CS 326: Operating Systems

File Systems

Lecture 15


STOP
programming computers
Sand was never meant to think
this is very cruel to rocks


rm -rf /
rm: permission denied


sudo rm -rf /
rm: it is dangerous to operate recursively on '/'
sudo rm -rf / --no-preserve-root

Today's Agenda

- File System Basics
- Mount Points

Today's Agenda

- **File System Basics**
- Mount Points

Who Needs Disks?

- System memory (RAM) is a great way to store data
  - It's super fast!
  - As we studied over the last few weeks, we can even share it among our processes
- So why would we need disks, a.k.a. **secondary storage**?
- Large hard disks are more cost effective ($/MB)
- Hard disks retain information even when they are powered off

Disk Trade-Offs

- Pros
  - Cheap
  - Big
  - Lots of connectivity options
  - USB External, SATA, etc.
- Cons
  - Slow!
  - Slow!
  - (repeated for emphasis)
  - Spinning (mechanical) disks are more prone to failure

A Simplistic View of Disks

- We can view a disk as a giant array of bytes
  - Want to store something? Just go and update the data in the array
- However, as usual, we don't give each process its own disk
- Rather than dividing up the disk by **process**, we represent it using one or more **file systems**
  - Dividing up space based on files instead

Files

- Files are a better abstraction for data stored on disks because sharing happens **much** more often
- We can open a text file in vim, edit it, and then compile it using gcc
- Having an explicit mechanism for these sharing operations goes a long way towards program interoperability
- A **file system** defines how files are organized and laid out on the disk

A Basic File System

- So, we have this large array of bytes and we want to split it up into **files…**
- We could take an approach that looks like our malloc() implementation:
  - Prefix each file with some metadata about its size, file name, etc.
  - Include the file data right after
  - Pointer to the next file on the disk
- Issues?

Problems with our Basic FS

- We don’t have support for folders
  - “Flat” namespace
- Operations like `ls` take forever because we have to scan the entire linked list
- Finding a file also takes a long time
  - You thought O(n) was bad on a CPU? Try it on a hard disk!
- We’ll have fragmentation when files are deleted or moved around

Metadata

- The first thing we need is a way to store file metadata
- **inodes**
  - Nobody really knows what the ‘i’ stands for, but it’s thought to be ‘index’ – ‘index node.’
- inodes contain file ownership, permissions, serial numbers, timestamps, size info, etc.
- *But not the file name!*

Indexing: dentry

- We need to create an index of the files we have stored on the disk
- Rather than reading the entire file system each time, we'll just consult the index
- File name information is stored in **dentries**
  - Directory entries
- File names get mapped to inode numbers

Link Count [1/3]

- Separating inodes from file names gives us more flexibility
- Each file name **links** to an inode
- We can have several file names all pointing at the same inode to create duplicates
  - No impact on disk space!
- Try it for yourself: the `ln` command
  - Related: symlinks (`ln -s`)

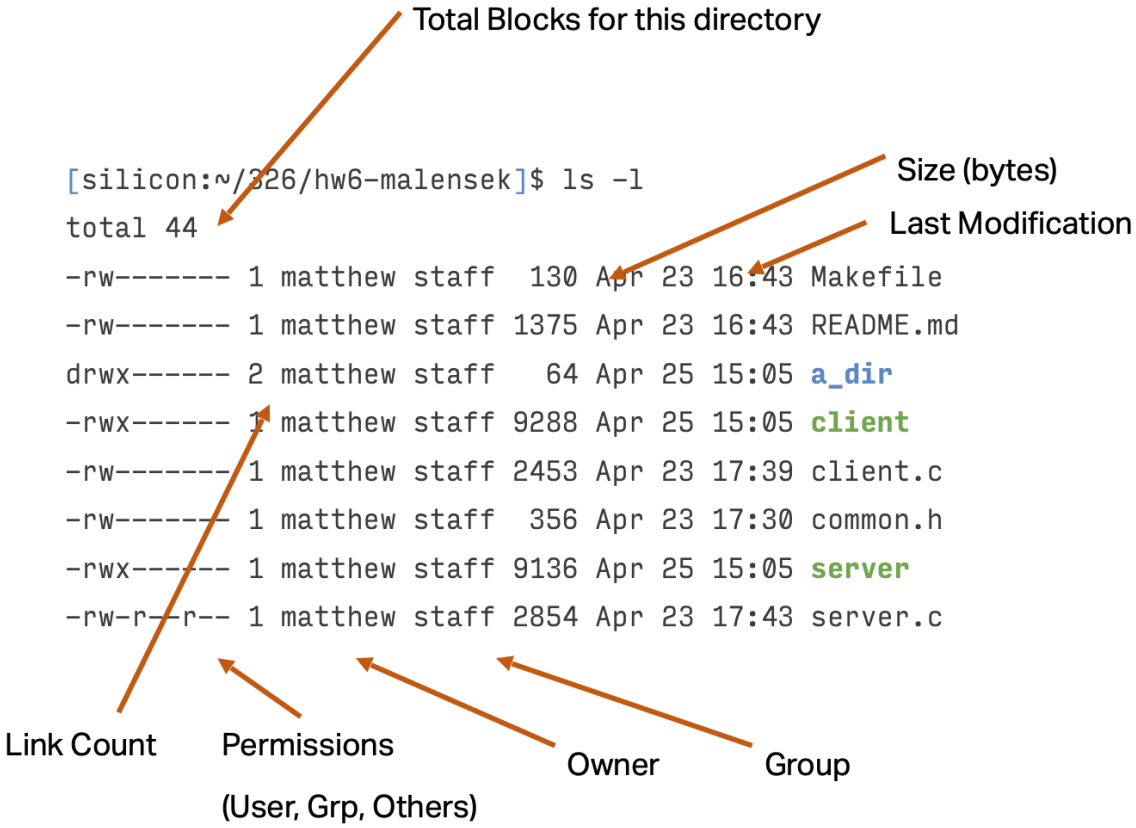
Link Count [2/3]

- When you create a new directory, its link count will start at 2
  - Hmm… Why?
- Each directory has `.` and `..` links
  - `.` – links to the directory itself (`./blah`)
  - `..` – links to the parent dir
- So the hard link count of 2 comes from the `.` and the directory file name

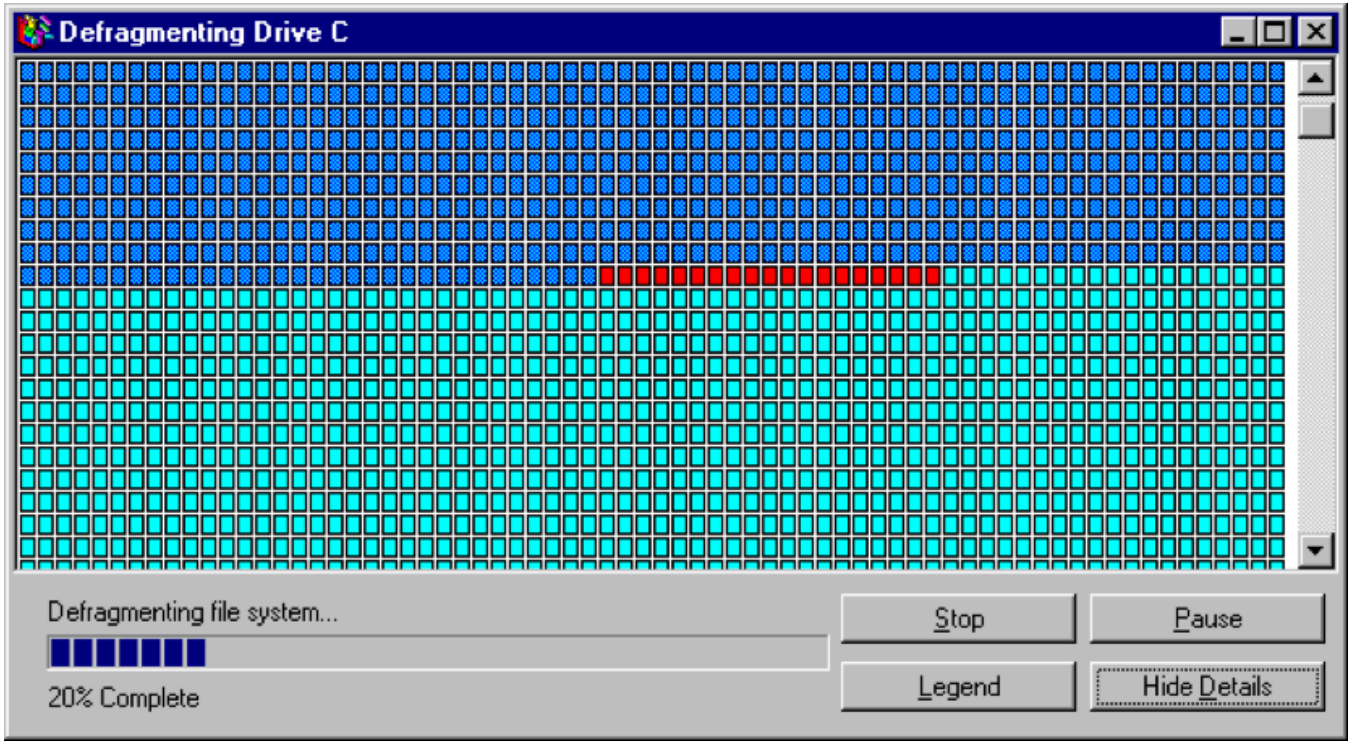
Link Count [3/3]

- When we delete directories with `rm -r`, these entries will be cleaned up as well
- When a file or directory link count reaches 0, the file is effectively deleted
- Behind the scenes, deleting is actually an ‘unlink’ operation
  - If data isn’t linked, we can safely write over it
  - Much like memory allocation…

ls -l


Total Blocks for this directory
[silicon:~/326/hw6-malensek]$ ls -l
total 44
-rw------- 1 matthew staff 130 Apr 23 16:43 Makefile
-rw------- 1 matthew staff 1375 Apr 23 16:43 README.md
drwx------ 2 matthew staff   64 Apr 25 15:05 a_dir
-rwx------ 1 matthew staff 9288 Apr 25 15:05 client
-rw------- 1 matthew staff 2453 Apr 23 17:39 client.c
-rw------- 1 matthew staff 356 Apr 23 17:30 common.h
-rwx------ 1 matthew staff 9136 Apr 25 15:05 server
-rw-r--r-- 1 matthew staff 2854 Apr 23 17:43 server.c
Size (bytes)
Last Modification
Link Count
Permissions
(User, Grp, Others)
Owner
Group

Fragmentation [1/3]



Fragmentation [2/3]

- We also mentioned that fragmentation can be a problem, especially with old file systems
  - You'll need to use a **disk defragmenter** to move files around and eliminate gaps
- Most modern file systems have other ways of dealing with this, but it's always going to be (somewhat) a problem

Fragmentation [3/3]

- To avoid fragmentation, we can split the disk into many small **blocks**
  - Next, we'll pre-allocate larger **extents** made up of several blocks for the files to grow into
    - This reduces the chances that tons of tiny blocks will be interleaved
- Another approach: defragment small files when you open them
  - HFS+ on macOS does this

The Superblock

- Disks are split into many small blocks
- We keep track of where all these blocks are with the **superblock**
- The superblock contains master file system information
  - Block extents, locations, file system info
- If we lose the superblock, our FS may be unrecoverable
  - Many FS incorporate superblock backups or split superblock info up into smaller pieces distributed across the disk

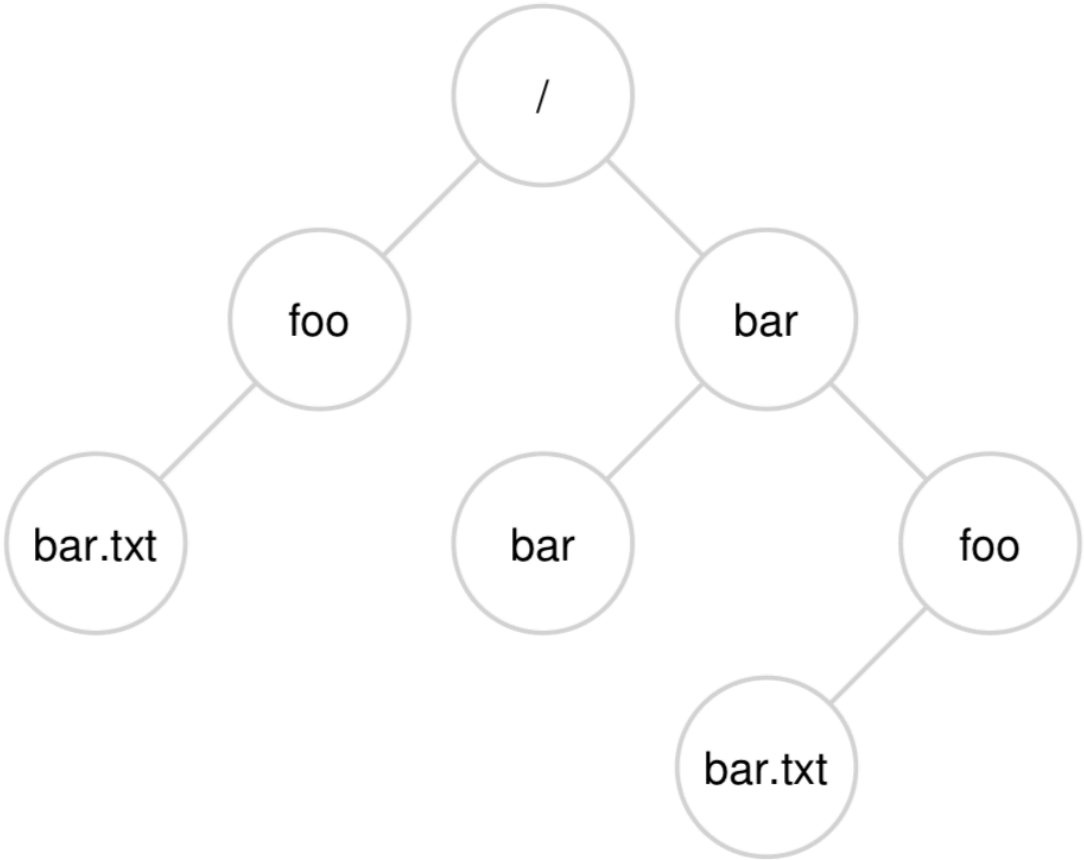
Today's Agenda

- File System Basics
- **Mount Points**

Disk Structure

- All Unix directory hierarchies start with /
  - The **root** directory
- We organize our files into a **directory structure**
  - a.k.a. folders
- Unix systems generally include some default directories such as:
  - /bin – binary utilities
  - /var – place for OS to write files during operation
  - /etc – configuration
  - /home – user home directories

A Sample File System Tree



Dealing with Multiple Disks

- If we have more than one disk, could we have two root directories?
  - **No!**
- Instead, we **mount** other disks underneath our root file system
  - For example, his allows me to mount another file system under `/home`
- After mounting, files inside `/home` are are on another disk

Drive Letters

- Windows (and DOS, and OS/2 😯) takes another approach: each disk is given a **drive letter**
  - In early OS, you'd access a file like this `A:FILE.TXT`
    - No directory support
    - Many folks had two floppy drives, `A:` and `B:`
- This abstraction is easier to understand, but tends to be less powerful
  - If we want to move program data to a separate hard drive on Unix-based systems, we can usually just copy it over and mount the new drive in the same location
  - On Windows, we will probably have to edit the **registry**
    - ew
- Note: Windows **can** mount drives under folders just like Unix, but seeing it done in production is less common

Mounting a Disk

- The mount command works as follows:
  - `mount -t fs_type /dev/sdb /some/mount/point`
- Generally, we don’t have to specify the file system type; just the device and mount point
- Let’s create a “disk” and experiment with this a bit…