ECE 531 – Advanced Operating Systems
Lecture 30

Vince Weaver

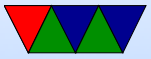
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

17 November 2025

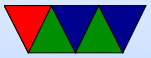
Announcements

- Homework #7 was posted
  - Had some issues, I will be posting an updated version



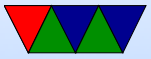
Reliability – What if Power Goes Out?

- Anything queued up to be written to disk can be lost.
- Worse than lost data, what if it's metadata. Can whole filesystem end up corrupted?
- This is also why you should unmount disks before ejecting or pulling cable out



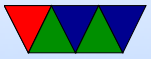
Why Writes get Lost

- Writing to disk is slow and can take a while (milliseconds)
- It is better for performance to group writes together (instead of trickling them out one byte at a time) but that means having a buffer
- If the power goes out before the buffer in memory gets to disk, problems
- Even worse, the hard-disk might have a memory cache too that also might be lost



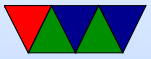
Can you force writes to disk?

- Maybe even have the OS flush buffers every few seconds?
- Doing a “sync” or “triple-sync” to force it?
- Can be bad for performance / power saving (old days idle drives could spin down)
- Even worse, disks have their own caches, and they might lie when you ask if the data made it to disk (why? benchmarks...)



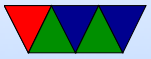
After a Crash

- 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?)
  - Could sometimes have file contents but not recover names. These end up in "lost+found" directory

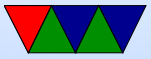


Journaling Filesystem

- 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.

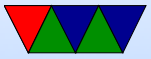


- On crash plays back all it has in journal at boot

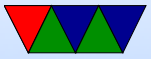


Backups

- 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

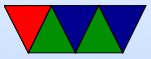


Other Filesystem Features



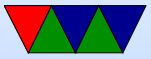
Compression

- Disk space valuable, can we fit more with compression?
- Downside is it can be slow to decompress
- Also harder to manage because different files have different compression ratios
- What happens when write a byte to middle of compressed block and now it's too big to fit and you have to recompress whole file
- Related: de-duplication, if same file multiple times remove copies and just point to the one copy



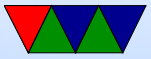
Sparse Files / Holes

- What if your file has lots of zeros?
- What if you seek way into a file (to write something at end)
- Do you need to allocate zeros on disk for these?
- Many filesystems support holes, where the inode list says a file has a zero, only allocates disk block if you write in this range
- Can save a lot of disk space



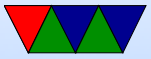
Deleting Files

- Is deleting a file permanent?
- Should you store info to make Undelete possible?
- Secure Delete – is just overwriting once fine? Multiple over-writes different patterns, power-drill and thermite?
- Old spinning-rust disks if you opened it up and had fancy tools could often see previous writes because write head never really lined up
- Modern flash drives, due to wear leveling old blocks might be left behind



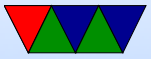
Encryption

- Can encrypt disks
- Secure
- Bad if you forget encryption key

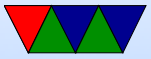


More Features

- Online fsck – constantly check for errors
- Defragmentation – in old days could improve performance by having files contiguous on disk (needed on modern fses?)
- Quotas – especially an issue on multi-user machines, you want to keep any one user from filling up the disk.
- Locking – may want to prevent more than one person writing a file at a time as it can get corrupted
- Checksums – can tell if file got corrupted on disk

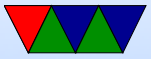


- Resizing – can you resize a filesystem to be bigger/smaller (why would you want to?)



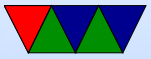
Common Filesystems

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



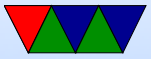
FAT Filesystem

- `https://wiki.osdev.org/FAT`
- Originally introduced for small floppy disks in late 70s/early 80s
- Fat-8 (obsolete), FAT-12/FAT-16/FAT-32
- Disk is broken up into chunks called “clusters”, the number after FAT- is number of bits used to identify a specific cluster.
- Benefits: mostly simplicity, widely used and supported



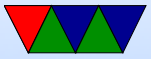
Bytes / Sectors

- The underlying disk traditionally has its own blocksize
  - Sometimes this is called a sector, dating back to floppy/hard disks with spinning discs, and a block was part of a circular track (sector)
  - Usually underlying block / sector was 512 bytes but not necessarily
  - On modern systems might be 4k. Some weird things like CD-ROMS (2336 bytes)



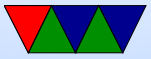
Clusters

- For FAT purposes disk broken up into chunks called clusters, which can be multiple contiguous disk blocks
- In order to handle disks as they got larger without changing the FAT format the clustersize would grow, from 512-bytes to 32k or more
- This led to a tradeoff, could have bigger disks, but possibly waste lots of empty space if files less than 32k (or if the last cluster not completely filled)



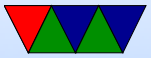
Chains

- A file that is bigger than a cluster can be found via a chain of clusters that don't have to be contiguous



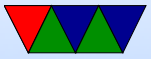
Overall Format

offset (bytes)	description
0	Boot Block
512	FAT #1 FAT #N
...	Root Directory
... ...	Data Blocks



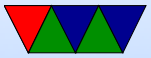
Reserved Sectors

- Some sectors are reserved at format time
- On 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 additions:
  - File system information sector (sector 1)
  - Bootloader can be spread to sector 2 also.
  - Also backup boot sector at 6, 7, 8 and maybe extra code at 12

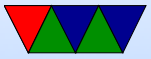


Boot Sector Details

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

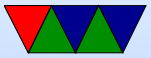
Offset	Len	Description
0x1a	2	disk heads
0x1c	4	hidden blocks (usually 0)
0x20	4	blocks on entire disk
0x24	2	drive num
0x26	1	boot signature
0x27	4	volume serial number
0x2b	11	volume label
0x36	8	fs id
0x3e	0x1c0	rest of boot code
0x1fe	2	0x55aa (end of boot block)



Directory Entries – Store metadata on File

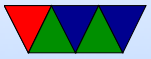
Note: values are little endian

offset	size (bytes)	description
0x00	8	filename
0x08	3	extension
0x0b	1	attributes
0x0c	1	(FAT32) uppercase
0x0d	1	(FAT32) extra bits for timestamp
0x0e	2	(optional) creation time
0x10	2	(optional) creation date
0x12	2	(optional) last access date
0x14	2	(FAT32) upper half of start cluster
0x16	2	last change time
0x18	2	last update date
0x1a	2	start cluster
0x1c	4	filesize (bytes)



Filename+Extension

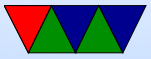
- Filename
  - First byte 0x0 = file slot never used before
  - First byte 0xe5 = file deleted (sigma) (how can you undelete? restore first char, then hope the file was contiguous and restore as many clusters as the filesize says. later DOS deleted char stored in ???)
  - first byte 0x05 = first char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



(..) 0x00 means root

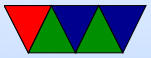
- If not 8 chars, padded with spaces
- By default, only capital letters, numbers. Excludes some punctuation.

- Extension
  - three bytes. dot is assumed



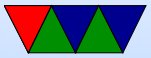
FAT Attributes

- 0x01 – read-only
- 0x02 – hidden
- 0x04 – system
- 0x08 – disklabel
- 0x10 – subdirectory
- 0x20 – archive (for backups)
- 0x0f – long file name



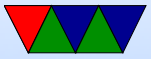
FAT Timestamps

- Time: hhhhhmmm.mmmssssss
  h=0..31, m=0..63, s=0..31 (seconds has to be even)
- Date yyyyyyym.mmmddddd
  y=0..127 (1980-2099 valid)
  m=0..15 (1=Jan), d=0..31 (1=first)
- FAT32 you can get extra bits to get full seconds and milliseconds



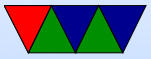
Directories

- If sub-dir attribute set, then cluster chain treated as a series of directory entries
- Some of them can be files, and some can be subdirectories too



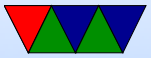
Root Directory

- On FAT12/16 area allocated and format time, so limited room (FAT32 lives in data area). Special, and has no '.' or '..'
- On FAT32 it's a more like a standard subdirectory
- How do we know where a file starts?
  Root directory entry follows after last FAT.



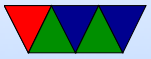
File Allocation Table (FAT)

- Data structure that stores linked-list of clusters used by files
- Instead of living with the file metadata, instead global datastructure that all share
- Was easy to parse/maintain on old low-RAM systems and also avoided jumping all over disk when following linked list
- There may be multiple copies (why?)



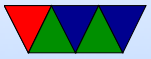
File Allocation Table Details

- 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)

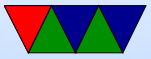


FAT Entry Meanings

- 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 to indicate if shutdown cleanly
- Entry contents
  - 0 means unused
  - 1 reserved

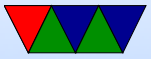


- 0xfff7 might mean bad cluster.
- 0xffff is end of chain



FAT Size Questions

- What is the maximum sized disk you can have?
  FAT32 can have 2^28 clusters. Max cluster size is 32k w/o hacks. This would imply 8TB.
  Microsoft usually limited it to 32GB
- What is the maximum sized file you can have?
  The filesize in the directory entry is 32-bits which means max is 4GB

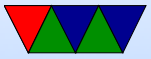


FAT Example

Example FAT:

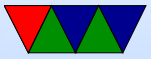
offset	value
0	//////
1	//////
2	3
3	5
4	0
5	ffff
. . .	
N	0

- Say the directory entry for a file points to "cluster 2" for the start of the file
- If cluster size is 4k, it means the first 4k of file are found in the data part of the filesystem, at disk block

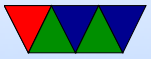


$data\_start + (2 * clustersize)$

- If you are reading past the first cluster, you need to check the FAT to find the next cluster. Look up entry #2 in the FAT and see it points to cluster 3. So the next 4k are found on cluster 3
- If you go beyond that, look in FAT entry #3 and see it points to cluster 5. So look there
- If you try to read past that, look in FAT for entry #5 which is 0xffff. This means end of file, there are no more clusters
- Note that you don't have to use up the whole cluster

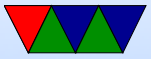


so you should also pay attention to the filesize in the directory entry and ignore any data past the end



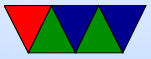
Undeleting

- 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

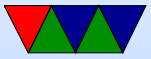


Long Filenames

- 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)

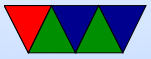


- Also a compatible one is created. Something like "HelloWorld.jpg" might be "HELLOW~1.JPG"
- Newer VFAT also re-used some reserved bytes in dir entry to extend creation time to have ms resolution.



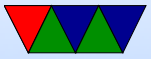
Newer FAT – Fat32

- Fat32 – allow larger files and filesystems. Larger directories. Lots of changes besides just making FAT twice as large. Still limited to 4GB-1 filesize

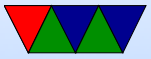


Newer FAT – exFAT

- Designed for use in digital cameras
- All FAT32 patents had expired so rumors this was MS way of extending things
- For while Linux driver was questionable leaked one, but now there's an approved on
- Allows more than 4GB filesize and 32GB of FAT32. 512TB - 128PB. Max filesize 128PB
- Filenames all unicode but not slashes, : * ? " angle brackets, pipe



- Hash-based lookups. Convert to uppercase and down to 16-bit hash before lookup
- Checksums?



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.