ECE 598 – Advanced Operating Systems
Lecture 17

Vince Weaver

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

vincent.weaver@maine.edu

3 April 2018

Announcements

- Project Topics
  - Should have gotten response on project topic
  - When creating your own userspace progrma, if you get a “GOT” error, the first thing to try is make all your global variables static.
  - If you’re adding binaries like sound files or disk images it’s fairly straightforward, let me know if you do have issues.

Homework #6 Review

- Not really much to say.
- Scheduler is round-robin. Just simple linked lists.
- Memory allocator was find-first.
- Making it find next was just a matter of storing the last found and starting from there. Plus making sure you wrap to zero if you get to the end, as well as exit if go whole way around without finding any.
- Hard to test the memory stuff without coming up with a fairly intense memory workload and limiting max memory

to something small.

- As said in midterm review, without virtual memory it's really hard (near impossible) to rearrange memory being used by C because it's not always possible to know where all the pointers are.

Homework #7 Review

- Speeds
  8-bit 5MB/s, 32-bit 23MB/s, Custom 46MB/s
  8-bit 204MB/s, 32-bit 858MB/s, Custom 1073MB/s

- Custom
  - Loop unrolling
  - in-line asm
  - vector instructions
  - x86: string instructions

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

- File Allocation Table – traverse a list of clusters
  Look up cluster number in table, shows next cluster (or end of file marker, or unused indicator)
- Various cluster sizes from 512 - 32kB (tradeoffs)
- Root directory – contains cluster start for each file in directory

Reserved Sectors

- Boot Sector (Sector 0)
  - Includes BIOS Parameter Block (BPB) which says where everything else is

Boot Block

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

Offset	Length	Description
0x00	3	bootstrap (jmp to later)
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 block)
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
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?
- Actually has to fit entirely in RAM.
- Just a table of 16-bit values, one for each cluster pointing to the next cluster in the file.
- 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.
- Size of entry
  - 12=fat12 (3 bytes hold 2 cluster)
  - 16 fat16
  - 32 fat 32

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 is 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.

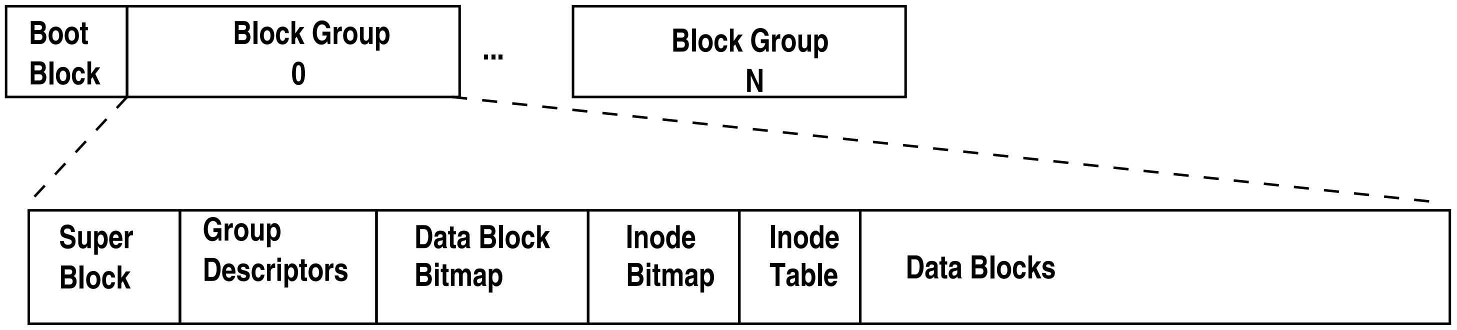
Ext2 FS

- From the early 1990s when minixfs and other options started being limiting
- Supports 4TB filesystem, 2GB file size
- All structures are little-endian (To aid in moving between machines)
- Block size 1k-4k (for various reasons it's complicated on Linux to have a block size greater than the page size) (also, does blocksize have to be power of 2? Some CD-ROMs had blocksize of 2336 bytes)

- 5% of blocks reserved for root. Why? Is that needed anymore?

Overall Layout

- Boot sector, boot block 1, boot block 2, boot block 3



- Block group: superblock, fs descriptor, block bitmap, inode bitmap, inode table, data blocks

Block Group

- A bitmap for free/allocated blocks
- A bitmap of allocated inodes
- An inode table
- Possibly a backup of the superblock or block descriptor table
- Effort is made to make files be allocated in same block group as their dir entry.

Superblock

- Superblock – located at offset 1024 bytes, 1024 bytes long Copies scattered throughout (fewer in later versions) Info on all the inode groups, block groups, etc.

Offset	Size	Description
0	4	Number of inodes in fs
4	8	Number of blocks in fs
8	4	Blocks reserved for root
12	4	Unallocated blocks
16	4	Unallocated inodes
20	4	block num of superblock
24	4	block size shift
28	4	fragment size shift
32	4	blocks in each group
36	4	fragments in each group
40	4	inodes per group
44	4	last mount time
48	4	last write time
52	2	mounts since last fsck
54	2	mounts between fsck
56	2	ext signature (0xef53)
58	2	fs status (dirty or clean)
60	2	what to do on error
62	2	minor version num
64	4	time of last fsck
68	4	interval between fsck
72	4	OS of creator
76	4	major version number
80	2	uid that can use reserved blocks
82	2	gid that can use reserved blocks
84	4	first non-reserved inode
88	2	size of each inode

Block Group Descriptor Table

- Follows right after superblock

offset	size	Description
0	4	address of block usage bitmap
4	4	address of inode usage bitmap
8	4	address of inode table
12	2	number of unallocated blocks in group
14	2	number of unallocated inodes in group
16	2	number of directories in group

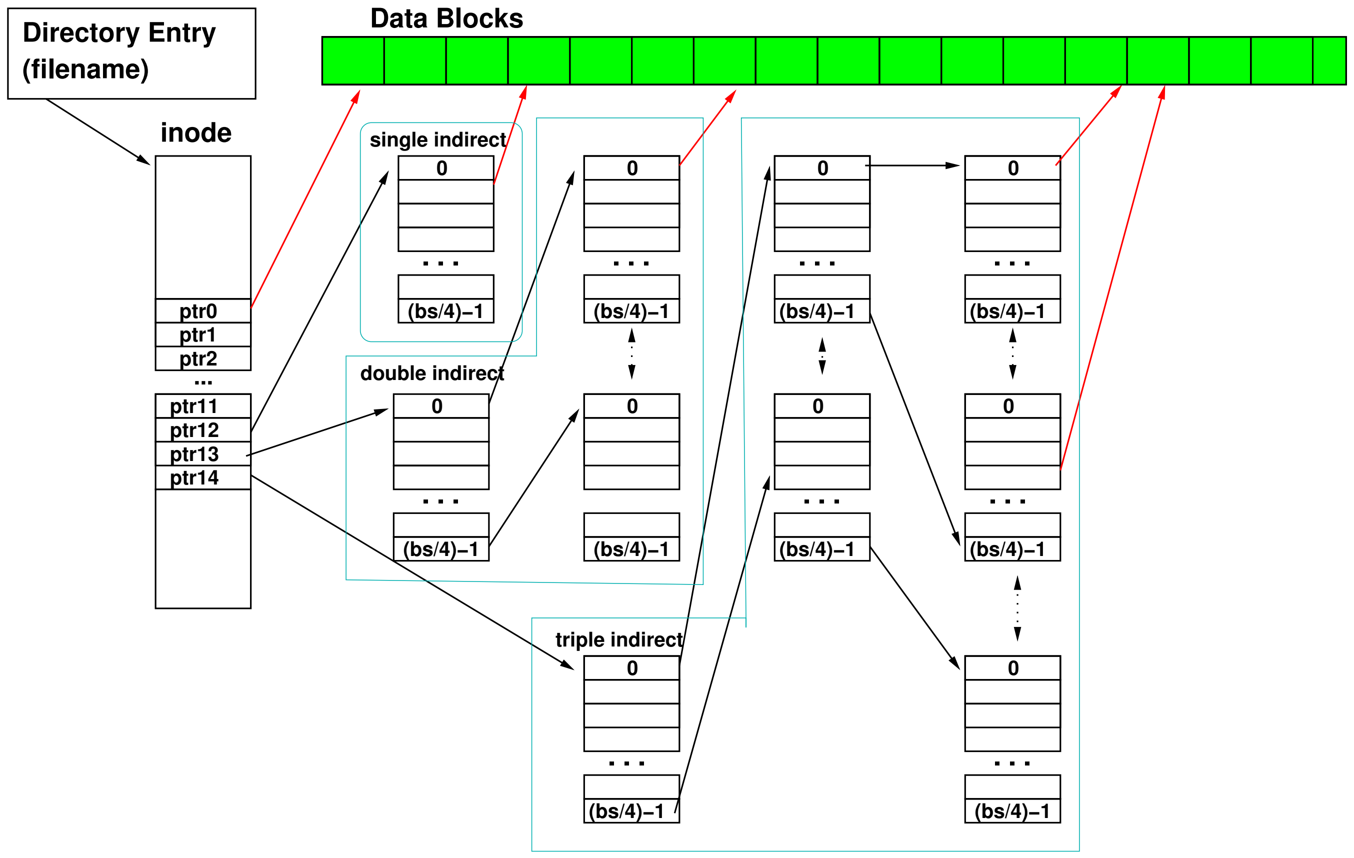
Block Tables

- Block bitmap – bitmap of blocks (1 used, 0 available) block group size based on bits in a bitmap. if 4kb, then 32k blocks = 128MB.

Inode Tables

- Inode bitmap – bitmap of available inodes
- Inode table – all metadata (except filename) for file stored in inode
  Second entry in inode table points to root directory
  inode entries are 128 bytes.

offset	size	desc
0	2	type and permissions
2	2	userid
4	4	lower 32 bits of size
8	4	last access time (atime)
12	4	creation time (ctime)
16	4	modification time (mtime)
20	4	deletion time
24	2	group id
26	2	count of hard links
28	4	disk sectors used by file?
32	4	flags
36	4	os specific
40 - 84		direct pointers 0 - 11
88	4	single indirect pointer
92	4	double indirect pointer
96	4	triple indirect pointer
100	4	generation number (NFS)
104	4	extended ACL
108	4	ACL (directory) else top of filesize
112	4	address of fragment


Directory Entry (filename)
Data Blocks
inode
ptr0
ptr1
ptr2
...
ptr11
ptr12
ptr13
ptr14
single indirect
double indirect
triple indirect
0
. . .
(bs/4)–1

Directory Info

- Superblock links to root directory, (usually inode 2)
- Directory inode has info/permissions/etc just like a file
- The block pointers point to blocks with directory info.
- Initial implementation was single linked list. ext3 and newer use hash or tree.
- Holds inode, and name (up to 256 chars). inode 0 means unused.

type	size
inode of file	4
size of entry	2
length of name	1
file type	1
file name	N

- Hard links – multiple directory entries can point to same inode
- `.` and `..` entries, point to inode of directory entry
- Subdirectory entries have name, and inode of directory

How to find a file

- Find root directory
- Iterate down subdirectories
- Get inode

How to read an inode

- Get blocksize, blocks per group, inodes per group, and starting address of first group from the superblock
- Determine which block group the inode belongs to
- Read the group descriptor for that block group
- Extract location of the inode table
- Determine index of inode in table
- Use the inode block pointers to read file

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

Ext3/Ext4

- Compatible with ext2
- ext3
  - Htree instead of linked list in directory search
  - online fs growth
  - Journal

    metadata and data written to journal before commit. Can be replayed in case of system crash.
- ext4
  - Filesize up to 1Exabyte, filesize 16TB

- Extents (Rather than blocks) , an extent can map up to 128MB of contiguous space in one entry
- Pre-allocate space, without having to fill with zeros (which is slow)
- Delayed allocation – only allocate space on flush, so data more likely to be contiguous
- Unlimited subdirectories (32k on ext3 and earlier)
- Checksums on journals
- Improved timestamps, nanosecond resolution, push beyond 2038 limit

Why use FAT over ext2?

- FAT simpler, easy to code
- FAT supported on all major OSes
- ext2 faster, more robust filename and permissions