ECE 598 – Advanced Operating Systems
Lecture 16

Vince Weaver

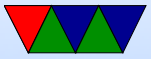
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 March 2018

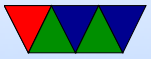
Announcements

- Project topics were due.
- Update on the problem with HW#7 (icache)
- Homework #8 will be posted
  Filesystems, should have it out sooner than #7
  Next will Be graphics, then Multi-core, then probably security.



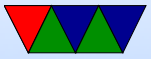
Filesystems

- Often a MBR (master boot record) and partition table
- Disks divided into partitions
  - Why partitions?
  - Split up system (/, /boot, /usr, /home)
  - Why is boot separate? Smaller so boot loader can access, maybe a different fs type.
  - Dual-booting operating systems
  - Swap partitions
- Then individual filesystems



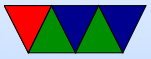
Filesystems – Organization

- A header containing master info (often called the superblock)
- Some sort of free list, saying what areas are free (bitmap or pointers)
- inodes, an array of data structures containing master info for each file (and if file is small, contents of file)
- Directory info: root directory entry, directory layout
- Actual file data



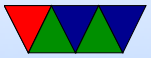
Filesystems – System Calls

- "mount" to put it in the proper place,
- "statfs" gives info on filesystem
  (including disk space, `df`)

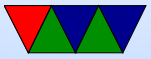


File Layout

- Contiguous.
  - Files in consecutive blocks.
  - Simple. Fast to read (just read X blocks)
  - Has fragmentation problems like with memory alloc.
  - What happens if append to file?
  - Ever used? read-only, CD-ROMs
- Linked list.
  - Inode points to first part, each block points to next.
  - No fragmentation, seeking through file involves lots of

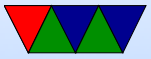


reads.
  - Waste part of block size for next pointer
- File Allocation Table
  - Like linked list, but the links are stored in a separate area on disk
  - Can also instead have the pointers in one single block, each pointing to next block.
  - Whole thing has to be in mem at once. Makes it faster (no need to do lots of disk reads on seek) but problem if structure is big
- Inode table



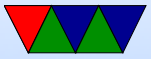
- Special structure named inode that holds file attributes and list of blocks.
- If need more blocks then fit, last one points to another block with more.
- Only has to be in memory if file is open

- Database
  - Treat disk as if it were a database, with the files the info you want to retrieve



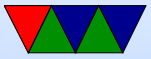
Directory Layout

- Directory table, holding the file name and location of the first block/inode
- Where to store attributes?
  - In the directory entry (FAT)
  - In the inode
- How big can a filename be?
  - Fixed (small) like FAT
  - Fixed (large)
  - Dynamically sized, meaning directory entry size is



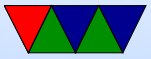
variable

- Searching for filenames
  - Linear search
  - Hash

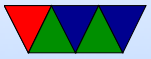


Shared Files (Linking)

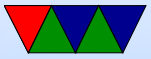
- What if you want to share files? Two names for same file?
- Symbolic Link – just a simple pointer
  - Downside, have to traverse link
  - if original file removed end up with broken links
  - Takes more disk space (extra inode/file)
- Hard Link – different directory entries can point to same inode
  - This is why filename is in dir entry but everything else



in inode on UNIX/Linux
  - Trouble: what if create circular links in filesystem?
  - What if backing up filesystem or doing a search, can see same file multiple times
- This is why delete on Linux is called "unlink"
  - You are just decrementing the link count.
  - What happens when link count goes to zero?
  - What happens if you have a file open and it gets unlinked to zero?
  - What happens if running an executable and it gets deleted (maybe as part of a system upgrade?)

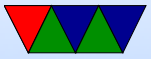


- Why might you do that on purpose?

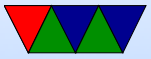


Disk performance

- Traditionally a lot of this came down to hardware.
- Spinning rust disks; head movement, cylinders/sectors. Reading consecutive faster, random access bad (millisecond bad)
  More complicated, fancy disk interfaces and embedded processors. Large caches (why can that be bad), shingled disks?
- Much of this goes away with flash disks, but still emulate old disk interface

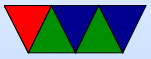


- Name lookup can also be slow.



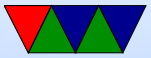
Disk Block Size

- Way too much overhead to have single byte granularity
- For a long time this was 512bytes/block
  Way too many for huge disks so disk drive companies pushing for 4k (but trying to remain backward compatible)
- Filesystems can allocate with larger blocksize.
- Large blocksize good: fewer blocks to track for each file
- Large blocksize bad: waste space on small files



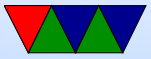
Disk Quotas

- Don't hear about them much any more, but on a shared system constantly running out of disk space.
- When undergrad, shared mail server, 7MB of disk quota
- Structure on disk with quota limits, checks on file access to make sure not go over.

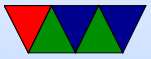


Reliability

- What happens when the power goes out?
- Your system caches disk info (writes are slow) and only writes them out every few seconds (best case)
  - This is why you should unmount disks before ejecting or pulling drive out
  - You can also use "sync" command to force to disk
  - Problem is, disks have own caches (and they lie, why?)
- What happens to the data that didn't get written to disk?



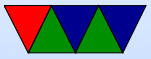
- What happens on next startup?
  - Traditionally, a tool called fsck (filesystem-check) would run and try to fix things. Find inodes without matching direntries and try to bring them back, etc.
  - fsck really slow, especially for large disks (hours?)
  - also could go very wrong. (disk images on disk?)
- Journaling Filesystem
  - A write that changes things (say remove a directory) Removes dir, releases inodes, frees up blocks. What if interrupted in there? Inconsistent, things not freed.
  - Store writes to metadata separately in a circular list.



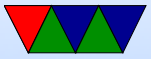
Then goes and done things.

- Make sure metadata written to disk, only then update disk
- Worst case you might lose some data written, but actual fs structure should be intact. Circular buffer.
- On crash plays back all it has in journal at boot

- Are disks perfect? No. Newer filesystems can store checksums and the like
- Backups! Are you backing things up? Can you backup a filesystem while it's being used? Can be tricky depending how it is designed.

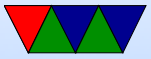


- Snapshots – we'll talk about this later
- Compression
- Encryption
- Defragmenting



Writing a Filesystem

- Linux VFS. TODO
- FUSE (userspace)



Common Filesystems

- Windows: NTFS, FAT
- UNIX/Linux: EXT4, BTRFS, ZFS
- OSX: HFS+, APFS (Apple Filesystem)
- Media: ISO9660, UDF
- Network: NFS, CIFS