Memory I: Intro to File Management

44-550: Operating Systems

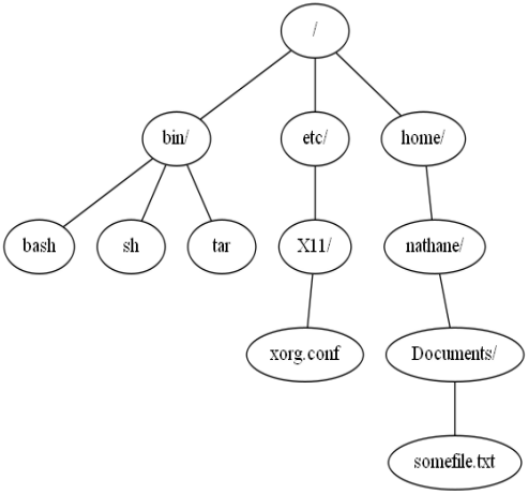
File Attributes

- Filename
- Size
- Location
- Type
  - Denoted (to the user) with a file extension, frequently
  - Some OSes use file extensions to denote what program to execute when opening a file
- Permissions
  - Defines what a user may or may not do to a file
  - Users may read (r), write(w), and execute(x).
    - On Linux/Unix: rwxrwxrwx
    - refers to owner, group, and everyone
- Timestamp

Files

- Files are frequently organized in *trees*
  - Folders/directories are internal nodes
  - Files are leaf nodes
- The *absolute path* of a file can be found by reading listing the nodes from the root of the tree
- A file's *relative path* can be read by starting at any internal node
  - an absolute path is just the relative path when the working directory is the root
  - You can always move to a node's parent using the directory ..
- Files can have symbolic links (shortcuts)
  - These are direct links from one location to another (without going through the entire directory structure)

A File Tree (Linux)



File Access: Open

- Needed on read or write
- OS needs to know the file's location
  - Requires traversing the directories
- Requires disk access (slooooooooow)
  - Want to avoid doing this with every read and write (open file once, do many operations)

File Access: Open

- Solution: Open file once, traverse directories once, and cache the directory entry in main memory
  - Fast access to the file
  - Calling open returns a file handle
    - Identifies the cached directory entry
- When you open a file, you must specify:
  - Whether reading or writing
  - Whether sequential or random access
- As an interesting note, writing can either:
  - append (begin writing at end of file)
  - overwrite (begin writing at beginning of file)

File Access: Close

- More than one function
- The close operation destroys a cached directory entry
- Before the entry is destroyed, it ensures that all data has been written (might be a buffer)
- Ensures that removable media (flash drives, etc) can be removed

File Access: Reading

- Reads data from file into the memory space of the process requesting the data
- The management system needs several pieces of information:
  - Which file? Where is it? (What is the *handle*)
  - Where is the data in the file?
  - What portion of the file should be read?
  - Where should the data be stored?
  - How much data should be read?
- Typically, a system level read call is: read(handle, file position, buffer, length)
  - Frequently abstracted away (BufferedReader, etc)
  - You've seen this operation with sockets

File Access: Writing

- Writes data to file frin the memory space of the process requesting the data
- The management system needs several pieces of information:
  - Which file? Where is it? (What is the *handle*)
  - Where is the data in the file?
  - Where should the data be stored/written to?
  - Where is the buffer that stores the data in memory?
- Typically, a system level read call is: write(handle, file position, buffer, length)
  - Frequently abstracted away (BufferedReader, etc)
  - You've seen this operation with sockets

Sequential Access

- Sequential access
  - Management system keeps track of current file position for reading and writing
- System maintains a file pointer for the position of the next read or write
  - Frequently set to zero to start reading/writing at the beginning of a file, or the end of the file to append data
- Pointer is incremented by the amount of data read or written after each operation

Streams, Pipes, and Redirection

- We've used many of these
- Stream
  - Flow of data bytes (sequentially) into the process (read) and out of the process (write)
  - Used by languages such as Java, C++, .NET...
- We've seen pipes in detail

Standard I/O Files

- In order to understand redirection, you must understand the standard I/O file handles
  - stdin: takes input from the user keyboard (typically)
  - stdout: writes to the screen
  - stderr: used for process error messages
- Each of these files is a pipe that connects to a system module that communicates with the appropriate device
- The CLI supports I/O redirection
- Using redirection, the user can:
  - Take data from a file or another process using redirection isntead of stdin
  - Send data to another file or process instead of stdout or stderr

Redirection

File redirection means sending input data to or forwarding output data to another file as input or output, respectively, via the command line. Examples:

- ls $>$ files.txt
- less $<$ files.txt
- ls | wc -l

When we get into "advanced" bash usage, you will learn to use this technique in very powerful and flexible ways.

Other I/O Calls

- Lots of other system I/O calls
  - flush
  - directory functions
  - create file
  - delete file
  - rename file
  - file exists
  - directory list
  - get/set attributes

File Space Allocation

- To store files, you need to allocate space for them
- Contiguous allocation is simple and gives optimal RW performance. But...
  - How much space will the file take?
  - After long run times, the system becomes cluttered
  - Need to move files around
- Cluster allocation solves these problems
  - Allocate pieces of data and link them together
  - Can also index these clusters
  - Think linked list vs array...

File Space Allocation

- Read and write addresses
  - Usually use a file pointer
  - Use the offset to get the address
  - Much harder when using clusters
  - Data requests may cross cluster boundaries
- Free space management
  - Keep track of clusters allocated to each file
  - Use a linked list or bit mapping
    - each bit in the bitmap represents a cluster (1 = used, 0 = free)