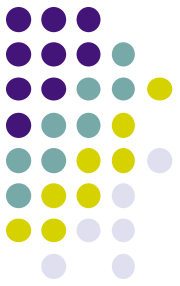


File System Overview

ECE469, March 30

Yiying Zhang





Announcement

- Quiz today (last 10 minutes of the lecture)
- Reading: Chapter 10



Role of OS for I/O

- Standard library
 - Provide abstractions, consistent interface
 - Simplify access to hardware devices
- Resource coordination
 - Provide protection across users/processes
 - Provide fair and efficient performance
 - Requires understanding of underlying device characteristics
- User processes do not have direct access to devices
 - Could crash entire system
 - Could read/write data without appropriate permissions
 - Could hog device unfairly
- OS exports higher-level functions
 - File system: Provides file and directory abstractions
 - File system operations: mkdir, create, read, write

File System: OS's storage (I/O) manager



- The concept of a file system is simple
 - the implementation of the abstraction for secondary storage
 - abstraction = files
 - logical organization of files into directories
 - the directory hierarchy
 - sharing of data between processes, people and machines
 - access control, consistency, ...



Abstraction: File

- User view
 - Named collection of bytes
 - Untyped or typed
 - Examples: text, source, object, executables, application-specific
 - Permanently and conveniently available
- Operating system view
 - Map bytes as collection of blocks on physical non-volatile storage device
 - Magnetic disks, tapes, flash, NVM
 - Persistent across reboots and power failures
- File system performs the magic / translation
 - Pack bytes into disk blocks on writing
 - Unpack them again on reading

Why Files?



- Physical reality

- Block oriented
- Physical sector numbers
- No protection among users of the system
- Data might be corrupted if machine crashes

- File system abstraction

- Byte oriented
- Named files
- Users protected from each other
- Robust to machine failures



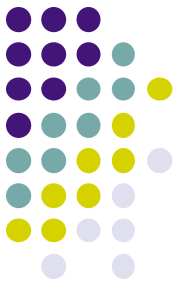
File Meta-Data

- Meta-data: Additional system information associated with each file
 - Name of file
 - Type of file
 - Pointer to data blocks on disk
 - File size
 - Times: Creation, access, modification
 - Owner and group id
 - Protection bits (read or write)
 - Special file? (directory? symbolic link?)
- Meta-data is stored on disk
 - Conceptually: meta-data can be stored as array on disk



Per-file Metadata

- In Unix, the data representing a file is called an inode (for indirect node)
 - Inodes contain file size, access times, owner, permissions
 - Inodes contain information on how to find the file data (locations on disk)
- Every inode has a location on disk.



File System API

- OS provides the file system abstraction
- How do application processes access the file system?



System Calls to UNIX File Systems

- 19 system calls into 6 categories:

Return file desp.	Assign inodes	Set file attr.	Process input/ output	Change file system	Modify view of file system
open close creat pipe dup	creat link unlink	chown chmod stat fstat	read write lseek	mount umount	chdir chroot



File Operations

- Create file with given pathname `/a/b/file`
 - Traverse pathname, allocate meta-data and directory entry
- Read from (or write to) offset in file
 - Find (or allocate) blocks of file on disk; update meta-data
- Delete
 - Remove directory entry, free disk space allocated to file
- Truncate file (set size to 0, keep other attributes)
 - Free disk space allocated to file
- Rename file
 - Change directory entry
- Copy file
 - Allocate new directory entry, find space on disk and copy
- Change access permissions
 - Change permissions in meta-data



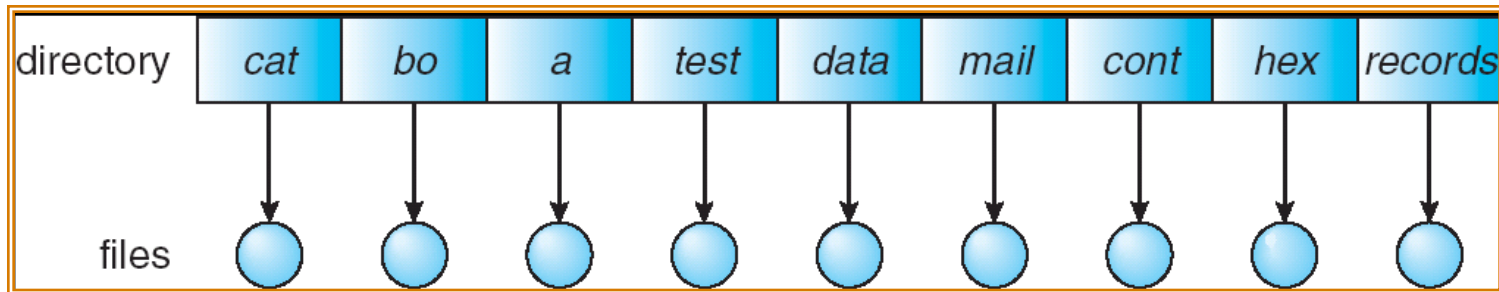
Abstraction: Directories

- Naming and mapping: Map file name to blocks of file data on disk
 - Actually, map file name to file meta-data (which enables one to find data on disk)



Single-Level Directory

- A single directory for all users
 - Called the root directory
 - Each file has unique name

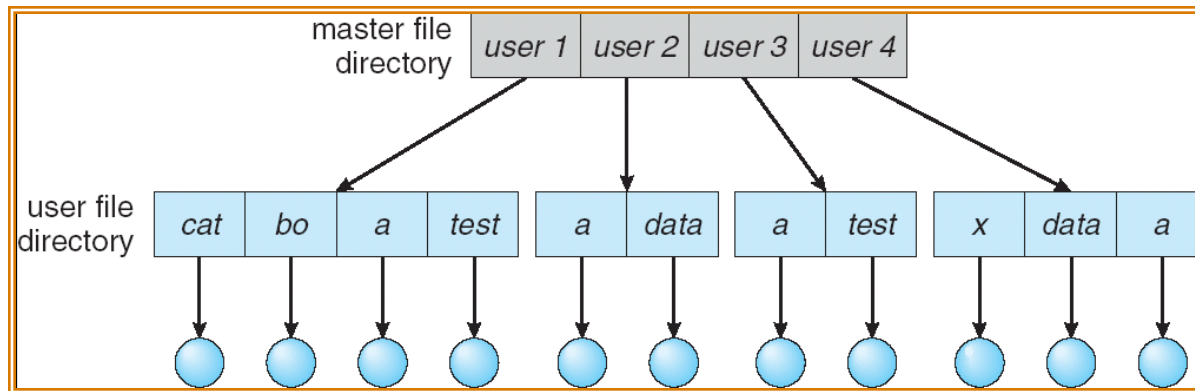


- Pros: Simple, easy to locate files
- Cons: inconvenient naming (uniqueness), no grouping



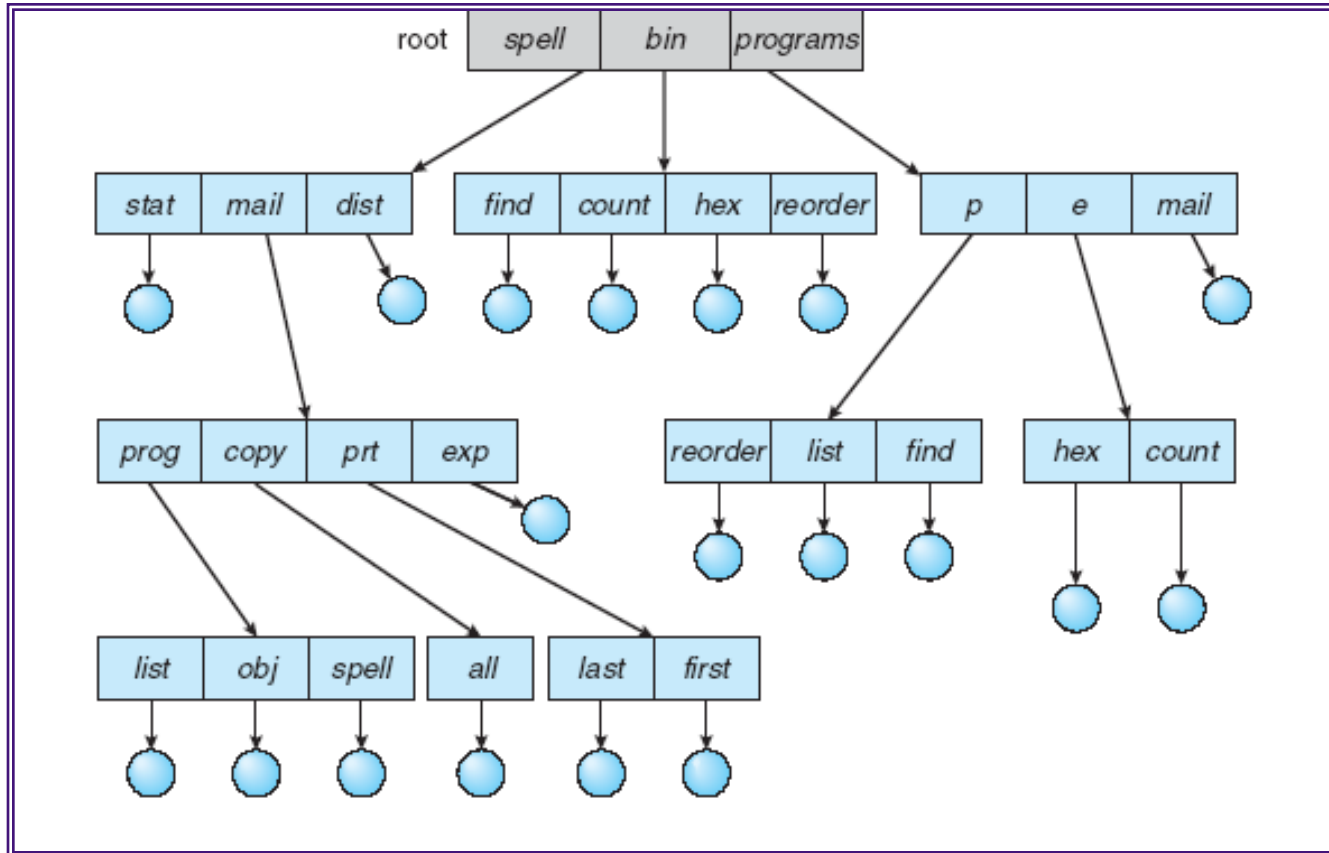
Two-Level Directory

- Separate directory for each user
 - Specify file with user name and file name



- Introduces the notion of a path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



- Directories can now contain files and subdirectories
- Efficient searching, allows grouping



Directory Internals

- A directory is typically just a file that happens to contain special metadata
 - Directory is stored and treated like a file
 - Special bit set in meta-data for directories
 - directory = list of (name of file, metadata of file)
 - metadata about file (Windows)
 - Size
 - Owner
 - data locations
 - Pointer to file metadata (Unix)
 - the directory list is usually unordered (effectively random)
 - when you type “ls”, the “ls” command sorts the results for you

[earlier this lecture] Per-file Metadata



- In unix, the data representing a file is called an inode (for indirect node)
 - Inodes contain file size, access times, owner, permissions
 - Inodes contain information on how to find the file data (locations on disk)
- Every inode has a location on disk.

Hierarchical (directory tree) File System





- A directory is a flat file of fixed-size entries
- Each entry consists of an inode number and a file name
- Special directories
 - Root: Fixed index for metadata (e.g., inode # 2)
 - This directory: .
 - Parent directory: ..

Inode Number	File Name
34	.
63	..
2	file1
56	file2
133	dir1


David Patterson



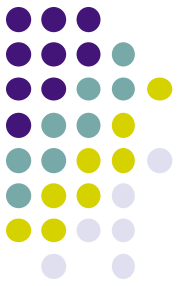
- Architecture
- RISC (RISC-V)
- RAID
- Architecture book



Retirement Celebration for Professor David A. Patterson
Friday, May 06, 2016 5:30 PM - Saturday, May 07, 2016 5:00 PM (Pacific Time)

University of California, Berkeley
Berkeley, California 94720
United States
patterson2016@eecs.berkeley.edu  [Email Us](#)

<https://www.regonline.com/Register/Checkin.aspx?EventID=1820518>



Using Directories

- How do you find files?
 - Read the directory, search for the name you want (checking for wildcards)
- How do you list files (ls)
 - Read directory contents, print name field
- How do you list file attributes (ls -l)
 - Read directory contents, open inodes, print name + attributes
- How do you remove a file?
 - Remove entry from directory, delete inode, delete data

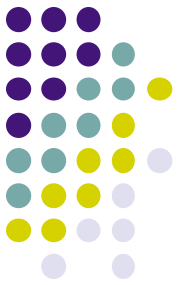
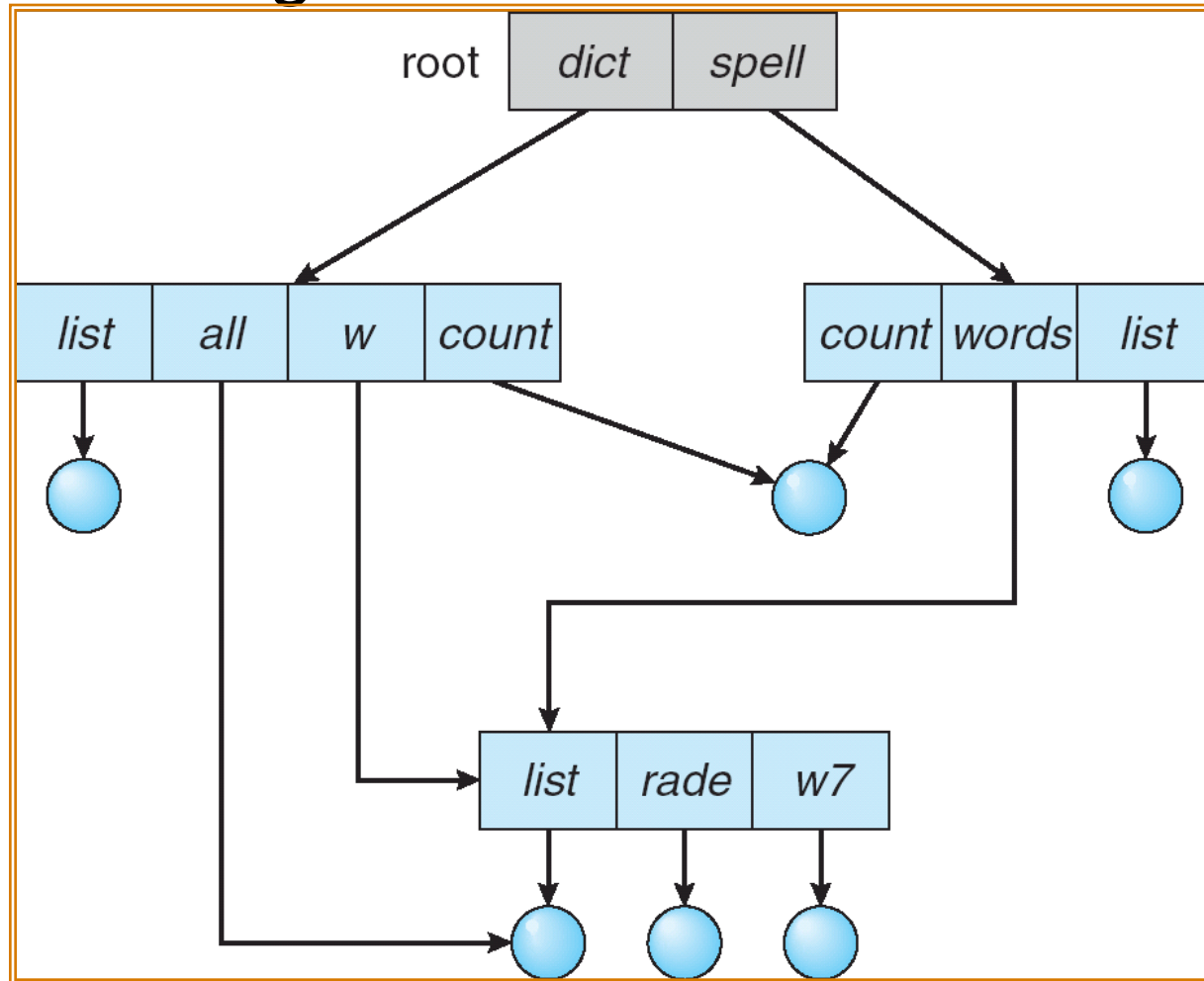
Path Name Translation



- Let's say you want to open `"/one/two/three"`
`fd = open("/one/two/three", O_RDWR);`
- What goes on inside the file system?
 - open directory `"/"` (well known, can always find)
 - search the directory for `"one"`, get location of `"one"`
 - open directory `"one"`, search for `"two"`, get location of `"two"`
 - open directory `"two"`, search for `"three"`, get loc. of `"three"`
 - open file `"three"`
 - (of course, permissions are checked at each step)
- Another example: `mkdir /a/b/c`
 - Read inode 2 (root), look for `"a"`: find `<"a", 5>`
 - Read inode 5, look for `"b"`: find `<"b", 9>`
 - Read inode 9, verify no `"c"` exists; allocate `c` and add `"c"` to directory
- FS spends lots of time walking down directory paths
 - this is why open is separate from read/write
 - OS will cache prefix lookups to enhance performance
 - `/a/b`, `/a/bb`, `/a/bbb` all share the `"a"` prefix

Acyclic-Graph Directories

- Allow sharing of subdirectories and files



Acyclic-Graph Directories



- More general than tree structure
 - Add connections across the tree (no cycles)
 - Create **links** from one file (or directory) to another
- Hard link: “ln a b” (“a” must exist already)
 - Idea: Can use name “a” or “b” to get to same file data
 - Implementation: Multiple directory entries point to same metadata
 - What happens when you remove a? Does b still exist?
 - How is this feature implemented???



Acyclic-Graph Directories

- Symbolic (soft) link: “ln -s a b”
 - Can use name “a” or “b” to get to same file data, if “a” exists
 - When reference “b”, lookup soft link pathname
 - b: Special file (designated by bit in meta-data)
 - Contents of b contain name of “a”
 - Optimization: In directory entry for “b”, put soft link filename “a”



Steps to Create a File

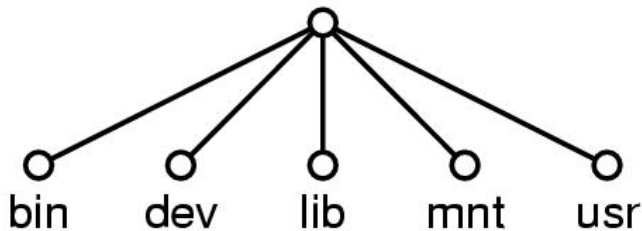
- 1. Check if name is in use
 - a. Find directory inode
 - b. Read directory contents for existing files
- 2. Allocate inode
 - a. Update from free inode bitmap/list
 - b. Fill in inode contents
- 3. Add an inode to directory
 - a. Write back directory contents
- What happens if you do this in the wrong order?

File System Mounting

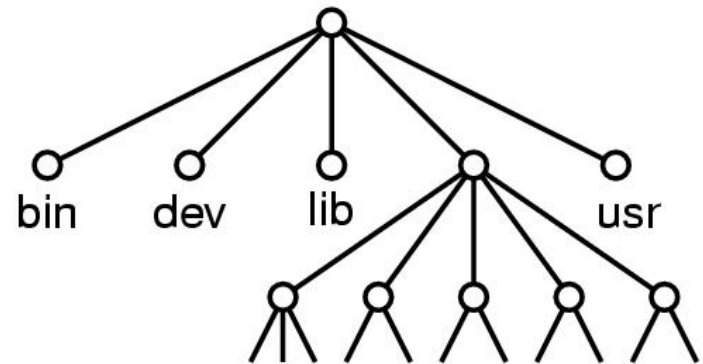


- Mounting stores in memory a pointer from a directory on one partition to the root of another partition
 - During access, the OS checks every directory to see if it is a real directory or a mount point
 - If mount point, follows pointer in memory to find the root directory of the other file system.

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



(a)



(b)



Opening Files

- Expensive to access files with full pathnames
 - On every read/write operation:
 - Traverse directory structure
 - Check access permissions
- Open() file before first access
 - User specifies mode: read and/or write
 - Search directories for filename and check permissions
 - Copy relevant meta-data to open file table in memory
 - Return index in open file table to process (file descriptor)
 - Process uses file descriptor to read/write to file
- Per-process open file table
 - Current position in file (offset for reads and writes)
 - Is this metadata? Does this metadata need to be stored on disk?
 - Open mode



Reading/Writing Files

- Default is sequential access
 - OS remembers last position in file object referenced by a file descriptor
- `Read(fd,buf,size)` reads the next size bytes in the file referred to by fd
- Random access:
 - `Lseek(fd, offset, whence)` move the file position
 - `SEEK_SET`: set to offset
 - `SEEK_CUR`: set to current pos + offset
 - `SEEK_END`: set to end of file + offset (usually 0)
 - `Pread(fd, buf, size, offset)`
 - Read from position offset without adjusting file position



File Access Methods

- Some file systems provide different access methods that specify ways the application will access data
 - Sequential access
 - read bytes one at a time, in order
 - Random access
 - access given a part of a file by block/byte #
- Why do we care about distinguishing sequential from random access?



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, “yiying” is in “ece469”



UNIX 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

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

