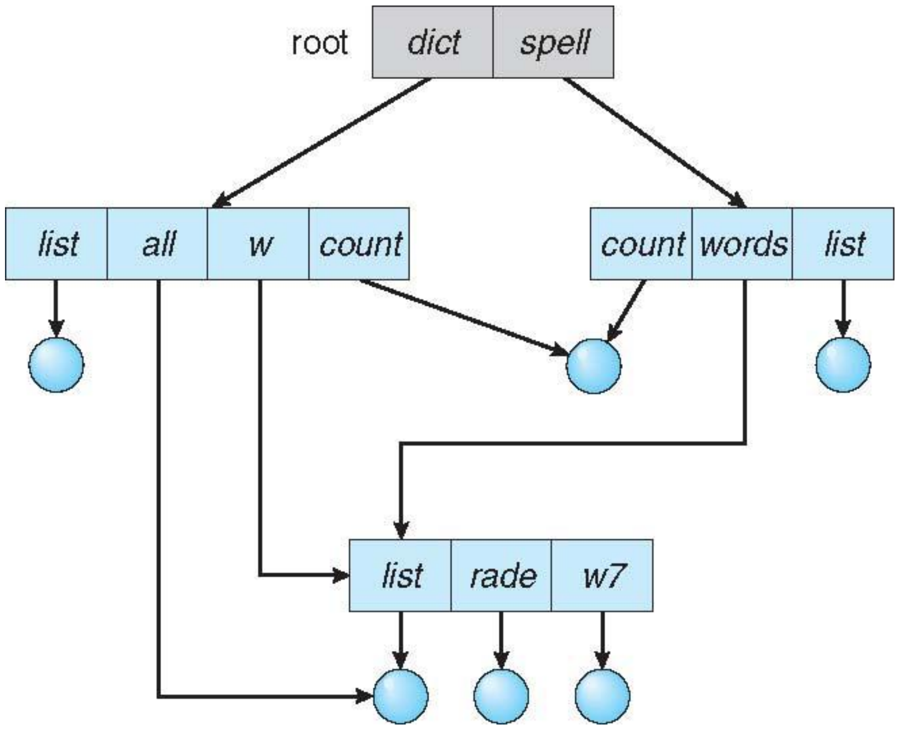
Tree-Structured Directories



Acyclic-Graph Directory

- Two different names (aliasing)
- If ***dict*** deletes **w/*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



Implementation:

New directory entry type

Link – another name (pointer) to an existing file

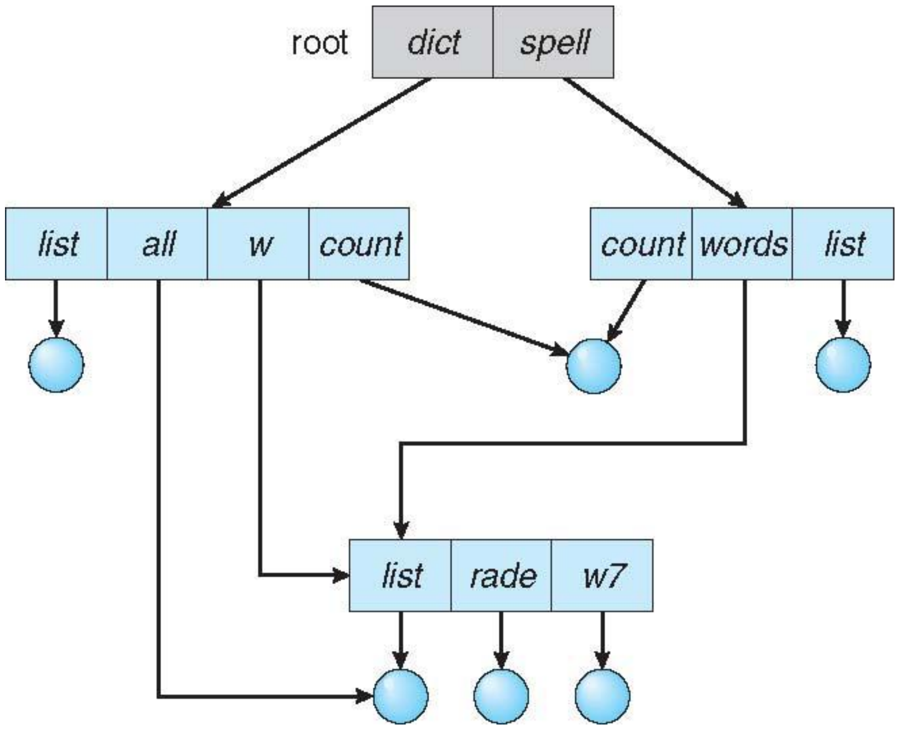
Resolve the link – follow pointer to locate the file

Duplicate the information in the directory where we want to have the shared file. Problem:

Acyclic-Graph Directory

- Two different names (aliasing)
- If ***dict*** deletes **w/*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



Implementation:

New directory entry type

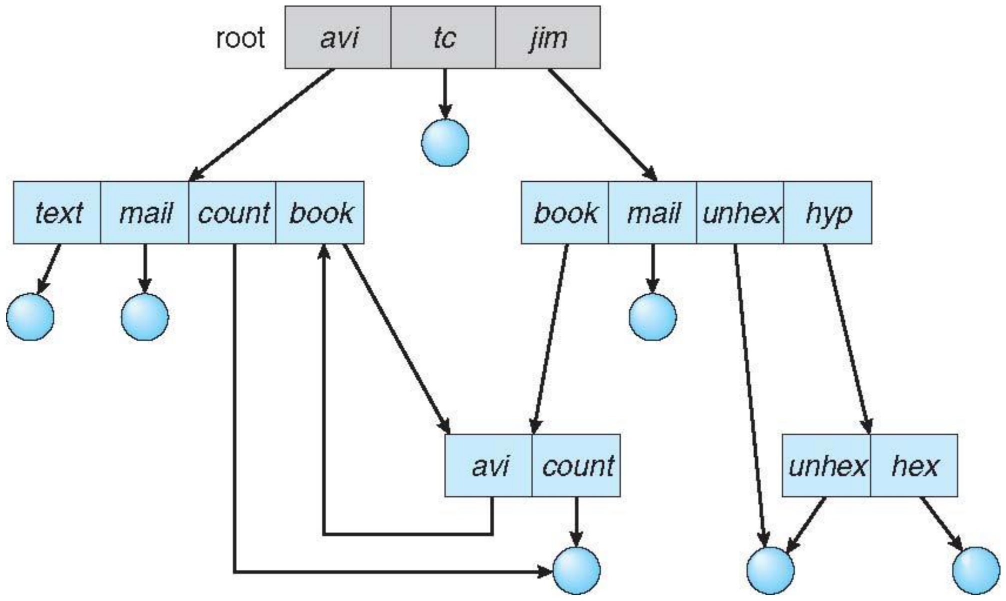
Link – another name (pointer) to an existing file

Resolve the link – follow pointer to locate the file

Duplicate the information in the directory where we want to have the shared file. Problem: Maintaining the consistency is expensive.

General Graph Directory (Cont.)

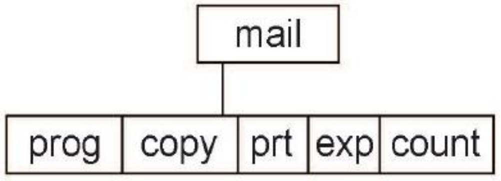
- Searching is a problem: potentially infinite traversing. Solution: limit the number of traversed directories.
- When to delete a file? Self cycles may makes that the counting reference for a file is never zero. Solution: **Garbage collection**
- How do we guarantee no cycles?
  - Allow only links to files not subdirectories
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK



Current Directory

- Can designate one of the directories as the current (working) directory
  - `cd /spell/mail/prog`
  - `type list`
- Creating and deleting a file is done in current directory
- Example of creating a new file
  - If in current directory is `/mail`
  - The command

    mkdir <dir-name>
  - Results in:



  - Deleting "mail" $\Rightarrow$ deleting the entire subtree rooted by "mail"

Protection

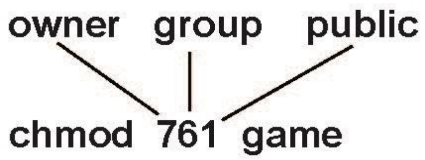
- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups in Unix

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

			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 file (say *game*) or subdirectory, define an appropriate access.



- Attach a group to a file

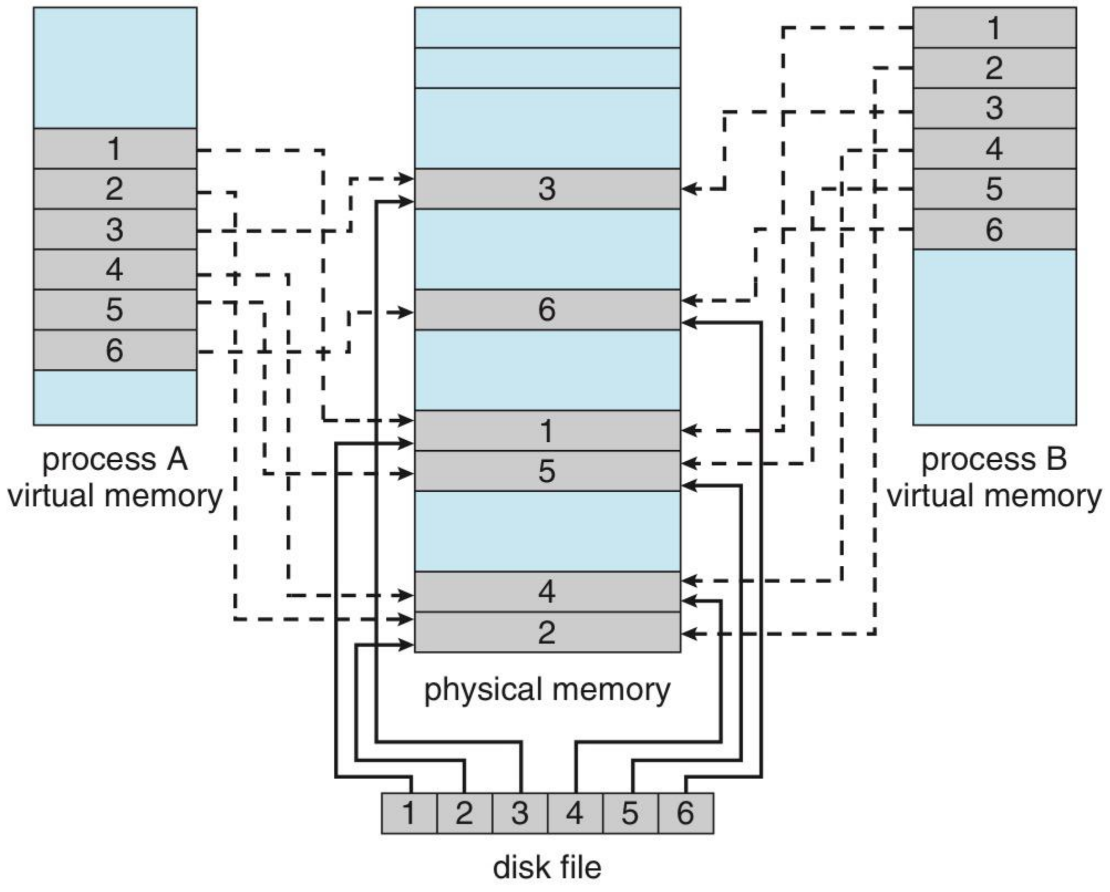
```
chgrp      G      game
```

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/
```

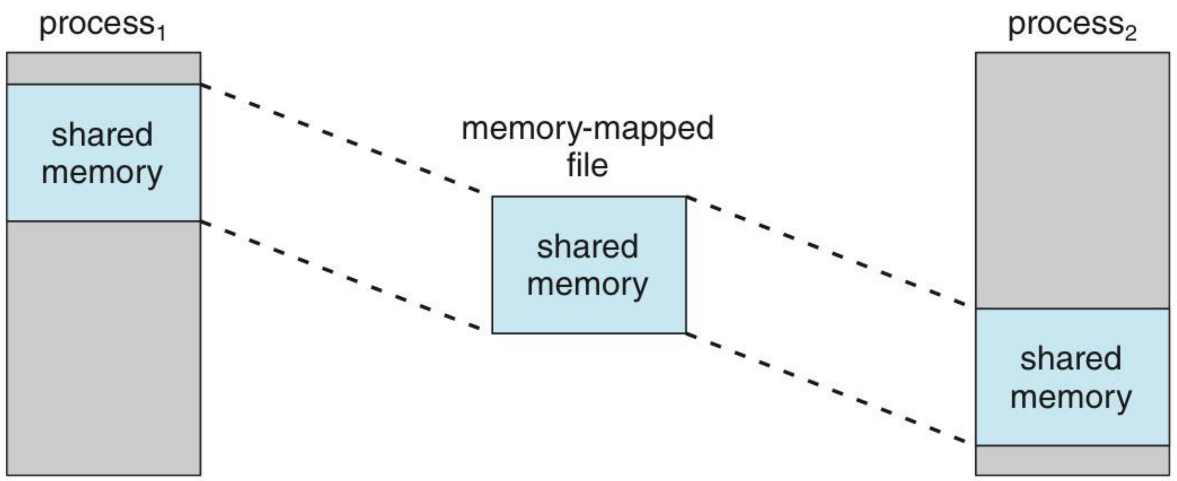
Memory Mapping

- It uses Virtual Memory to map disk blocks to a page or pages in memory.
- It allows to manipulate information on the disk without using read or write, but simply doing the memory mapping.
- Disk information and memory information in the mapping are not synchronized all the time.



Memory Mapping

- It uses Virtual Memory to map disk blocks to a page or pages in memory.
- It allows to manipulate information on the disk without using read or write, but simply doing the memory mapping.
- Disk information and memory information in the mapping are not synchronized all the time.



End of File System Interface