File Systems

Ken Birman

Storing Information

- We often need to store information
  - Sometimes, the stored form has a life its own
    - Pictures of your friends, videos, songs
  - Sometimes, the stored data is for recovery
    - In case you want to restart the game later
    - Or perhaps your application is a little fragile and crashes now and then
  - Files are a good way for processes to cooperate
    - You do X, I'll do Y and we'll merge the results

2

File Systems

- Three 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!

3

Today: User's perspective of FS

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

4

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

5

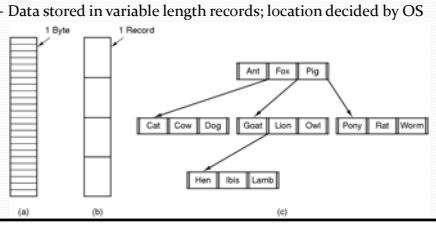
File Extensions

- 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
  - You can see and even change these if you like

6

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



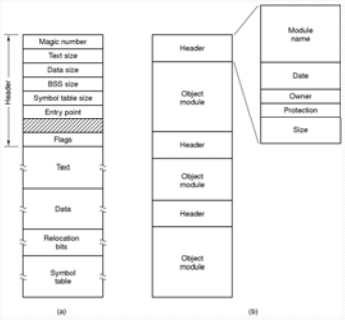
7

File Types

- 5 types of files
  - Regular files: contain user information
  - Directories: system files for maintaining structure of FS
  - Character special files: for serial I/O in UNIX
  - Block special files: to model disks in UNIX
  - Symbolic links
- Regular files are usually:
  - ASCII files: lines of text
    - Useful for editing, portability across applications
  - Binary files: usually have an internal structure
    - Look at executables and archives in UNIX
    - Every OS needs a way to recognize its own executable!

8

Executables and Archives



9

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 ...

10

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

11

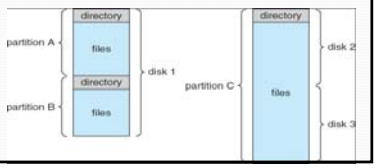
File Operations

- File is an Abstract Data Type
- Some operations:
  - Create a file: find space in FS, add directory entry
  - Open: system fetches attributes and disk addresses in memory
  - Write a file: locate file and write at current position
    - Might need to increase the size attribute
  - Read a file: locate file, read from current position, store in buffer
    - Read/write pointer can be stored as per-process file pointer

12

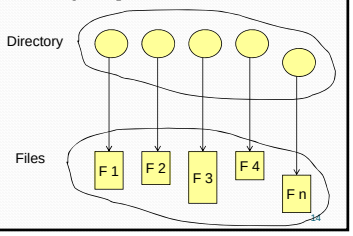
FS on disk

- 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, …



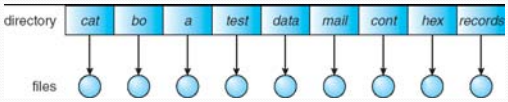
Directories

- 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 foll.:
  - Search a file
  - Create a file
  - Delete a file
  - List directory
  - Rename a file
  - Traversing the FS



Single-level Directory

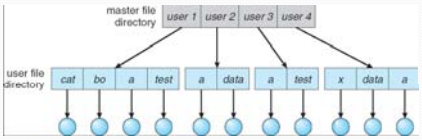
- One directory for all files in the volume
  - Called root directory



  - 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

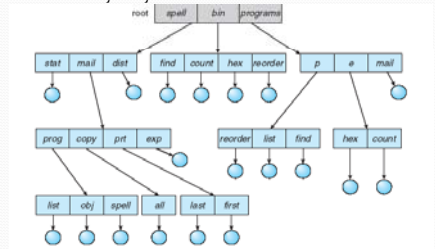
- Each user has a separate directory



- 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

Links

- In Linux, a name is really a "link"
  - We adopt the view that the name (path) leads us to a kind of unique file identification number
  - Called an *inode number* and, on a given disk, it refers to the data structure representing the file
- In Linux, multiple names can link to the same inode!
  - The file ends up with more than one name but there is just one file and everyone shares it
  - Permissions are a property of the file, not its name(s)

19

Shortcuts, Symbolic Links

- On Windows, there is a different way to create a kind of link
  - A file on Windows only has one real name
  - But you can create a "shortcut" to the file
- Linux has these too, in addition to true links. Calls them "symbolic links"
- A shortcut, or symbolic link, is a file that has another file name inside it.

20

Pros and Cons

- True links have advantages and disadvantages
  - The main disadvantage is that deleting a file may not get rid of it, which can be confusing
  - E.g. if two names point to "memo" and you delete one name (one link), the other still points to the file
- But a nice feature of a true link is that it lets you think of the name space separately from the files per-se

21

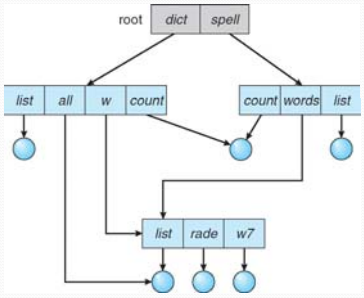
Pros and Cons

- Symbolic links (shortcuts)
  - Are super flexible: the file doesn't even need to exist when you make the shortcut (like a URL)
  - But this is a problem too: the file has no idea that other names point to it, so when you delete the file, it goes away even if symbolic links persist
- Thought question: *suppose that "memo" has a symbolic link to it called "Mom's Memo" and now I delete memo and create a new file called memo. What happens if I open "Mom's Memo"? What it was a real link?*

22

Acyclic Graph Directories

- Share subdirectories or files



23

Acyclic Graph Directories

- How to implement shared files and subdirectories:
  - Why not copy the file?
  - New directory entry, called Link (used in UNIX)
    - Link is a pointer to another file or subdirectory
    - Links are ignored when traversing FS
    - *ln* in UNIX, *fsutil* in Windows for hard links
    - *ln –s* in UNIX, shortcuts in Windows for soft links
- Issues?
  - Two different names (aliasing)
  - If *dict* deletes *list* ⇒ dangling pointer
    - Keep backpointers of links for each file
    - Leave the link, and delete only when accessed later
    - Keep reference count of each file

24

File System Mounting

- Mount allows two FSes to be merged into one
  - For example you insert your floppy into the root FS

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

25

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

26

Executables in mounted file sys

- A very controversial feature!
  - For example, in a USB disk
- Executables enable features like "autoplay"... which are extremely popular
  - Although there are other ways to support autoplay, often it is done by running an "autoexec" file from the root
- But should the OS trust the contents of a USB?
  - Who has it been sleeping with?

27

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

28

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, "ken" is in "csfaculty"
  - Again could just be automatic or could involve talking to a service that might assign, say, a temporary cryptographic key

29

Linux Access Rights

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

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

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

30

Unix executable files

- In Linux, an executable file can be
  - A text file. In this case Linux runs the shell program on the file (treats the file as if it contained commands)
  - The first line of the file can specify *which* shell you prefer for it to use (Unix has several shells)
  - A very popular one is "Perl"
- Linux executables can also specify
  - *Setuid:* means "run under the UID of the file owner"
  - Se*tgid:* means "un under the GID of the file owner"
  - *… used to control access to special applications, like medical records or accounting systems*

31

Setuid "root" ("administrator")

- This is a very risky feature but common
  - It allows a program to gain complete control
  - Overrides all file system and other permissions
- Basically, "become god"
- Unix allows it… much debate about the right way to handle USB disks that contain setuid programs….

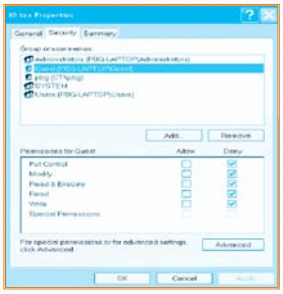
32

More 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

33

XP ACLs



34

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"

35

Unix Remote File System Security

- Since early days of Unix, 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…

36

Spoofing

- Question: what stops you from "spoofing" by building NFS packets of your own that lie about id?
- Answer?
  - In insecure mode… nothing!
  - In fact people have written this kind of code
  - Many NFS systems are wide open to this form of attack, often only the firewall protects them

37