ECE 531 – Advanced Operating Systems
Lecture 18

Vince Weaver

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

vincent.weaver@maine.edu

7 November 2023

Announcements

- Project topics were due, I sent out e-mails

Directories Layout

- Directory entry stores a list of files in the directory
- Some of these file entries can be directory entries themselves (special flag set) that lets you know it's a directory

Directories – Filename Size

- How big can a filename be?
  - Fixed (small) like FAT
  - Fixed (large)
  - Dynamically sized, meaning directory entry size is variable

Directories Filename Lookup

- Linear search
- Some sort of tree
- Hash

Directories – Where to store metadata?

- Where to store attributes?
  - In the directory entry (FAT)
  - In the inode

Shared Files (Linking)

- What if you want to share files? Two names for same file?
- Symbolic Link – just a simple pointer
  - Can link to files on other filesystems
  - 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

Deleting Files

- 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 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 file contents
- 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

Backups

- 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 on Linux

- Linux VFS interface. Will discuss later
- FUSE (userspace)

Common Filesystems

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

FAT Filesystem

- Originally introduced for small floppy disks in late 70s/early 80s
- Fat-8 (obsolete), FAT-12/FAT-16/FAT-32
  The number is number of bits used for cluster number.
- Benefits: mostly simplicity, widely used and supported

FAT Filesystem Overview

- Primary data structure is the File Allocation Table (FAT) that is used to find the file's content on disk
- Disk is divided up into chunks called clusters
- Cluster size can vary from 512 - 32kB (tradeoffs)
- Root directory – contains cluster start for each file in directory

Reserved Sectors

- Some sectors at start are reserved. FAT32 12 are reserved
- Boot Sector (Sector 0)
  - Includes BIOS Parameter Block (BPB) which has info on the filesystem, pointers to location of other sections
  - Also includes initial bootloader code
- FAT32: File system information sector (sector 1)
- FAT32: bootloader can be spread to sector 2 also. Also backup boot sector at 6, 7, 8 and maybe extra code at 12

Boot Block

512 bytes, first part configuration info (block size, blocks in disk, FATs, etc), rest actual boot loader code

Offset	Len	Description
0x00	3	bootstrap (JMP insn to code start)
0x03	8	manufacturer/OEM name
0x0b	2	bytes per block (start of BPB)
0x0d	1	blocks per unit (sectors per cluster)
0x0e	2	reserved blocks (usu. 1 for boot)
0x10	1	number of FATs
0x11	2	total root dir entries
0x13	2	blocks per disk. if $> 2^{16}$ see 0x20
0x15	1	media descriptor
0x16	2	FAT size (blocks)
0x18	2	blocks per track

Offset	Len	Description
0x1a	2	disk heads
0x1c	4	hidden blocks (usually 0)
0x20	4	blocks on entire disk
0x24	2	drive num
0x26	1	boot signature
0x27	4	volume serial number
0x2b	11	volume label
0x36	8	fs id
0x3e	0x1c0	rest of boot code
0x1fe	2	0x55aa (end of boot block)

File Allocation Table (FAT)

- One or more copies of File Allocation Table (FAT). Why multiple copies?
- Has to fit entirely in RAM.
- Just a table of values, one for each cluster pointing to the next cluster in the file.
  - FAT12 these were 12-bits (two spread across 3 bytes)
  - FAT16 16-bits
  - FAT32 32-bits (actual 28 with top 4 bits reserved)
- Entry 0 and 1 are reserved.

- ◦ 0 holds FAT id (0xfff0 - 0xffff)
  will end chain if try to follow an empty (0) cluster
- ◦ 1 holds the end-of-chain marker (usually 0xffff) The last entry in a list is 0xffff
  Some bits cleared/set and start stop, used to indicate if shutdown cleanly

- Entry values
  - ◦ 0 means unused
  - ◦ 1 reserved
  - ◦ 0xfff7 might mean bad cluster.
  - ◦ 0xffff is end of chain

FAT Example

Example, a file might start at 2:

offset	value
0	//////
1	//////
2	3
3	5
4	0
5	ffff
. . .	
N	0

Root Directory

- On FAT12/16 area allocated and format time, so limited room (FAT32 lives in data area)
- How do we know where a file starts?
  Root directory entry follows after last FAT.
- Values are little endian

offset	size	description
0x00	8	filename
0x08	3	extension
0x0b	1	attributes
0x0c	10	reserved
0x16	2	creation/update time (h/m/s) second must be even
0x17	2	creation/update date (see further slide)
0x1a	2	start cluster
0x1c	4	filesize (bytes)

Filename+Extension

- Filename
  - ◦ First byte 0x0 = file slot never used before
  - ◦ First byte 0xe5 = file deleted (sigma) (how can you undelete? restore first char, then hope the file was contiguous and restore as many clusters as the filesize says. later DOS deleted char stored in ???)
  - ◦ first byte 0x05 = first char actually 0xe5
  - ◦ 0x2e '.' this is current directory
  - ◦ If another 0x2e '.' then cluster field is parent directory

(..) 0x00 means root

- ◦ If not 8 chars, padded with spaces
- ◦ By default, only capital letters, numbers. Excludes some punctuation.

- Extension
  - ◦ three bytes. dot is assumed

Attributes

- Attributes
  - 0x1=r/o
  - 0x2=hidden
  - 0x4=system
  - 0x8=disklabel
  - 0x10 subdirectory
  - 0x20=archive (for backups)
- Time: hhhhhmmmmmmsssss. seconds has to be even
- Date yyyyyyymmmmddddd y = 0-199 (1980-2099)

Directories

- Directories: if attribute set, then cluster chain treated as a series of directory entries

Undeleting

- Have to remember first char of file (later DOS stored this somewhere)
- Deleted file entry still has start cluster. Have to hope none of the clusters have been reused
- To help, later DOS did last-fit and kept allocation pointer to try to avoid reusing clusters right away

Long Filenames

- UMSDOS – Linux hack that had a –linux.— file in each dir that held permissions, etc.
- VFAT – Windows95 solution
  - ○ A dummy file entry put beforehand to hold long name Has attributes VOLUME SYSTEM HIDDEN READONLY (0xf) which old will ignore
  - ○ Up to 13 UCS-2 (unicode) characters per entry, up to 20 of them can be chained (for up to 255 char long filenames)

- Also a compatible one is created. Something like "HelloWorld.jpg" might be "HELLOW$\sim$1.JPG"
- Newer VFAT also re-used some reserved bytes in dir entry to extend creation time to have ms resolution.

Newer FAT

- Fat32 – allow larger files and filesystems. Larger directories. Lots of changes besides just making FAT twice as large. Still limited to 4GB-1 filesize
- exFAT. Designed for use in digital cameras. more than 4GB filesize and 32GB or so disk size. also many other improvements, not backwards compatible before windows XP.

FAT on Linux

- Linux uses inodes for file access. How can you mount a FAT filesystem then?
- Really, inode just has to be a unique identifier for a file that can be used to find the start of the file info on disk. So you can use the cluster number or similar.