Operating Systems
[13. File-System Interface]

Chung-Wei Lin
cwlin@csie.ntu.edu.tw
CSIE Department
National Taiwan University

From Operating System Concepts (10th Edition)

Objectives

- Explain the function of file systems
- Describe the interfaces to file systems
- Discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- Explore file-system protection

Outline

- **File Concept**
  - File Attributes
  - File Operations
  - File Types
  - File Structure
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File Concept

- File
  - The operating system abstracts from the physical properties of its storage devices to define a logical storage unit
- The information in a file is defined by its creator
  - Many different types
  - Examples
    - Text file
    - Source file
    - Executable file

Outline

- **File Concept**
  - **File Attributes**
  - File Operations
  - File Types
  - File Structure
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File Attributes (1/2)

- **Name**
  - Only information kept in human-readable form
- **Identifier**
  - Unique tag (number) identifies file within the file system
- **Type**
  - Needed for systems that support different types of files
- **Location**
  - Pointer to a device and the file location on the device
- **Size**
  - Current file size

File Attributes (2/2)

- **Protection**
  - Access-control information
- **Timestamps and user identification**
  - Useful for protection, security, and usage monitoring
- **Extended file attributes** (some new file systems)
  - Character encoding of the file and security features such as a file checksum
- The information about all files is kept in the directory structure
  - Typically, a directory entry consists of the file's name and its unique identifier
  - The identifier locates the other file attributes

Outline

- **File Concept**
  - File Attributes
  - **File Operations**
  - File Types
  - File Structure
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File Operations

- A file is an **abstract data type**
- **Create**
- **Open**
  - All operations except create and delete require a file `open()` first
- **Write**: at **write pointer** location
- **Read**: at **read pointer** location
- **Reposition** (**seek**)
- **Delete**
- **Truncate**
- Close
  - Move the content in memory to directory structure on disk

Open Files

- **Open-file table**
  - Contain information about all open files
- Several pieces of information are associated with an open file
  - File pointer
    - The last read-write location as a **current-file-position pointer**, unique to each process operating on the file
  - File-open count
    - The number of opens and closes, reaching zero on the last close
  - Location of the file
    - Wherever the file is located, on mass storage, on a file server across the network, or on a RAM drive
  - Access rights
    - Stored on the per-process table so the operating system can allow or deny subsequent I/O requests

Locking Open Files

- Provided by some operating systems
  - Similar to reader-writer locks (Chapter 7)
  - A **shared lock** is similar to reader lock
    - Several processes can acquire concurrently
  - An **exclusive lock** is similar to writer lock
    - Only one process at a time can acquire such a lock
  - Some systems provide only exclusive file locking
- **Mandatory** file-locking mechanism
  - Once a process acquires an exclusive lock, the operating system will prevent any other process from accessing the locked file
- **Advisory** file-locking mechanism
  - It is up to software developers to ensure that locks are appropriately acquired and released

File-Locking Example in Java

```
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();
                  // 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();
            }
      }
}
```

Outline

- **File Concept**
  - File Attributes
  - File Operations
  - **File Types**
  - File Structure
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File Types

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	mp3, mp4, mov, rm, avi	Binary file containing audio or A/V information

Outline

- **File Concept**
  - File Attributes
  - File Operations
  - File Types
  - **File Structure**
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File Structure

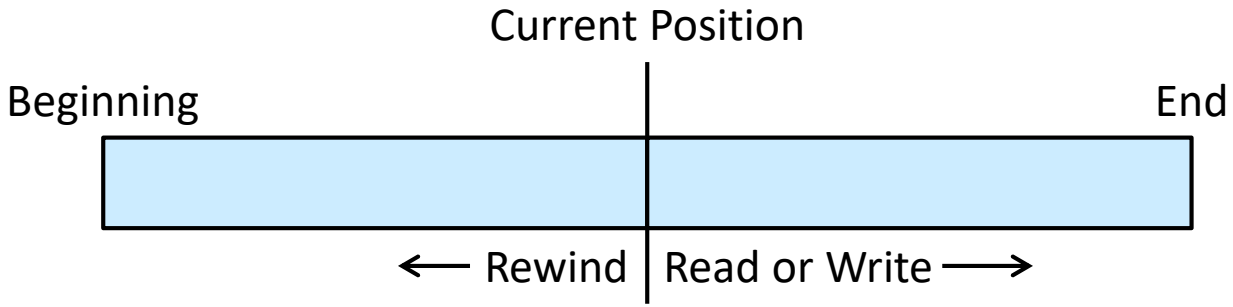
- Examples
  - None
    - Sequence of words, bytes
  - Simple record structure
    - Lines, fixed length, variable length
  - Complex structures
    - Formatted document, relocatable load file
  - Can simulate last two with first method by inserting appropriate control characters
- Who decides
  - Operating system
  - Program

Outline

- File Concept
- **Access Methods**
  - **Sequential Access**
  - Direct Access
  - Other Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

Sequential Access

- Information in the file is processed in order, one record after the other
  - `read_next()`
  - `write_next()`
  - `reset()`
- Examples
  - Editors and compilers usually access files in this fashion



Outline

- File Concept
- **Access Methods**
  - Sequential Access
  - **Direct Access**
  - Other Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

Direct Access (Relative Access) (1/2)

- A file is made up of fixed-length **logical records**
  - Allow programs to read and write records rapidly in no particular order
  - `read(n)`
  - `write(n)`
  - `position_file(n), read_next()`
  - `position_file(n), write_next()`
  - n = **relative block number**
- The use of relative block numbers allows the operating system to decide where the file should be placed
  - **Allocation problem** in Chapter 14

Direct Access (Relative Access) (2/2)

- Simulate sequential access on a direct-access file
  - Keeping a variable **`cp`** that defines our current position
  - **`reset`**
    - `cp = 0;`
  - **`read_next`**
    - `read cp;`
    - `cp = cp + 1;`
  - **`write_next`**
    - `write cp;`
    - `cp = cp + 1;`

Outline

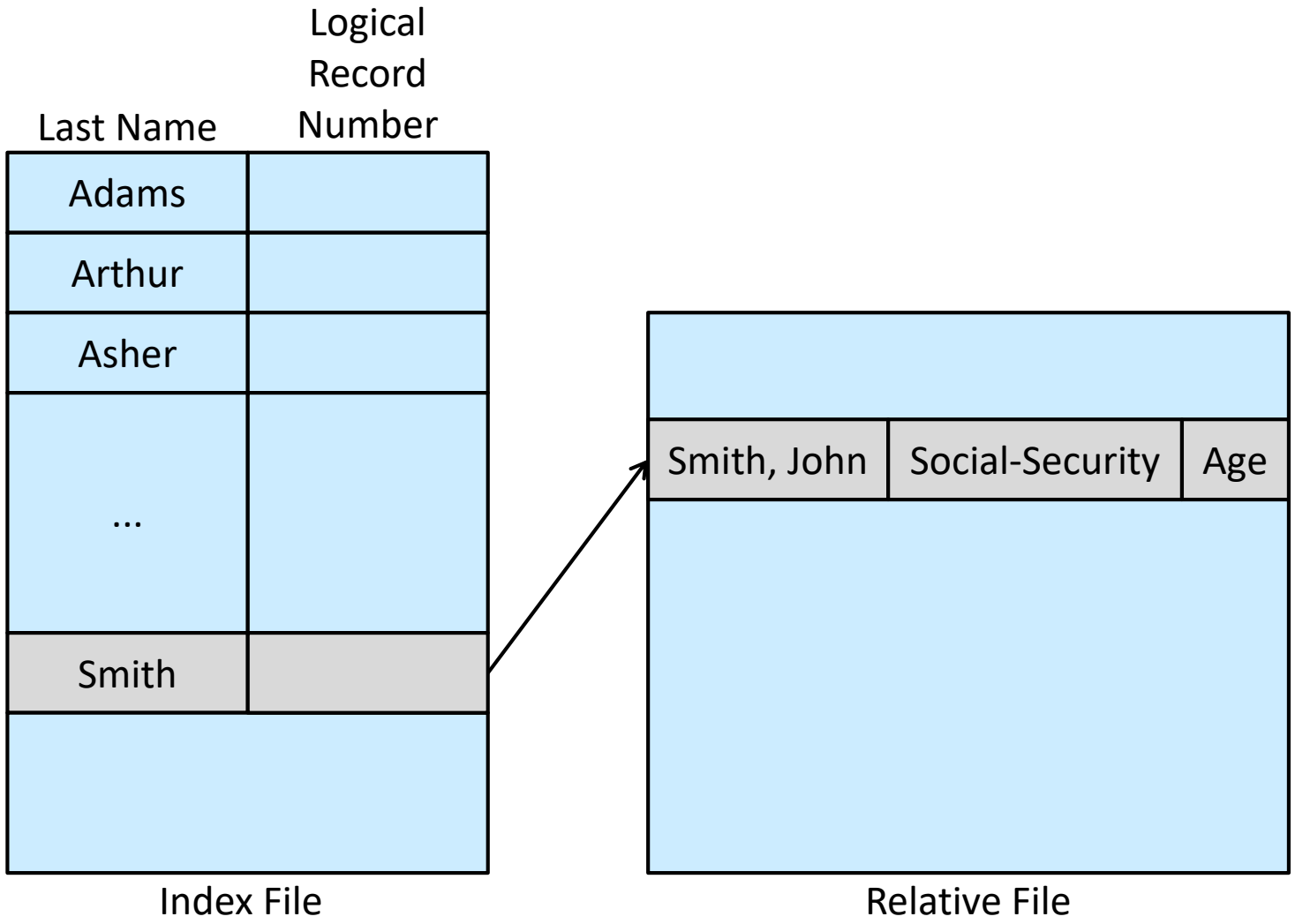
- File Concept
- **Access Methods**
  - Sequential Access
  - Direct Access
  - **Other Access Methods**
- Directory Structure
- Protection
- Memory-Mapped Files

Other Access Methods (1/2)

- Other access methods can be built on top of a direct-access method
- These methods generally involve the construction of an **index** for the file
  - The index contains pointers to the various blocks
- With large files, the index file itself may become too large to be kept in memory
  - Create an index for the index file
    - The primary index file contains pointers to secondary index files, which point to the actual data items
  - IBM indexed sequential-access method (ISAM)

Other Access Methods (2/2)

- OpenVMS index and relative files



Outline

- File Concept
- Access Methods
- **Directory Structure**
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - General Graph Directory
- Protection
- Memory-Mapped Files

Operations Performed on Directory

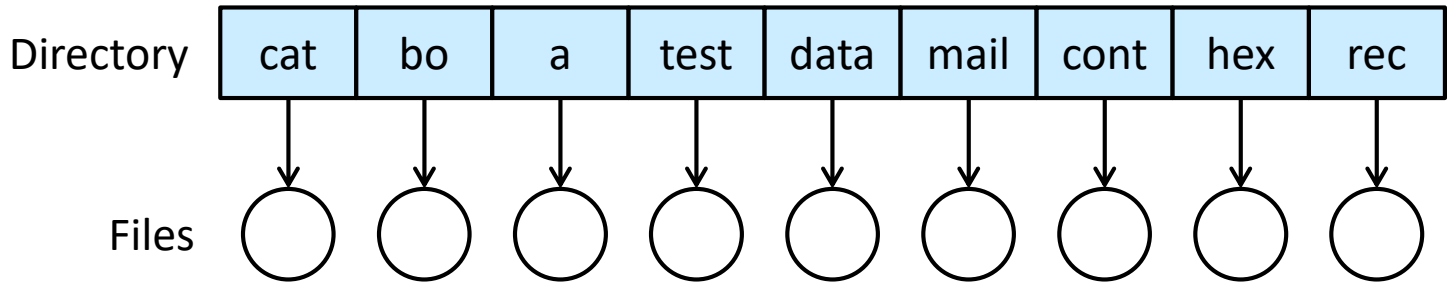
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Outline

- File Concept
- Access Methods
- **Directory Structure**
  - **Single-Level Directory**
  - Two-Level Directory
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - General Graph Directory
- Protection
- Memory-Mapped Files

Single-Level Directory

- All files are contained in the same directory



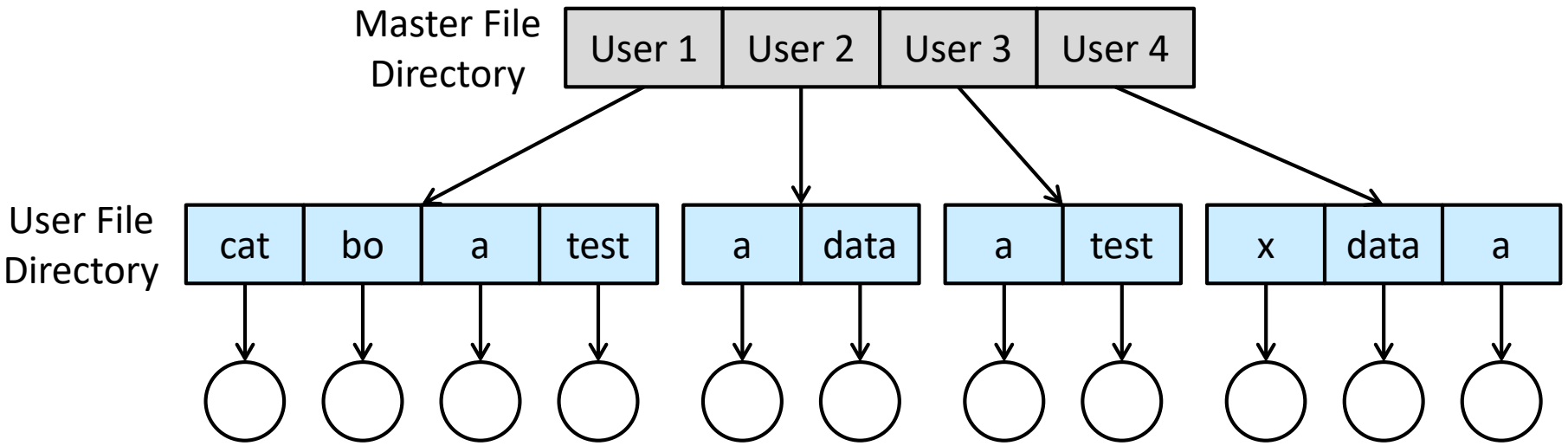
- Naming problem

Outline

- File Concept
- Access Methods
- **Directory Structure**
  - Single-Level Directory
  - **Two-Level Directory**
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - General Graph Directory
- Protection
- Memory-Mapped Files

Two-Level Directory

- A separate directory for each user

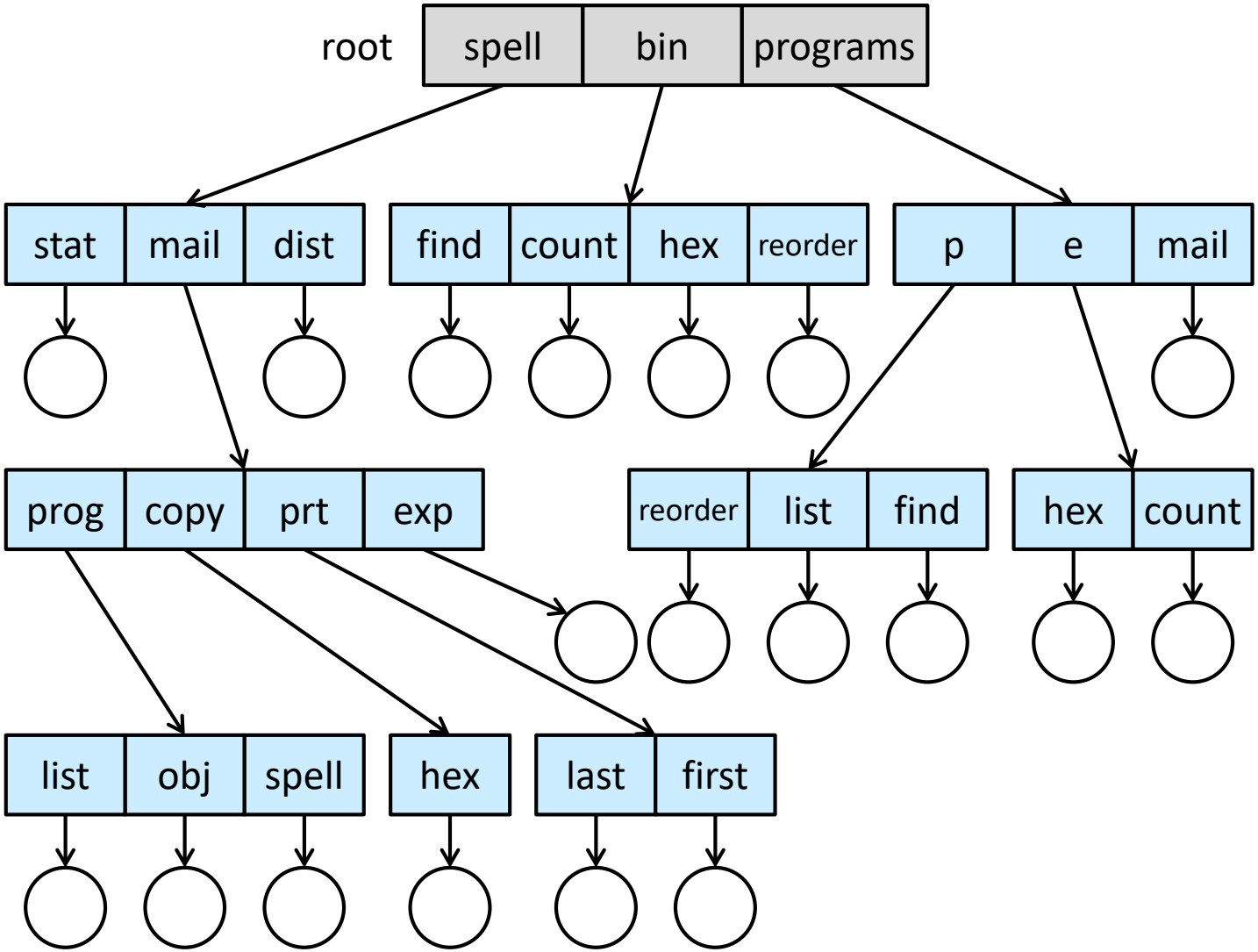


- A user name and a file name define a **path name**
- Search procedure and **search path**
  - The sequence of directories searched when a file is named is called

Outline

- File Concept
- Access Methods
- **Directory Structure**
  - Single-Level Directory
  - Two-Level Directory
  - **Tree-Structured Directories**
  - Acyclic-Graph Directories
  - General Graph Directory
- Protection
- Memory-Mapped Files

Tree-Structured Directories (1/2)



Tree-Structured Directories (2/2)

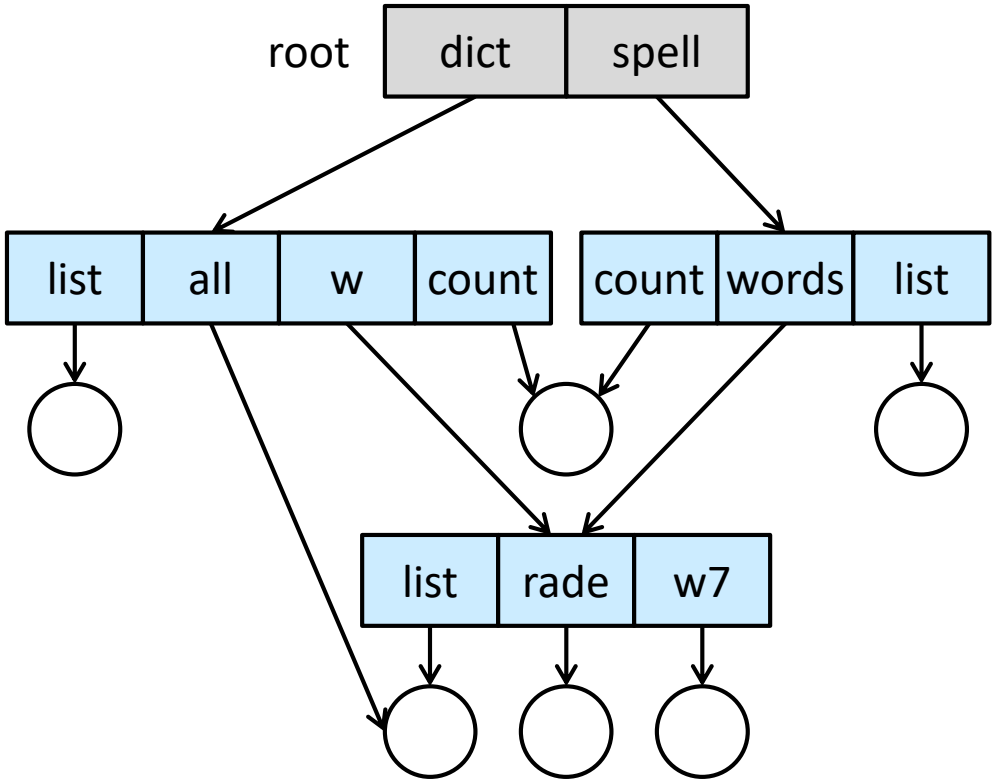
- A directory (or subdirectory) contains a set of files or subdirectories
  - One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1)
- Current directory (working directory)
  - The user can change the current directory
  - Some systems leave it to the application (e.g., a shell) to track and operate on a current directory
- **Absolute path name** or **relative path name**
- How to handle the deletion of a directory?
  - Some systems will not delete a directory unless it is empty
  - Some systems will delete all files and subdirectories

Outline

- File Concept
- Access Methods
- **Directory Structure**
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directories
  - **Acyclic-Graph Directories**
  - General Graph Directory
- Protection
- Memory-Mapped Files

Acyclic-Graph Directories (1/2)

- A subdirectory needs to be shared
- An acyclic graph allows directories to share subdirectories and files



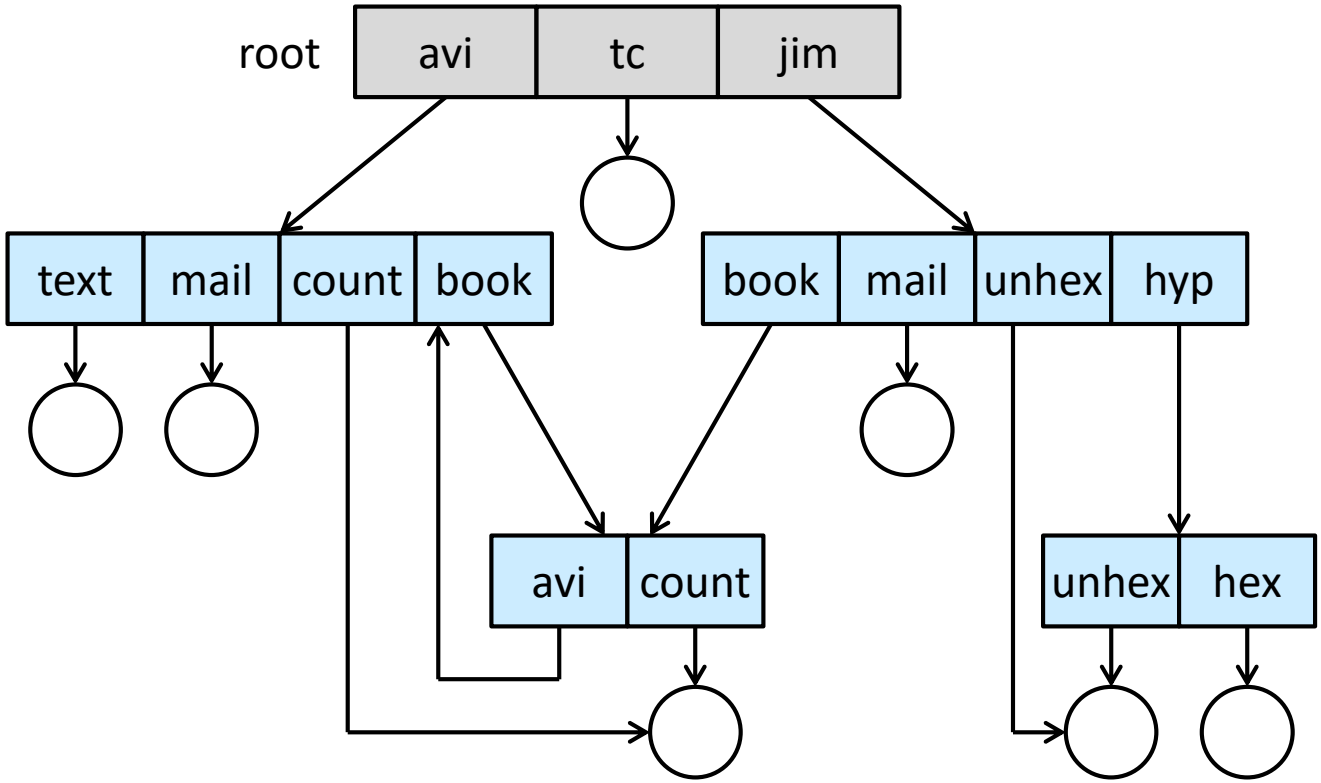
Acyclic-Graph Directories (2/2)

- **Link**
  - A pointer to another file or subdirectory
  - We **resolve** the link by using that path name to locate the real file
- Distinct file names may refer to the same file
  - This situation is similar to the aliasing problem for programming languages
- Removing a file may leave dangling pointers (to the now-nonexistent file)
  - Search for the links and remove them as well
    - This search can be expensive
  - Leave the links until an attempt is made to use them
  - Preserve the file until all references to it are deleted
    - Keep a list of all references to a file

Outline

- File Concept
- Access Methods
- **Directory Structure**
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - **General Graph Directory**
- Protection
- Memory-Mapped Files

General Graph Directory (1/2)



General Graph Directory (2/2)

- A poorly designed algorithm might result in an infinite loop continually searching through the cycle and never terminating
- When cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file
  - A **garbage collection** scheme to determine when the last reference has been deleted and the disk space can be reallocated

Outline

- File Concept
- Access Methods
- Directory Structure
- **Protection**
  - **Types of Access**
  - Access Control
- Memory-Mapped Files

Types of Access

- **Read** from the file
- **Write** or rewrite the file
- **Execute**
  - Load the file into memory and execute it
- **Append**
  - Write new information at the end of the file
- **Delete** the file and free its space for possible reuse
- **List** the name and attributes of the file
- **Attribute change**
  - Change the attributes of the file

Outline

- File Concept
- Access Methods
- Directory Structure
- **Protection**
  - Types of Access
  - **Access Control**
- Memory-Mapped Files

Access Control

- **Access-control list** (ACL) specifies user names and the types of access allowed for each user
  - Constructing such a list may be a tedious and unrewarding task
  - The directory entry, previously of fixed size, now must be of variable size
- Three classifications of users on Unix / Linux
  - Owner, group, other
- Three modes of access
  - Read, write, execute
- Define access for a particular file or subdirectory
  - `chmod 761 name`
    - 7: owner access (RWX = 1 1 1)
    - 6: group access (RWX = 1 1 0)
    - 1: public access (RWX = 0 0 1)

Outline

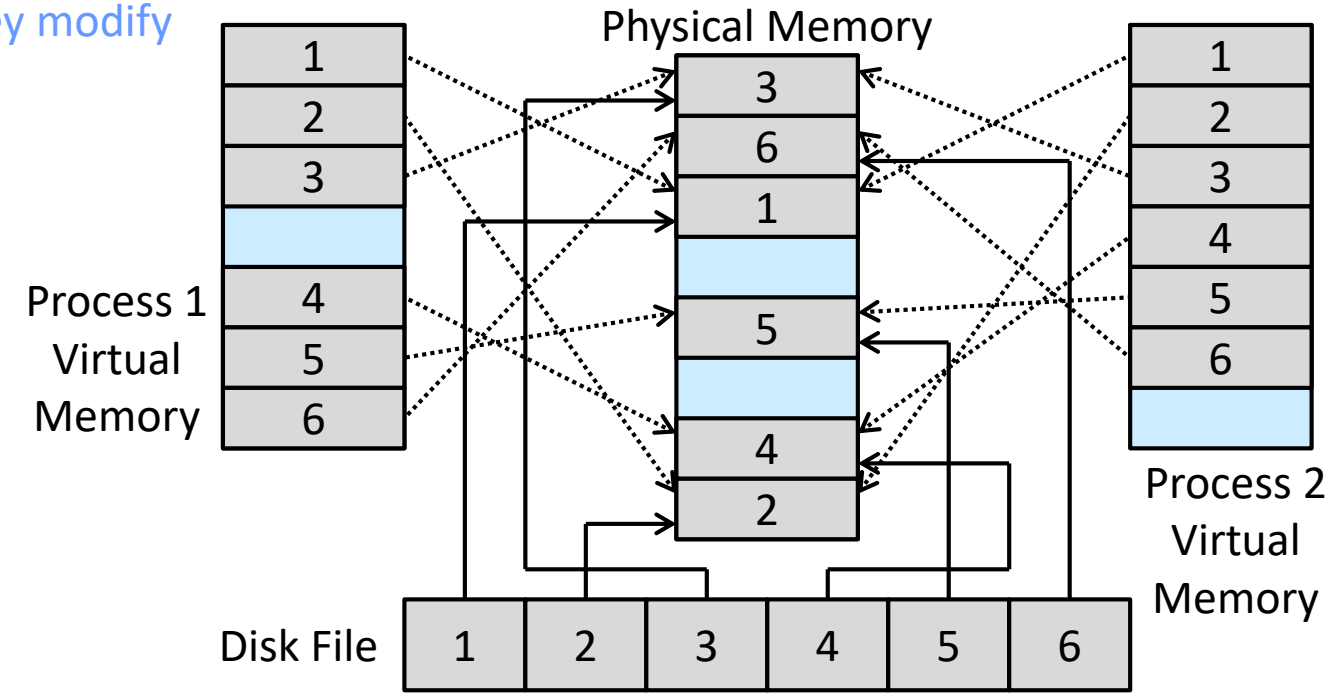
- File Concept
- Access Methods
- Directory Structure
- Protection
- **Memory-Mapped Files**
  - **Basic Mechanism**
  - Shared Memory in Windows API

Basic Mechanism

- **Memory mapping a file** allows a part of the virtual address space to be logically associated with the file
  - Initial access to the file proceeds through ordinary demand paging, resulting in a page fault
    - A page-sized portion of the file is read from the file system into a physical page
    - Some systems may read in more than a page-sized chunk of memory at a time
  - Subsequent reads and writes to the file are handled as routine memory accesses
    - Manipulating files through memory rather than using the `read()` and `write()` system calls simplifies and speeds up file access and usage
  - When the file is closed, all the memory-mapped data are written back to the file on secondary storage
    - Note that writes to the file mapped in memory are not necessarily immediate (synchronous) writes to the file on secondary storage

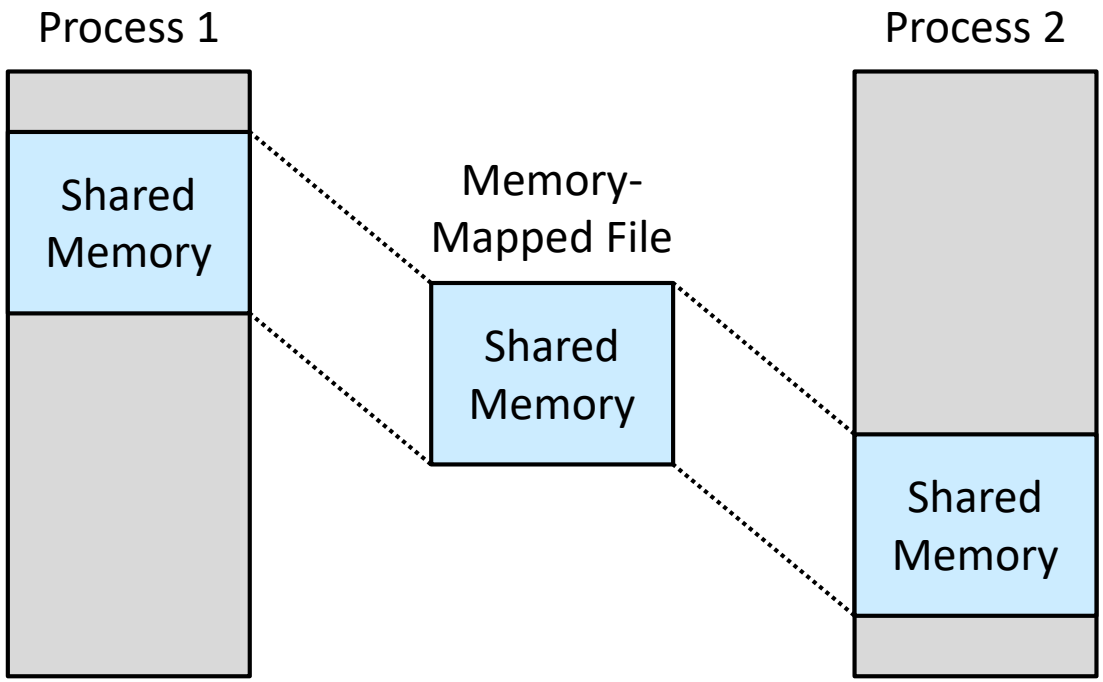
Sharing Between Processes

- Multiple processes may be allowed to map the same file concurrently to allow sharing of data
  - Writes by any of the processes modify the data in virtual memory and can be seen by all others that map the same section of the file
  - They can also support copy-on-write functionality
    - Allow processes to share a file in read-only mode but have their own copies of any data they modify



Implementing Shared Memory

- Quite often, shared memory is implemented by memory mapping files
  - Processes can communicate using shared memory by mapping the same file into their virtual address spaces



Outline

- File Concept
- Access Methods
- Directory Structure
- Protection
- **Memory-Mapped Files**
  - Basic Mechanism
  - **Shared Memory in Windows API**

Shared Memory in Windows API

- **`CreateFile()`** opens a file to be mapped
- **`CreateFileMapping()`** creates a mapping of a file
- **`MapViewOfFile()`** establishes a view of the mapped file in its virtual address space
  - The view of the mapped file represents the portion of the file being mapped in the virtual address space of the process
    - The entire file or only a portion of it may be mapped

Objectives

- Explain the function of file systems
- Describe the interfaces to file systems
- Discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- Explore file-system protection

Q&A