File Systems

Yi Shi

Fall 2018

Xi'an Jiaotong University

Goals for Today: User's perspective of FS

- File system motivation
- Files
  - Naming, structure, types, access, attributes, operations
- Directories
  - Structure, path and operations
- Mounting file systems
- File Protection

- Next Lecture,
  - Implementing internal file system structures

Storing Information

- So far…
  - we have discussed processor, memory
- How do we make stored information usable?
- Applications can store information in the process address space
- Why is it a bad idea for permanent storage?
  - Size is limited to size of virtual address space
    - May not be sufficient for airline reservations, banking, etc.
  - The data is lost when the application terminates
    - Even when computer crashes!
  - Multiple process might want to access the same data
    - Imagine a telephone directory part of one process

File Systems

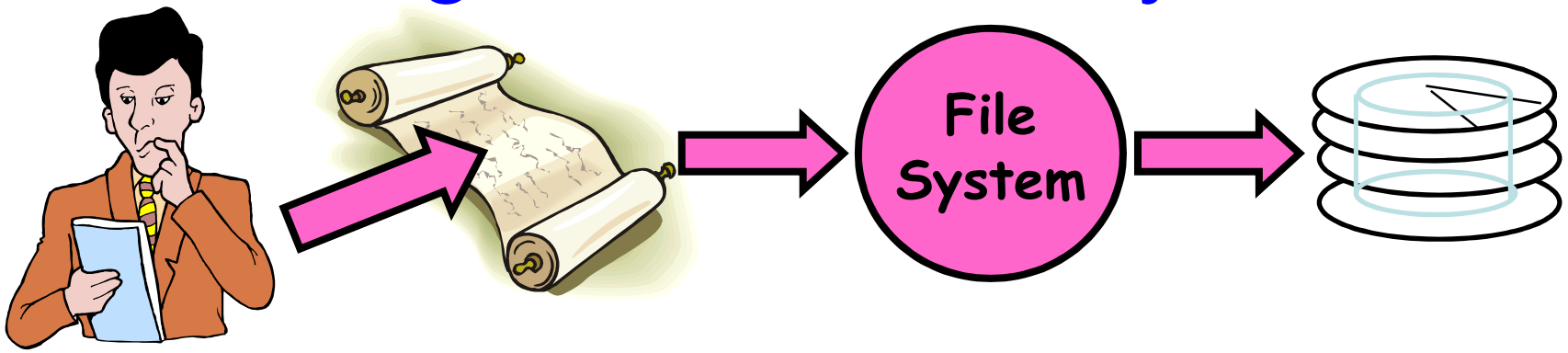
- 3 criteria for long-term information storage:
  - Should be able to store very large amount of information
  - Information must survive the processes using it
  - Should provide concurrent access to multiple processes
- Solution:
  - Store information on disks in units called **files**
  - Files are persistent, and only owner can explicitly delete it
  - Files are managed by the OS

- File Systems: How the OS manages files!

File System

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- File System Components
  - Disk Management: collecting disk blocks into files
  - Naming: Interface to find files by name, not by blocks
  - Protection: Layers to keep data secure
  - Reliability/Durability: Keeping of files durable despite crashes, media failures, attacks, etc
- User vs. System View of a File
  - User's view:
    - Durable Data Structures
  - System's view (system call interface):
    - Collection of Bytes (UNIX)
    - Doesn't matter to system what kind of data structures you want to store on disk!
  - System's view (inside OS):
    - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
    - Block size $\geq$ sector size; in UNIX, block size is 4KB

Translating from User to System View



- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block
- Everything inside File System is in whole size blocks
  - For example, `getc()`, `putc()` ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks (i.e. systems view inside OS)

Disk Management Policies

- Basic entities on a disk:
  - File: user-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files (next lecture)
- Access disk as linear array of sectors. Two Options:
  - Identify sectors as vectors [cylinder, surface, sector]. Sort in cylinder-major order
  - Logical Block Addressing (LBA). Every sector has integer address from zero up to max number of sectors.
  - Controller translates from address $\Rightarrow$ physical position
- Need way to track free disk blocks
  - Link free blocks together $\Rightarrow$ too slow today
  - Use bitmap to represent free space on disk
- Need way to structure files: File Header (next lecture)
  - Track which blocks belong at which offsets within the logical file structure
  - Optimize placement of files' disk blocks to match access and usage patterns

File System Patterns

- How do users access files?
  - Sequential Access
    - bytes read in order (“give me the next X bytes, then give me next, etc”)
  - Random Access
    - read/write element out of middle of array (“give me bytes i—j”)

- What are file sizes?
  - Most files are small (for example, .login, .c, .o, .class files, etc)
  - Few files are large (for example, core files, etc.)
  - Large files use up most of the disk space and bandwidth to/from disk

File Naming

- Motivation: Files abstract information stored on disk
  - You do not need to remember block, sector, …
  - We have human readable names
- How does it work?
  - Process creates a file, and gives it a name
    - Other processes can access the file by that name
  - Naming conventions are OS dependent
    - Usually names as long as 255 characters is allowed
    - Digits and special characters are sometimes allowed
    - MS-DOS and Windows are not case sensitive, UNIX family is

File Extensions

- Name divided into 2 parts, second part is the extension
- On UNIX, extensions are not enforced by OS
  - However C compiler might insist on its extensions
    - These extensions are very useful for C
- Windows attaches meaning to extensions
  - Tries to associate applications to file extensions

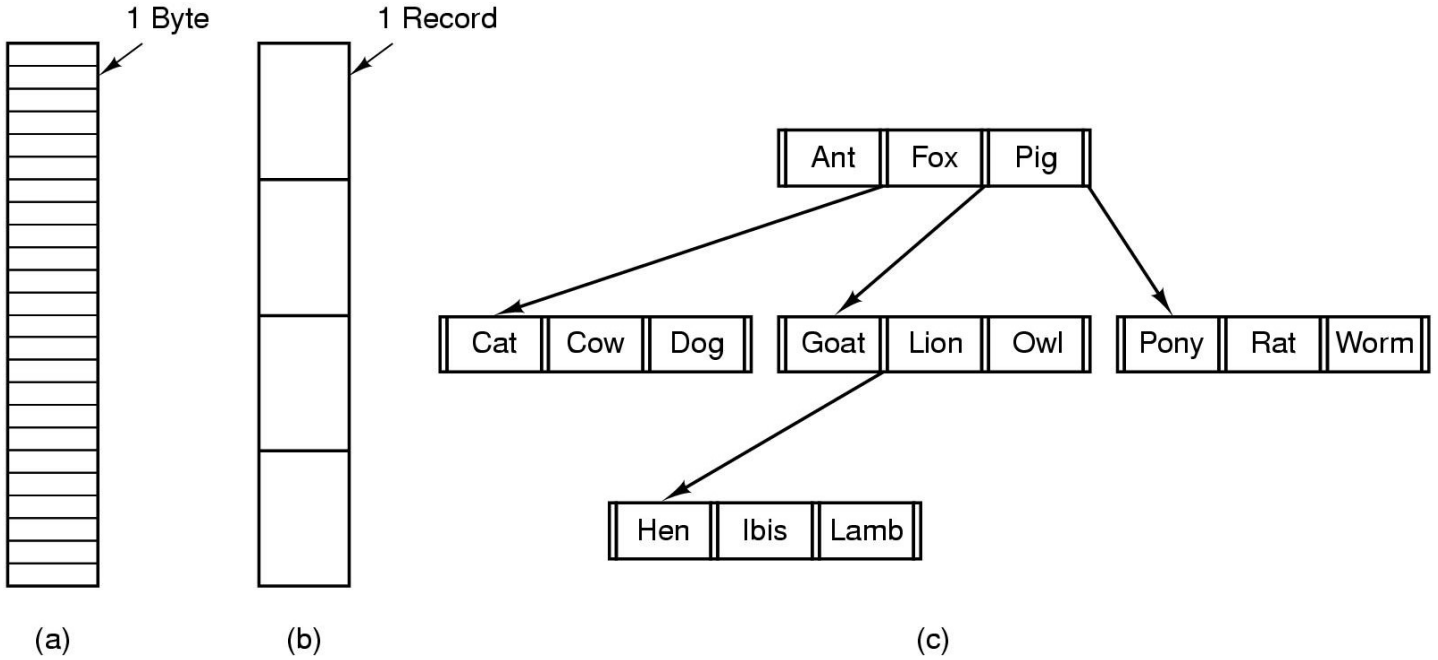
File Structure

(a) Byte Sequence: unstructured, most commonly used

(b) Record sequence: r/w in records, used earlier

(c) Complex structures, e.g. tree

- Data stored in variable length records; location decided by OS



File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or forward
  - convenient when medium was magnetic tape
- Random access
  - bytes/records read in any order
  - essential for database systems
  - 2 possible reads
    - Specify disk block in read
    - move file marker (seek), then read or …

File Attributes

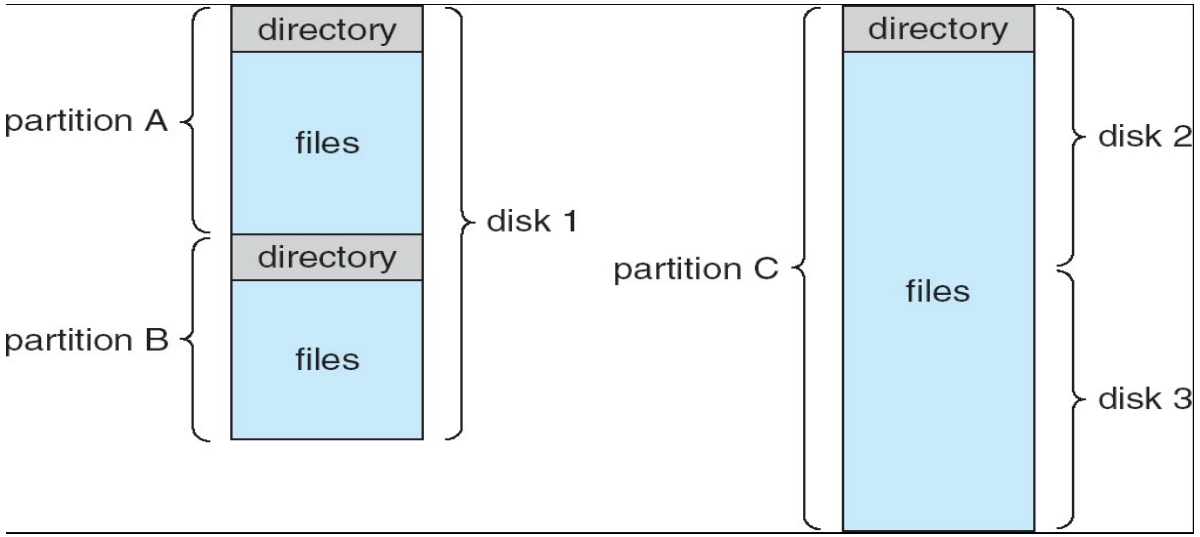
- File-specific info maintained by the OS
  - File size, modification date, creation time, etc.
  - Varies a lot across different OSes
- Some examples:
  - Name – only information kept in human-readable form
  - Identifier – unique tag (number) identifies file within file system
  - Type – needed for 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

File Operations

- File is an Abstract Data Type
- Some basic file operations:
  - *Create*: find space in FS, add directory entry
  - *Write* (e.g. write at current position)
    - Read/write pointer can be stored as per-process file pointer
    - Increase the size attribute
  - *Read* (e.g. read from current position, store in buffer)
  - *Seek*: move current position somewhere in a file
  - *Delete: Remove file from directory entry, mark disk blocks as free*
  - *Truncate: mark disk blocks as free, but keep entry in directory*
    - Reduce the size attribute
  - *Open*: system fetches attributes and disk addresses in memory
  - *Close*

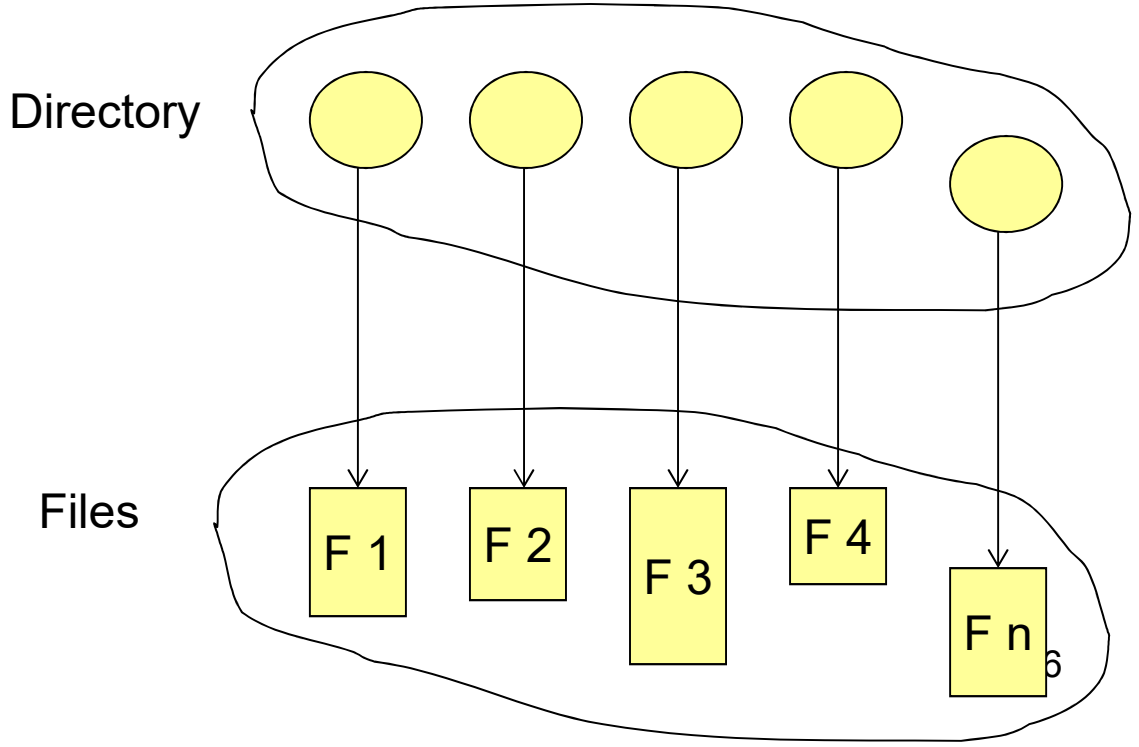
FS on disk

- Could use entire disk space for a FS, but
  - A system could have multiple FSes
  - Want to use some disk space for swap space
- Disk divided into partitions or minidisks
  - Chunk of storage that holds a FS is a volume
  - Directory structure maintains info of all files in the volume
    - Name, location, size, type, …



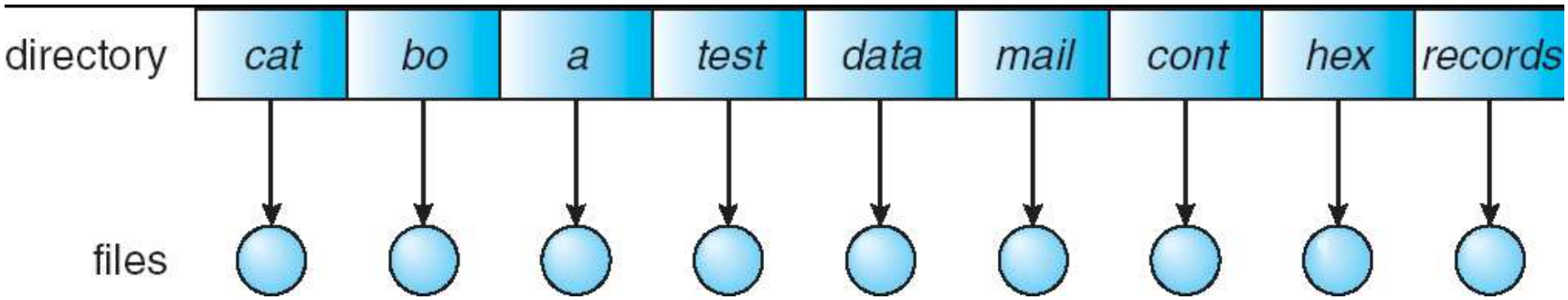
Directories

- Directories/folders keep track of files
  - Is a symbol table that translates file names to directory entries
  - Usually are themselves files
- How to structure the directory to optimize all of the following:
  - Search for a file
  - Create a file
  - Delete a file
  - List directory
  - Rename a file
  - Traversing the FS



Single-level Directory

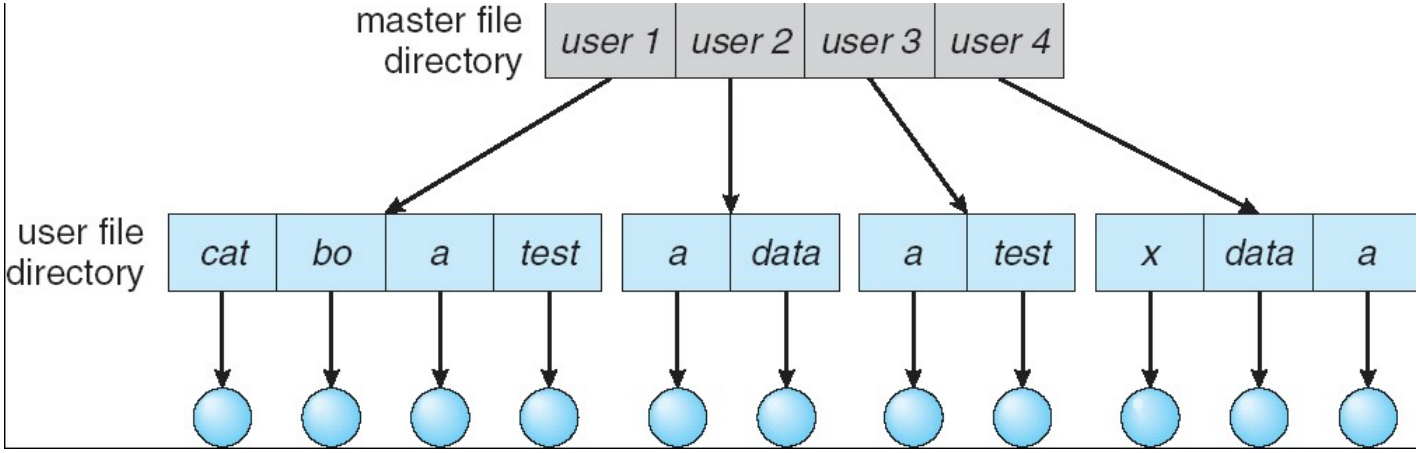
- One directory for all files in the volume
  - Called root directory



  - Used in early PCs, even the first supercomputer CDC 6600
- Pros: simplicity, ability to quickly locate files
- Cons: inconvenient naming (uniqueness, remembering all)

Two-level directory

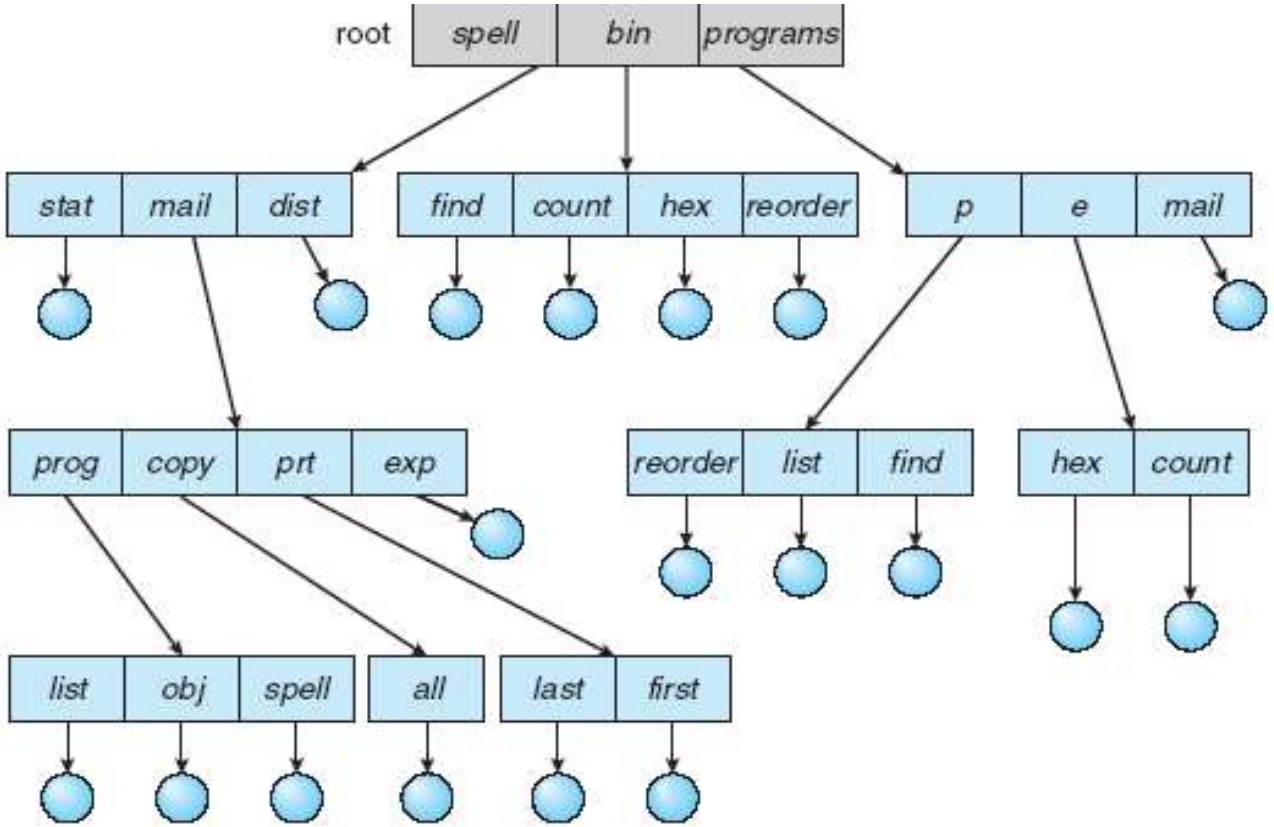
- Each user has a separate directory



- Solves name collision, but what if user has lots of files
- Files need to be addressed by path names
  - Allow user's access to other user's files
  - Need for a search path (for example, locating system files)

Tree-structured Directory

- Directory is now a tree of arbitrary height
  - Directory contains files and subdirectories
  - A bit in directory entry differentiates files from subdirectories

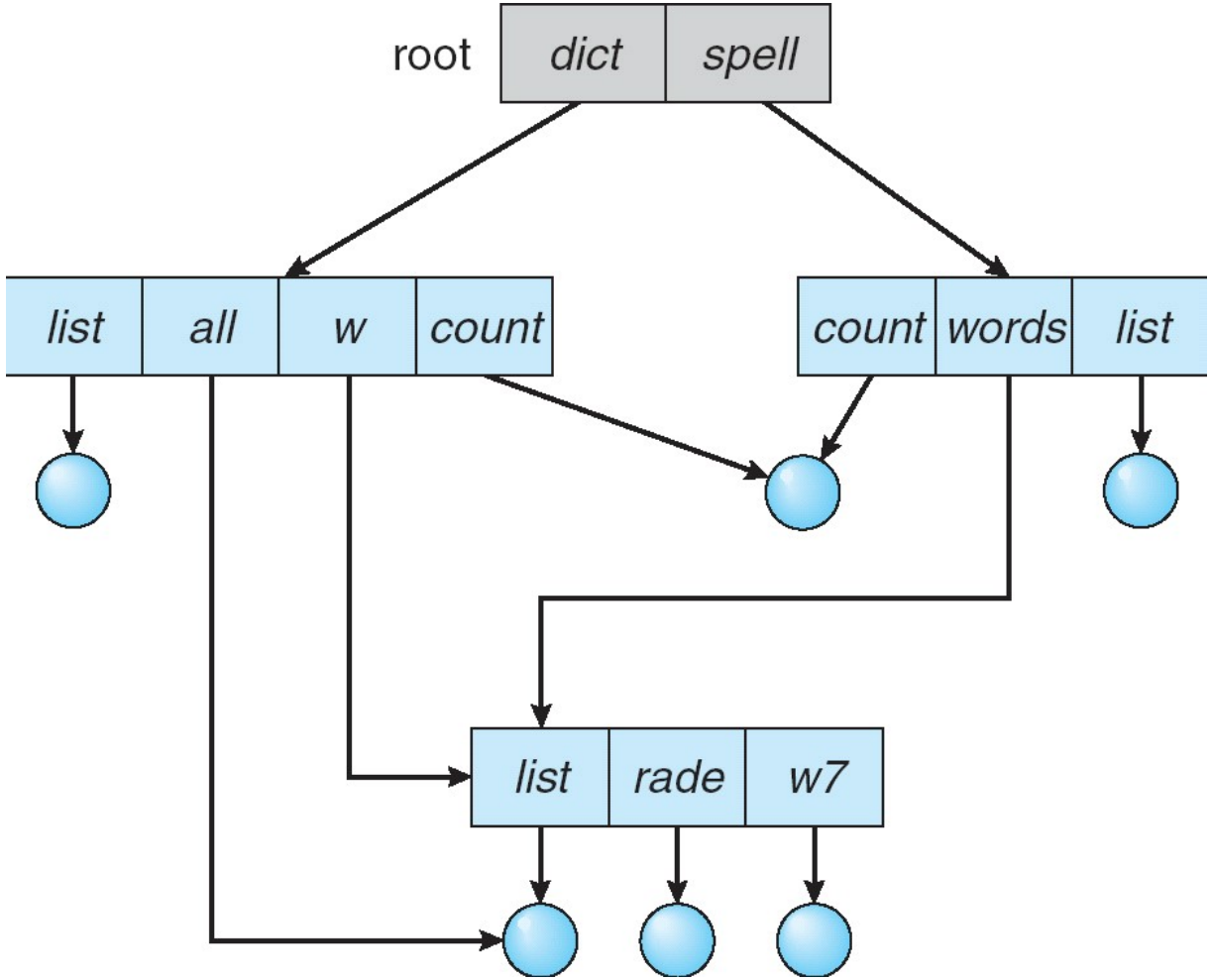


Path Names

- To access a file, the user should either:
  - Go to the directory where file resides, or
  - Specify the **path** where the file is
- Path names are either absolute or relative
  - Absolute: path of file from the root directory
  - Relative: path from the current working directory
- Most OSes have two special entries in each directory:
  - “.” for current directory and “..” for parent

Acyclic Graph Directories

- What about sharing?
- Share subdirectories or files



Acyclic Graph Directories

- How to implement shared files and subdirectories:
  - Why not copy the file?
    - Waste of space, inconsistencies, etc.
  - New directory entry, called Link (used in UNIX)
    - Link is a pointer to another file or subdirectory
    - Links are ignored when traversing FS
- Issues?
  - Two different names (aliasing)
  - If *dict* deletes *list* $\Rightarrow$ dangling pointer
    - Keep backpointers of links for each file
    - Keep reference count of each file
    - Leave the link, and delete only when accessed later

File System Mounting

- Mount allows two FSes to be merged into one
  - For example you insert your floppy into the root FS
- Macintosh, Windows and UNIX

mount("/dev/fd0", "/mnt", 0)

Remote file system mounting

- Same idea, but file system is actually on some other machine
- Implementation uses remote procedure call
  - Package up the user's file system operation
  - Send it to the remote machine where it gets executed like a local request
  - Send back the answer
- Very common in modern systems

File Protection

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

Categories of Users

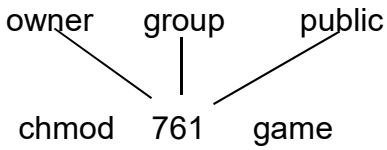
- Individual user
  - Log in establishes a user-id
  - Might be just local on the computer or could be through interaction with a network service
- Groups to which the user belongs
  - For example, “hweather” is in “csfaculty”
  - Again could just be automatic or could involve talking to a service that might assign, say, a temporary cryptographic key

Linux Access Rights

- Mode of access: read, write, execute
- Three classes of users

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

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

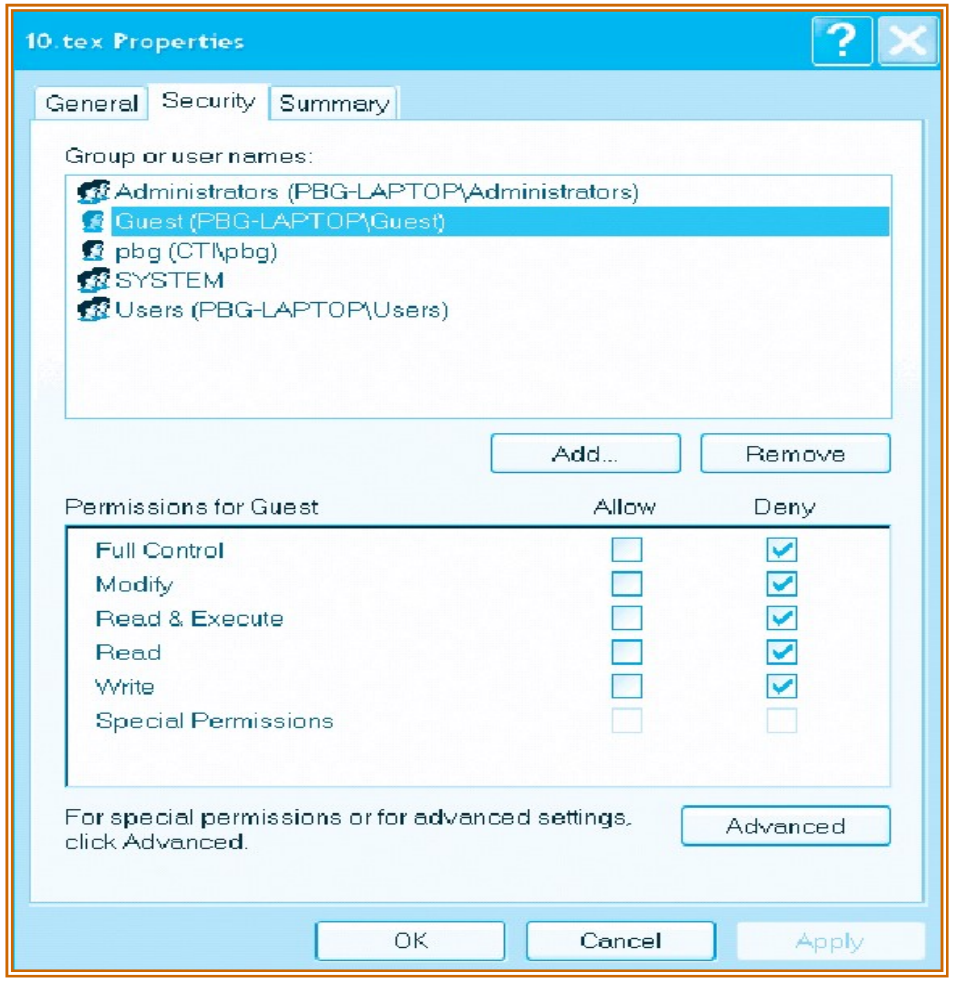



```
$ ls -l game
-rwxrw---x 1 user group 6 Mar 11 12:50 game
```

Issues with Linux

- Just a single owner, a single group and the public
  - Pro: Compact enough to fit in just a few bytes
  - Con: Not very expressive
- *Access Control List:* This is a per-file list that tells who can access that file
  - Pro: Highly expressive
  - Con: Harder to represent in a compact way

XP ACLs


10.tex Properties
General
Security
Summary
Group or user names:
Administrators (PBG-LAPTOP\Administrators)
Guest (PBG-LAPTOP\Guest)
pbg (CTI\pbg)
SYSTEM
Users (PBG-LAPTOP\Users)
Add...
Remove
Permissions for Guest
Allow
Deny
Full Control
Modify
Read & Execute
Read
Write
Special Permissions
For special permissions or for advanced settings, click Advanced.
Advanced
OK
Cancel
Apply

Security and Remote File Systems

- Recall that we can “mount” a file system
  - Local: File systems on multiple disks/volumes
  - Remote: A means of accessing a file system on some other machine
    - Local stub translates file system operations into messages, which it sends to a remote machine
    - Over there, a service receives the message and does the operation, sends back the result
    - Makes a remote file system look “local”

Unix Remote File System Security

- Since early days of Unix, network file system (NFS) has had two modes
  - Secure mode: user, group-id's authenticated each time you boot from a network service that hands out temporary keys
  - Insecure mode: trusts your computer to be truthful about user and group ids
- Most NFS systems run in *insecure* mode!
  - Because of US restrictions on exporting cryptographic code…

Summary

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for access and usage patterns
  - Maximize sequential access, allow efficient random access
- Disk Management
  - collecting disk blocks into files
- Naming
  - the process of turning user-visible names into resources (such as files)
- Protection
  - Layers to keep data secure