# ECE 598 – Advanced Operating Systems
# Lecture 18

Vince Weaver

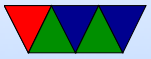
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

5 April 2016

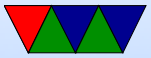
# Announcements

- Homework #7 was posted
- Project update



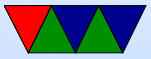
# Notes on HW7

- More like a 571 HW
- Enabling cache, etc
- Enabling virtual memory, but old legacy ARMv5 version. Found code online, have to figure out what doing. Why do you need to enable VM for L1 dcache?



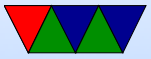
# Notes from last time

- undelete char is a sigma character
- when you delete a file, the FAT entries are zeroes out. How can you undelete? Deleted entry still has pointer to first data block. You have to *really* hope your file was not fragmented
- 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.



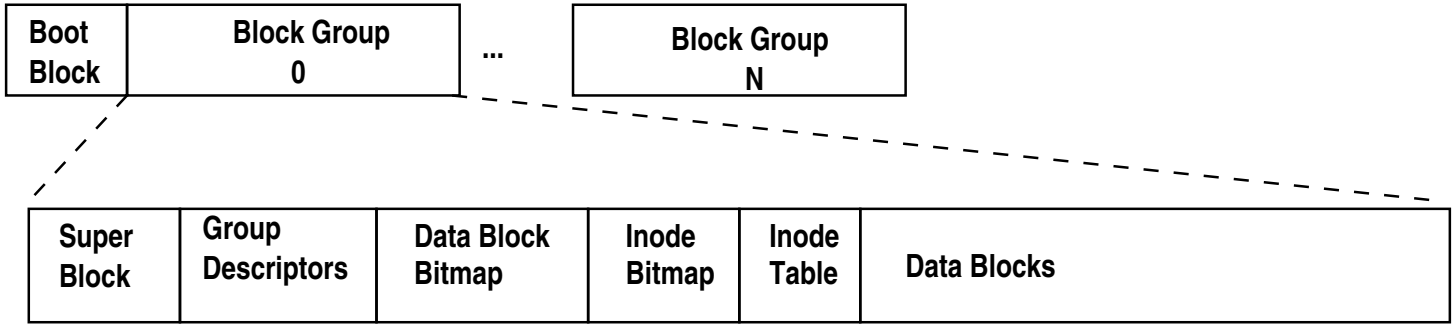
# Ext2 FS

- All structures are little-endian (To aid in moving between machines)
- Block size 1024-4096 (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)

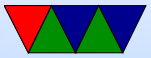


# Overall Layout

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

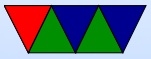



- Block group: superblock, fs descriptor, block bitmap, inode bitmap, inode table, data blocks



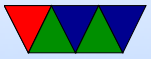
# Block Group

- A bitmap for free/allocated blocks
- A bitmap of allocated inodes
- An inode table
- Possibly a backup of the superblock or block descriptor table

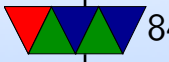


# Superblock

- 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.



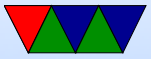
| Offset | Size | Description |
|---|---|---|
| 0 | 4 | Number of inodes in fs |
| 4 | 8 | Number of blocks in fs |
| 8 | 4 | Blocks reserved for root |
| 12 | 4 | Unallocated blocks |
| 16 | 4 | Unallocated inodes |
| 20 | 4 | block num of superblock |
| 24 | 4 | block size shift |
| 28 | 4 | fragment 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

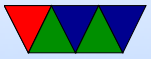
- Follows right after superblock

| 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

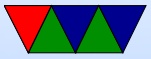
- Block bitmap – bitmap of blocks (1 used, 0 available) block group size based on bits in a bitmap. if 4kb, then 32k blocks = 128MB.



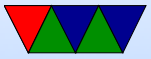
# Inode Tables

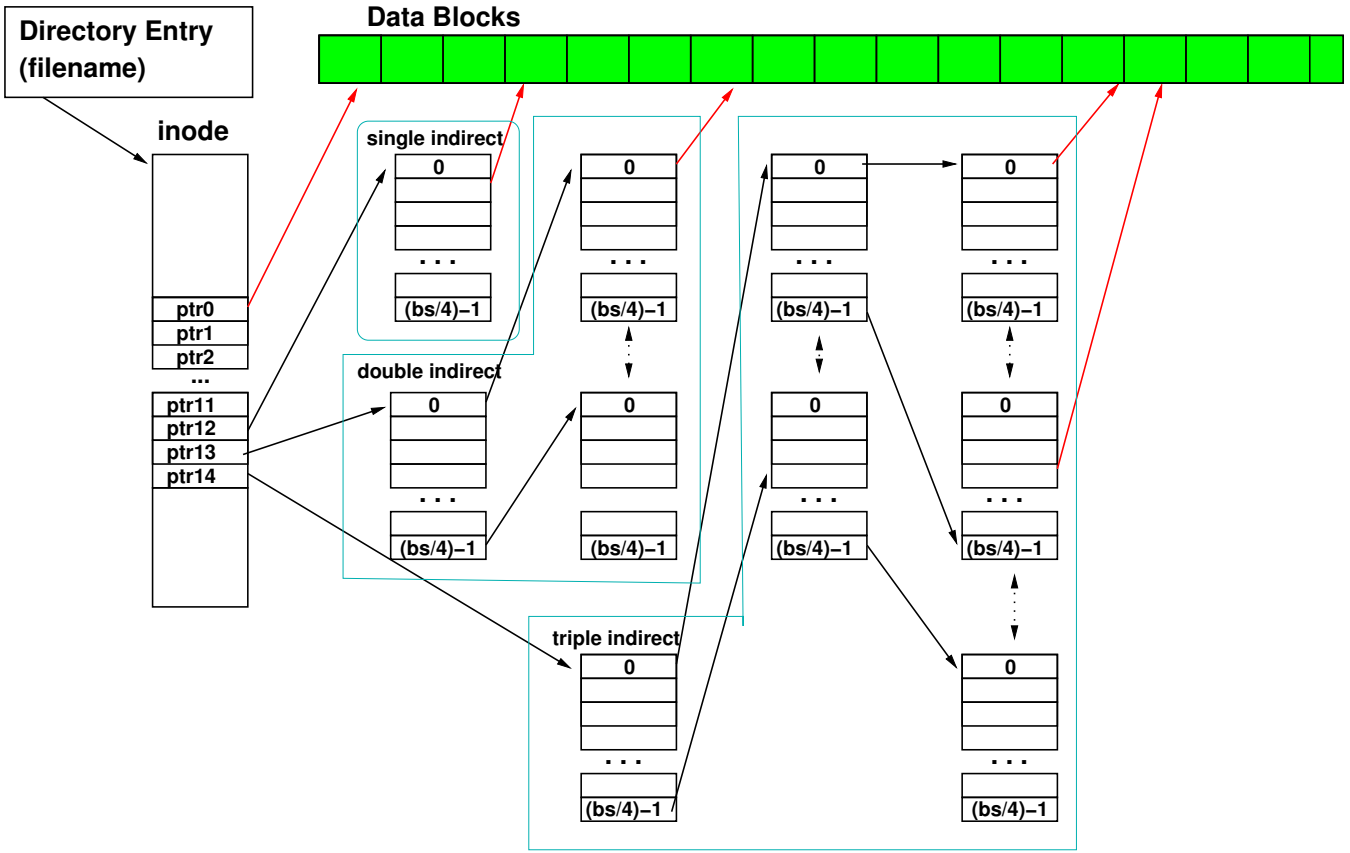
- Inode bitmap – bitmap of available inodes

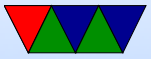
- Inode table – all metadata (except filename) for file stored in inode
  Second entry in inode table points to root directory
  inode entries are 128 bytes.



| 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 |

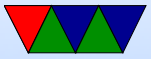



Directory Entry
(filename)
Data Blocks
inode
ptr0
ptr1
ptr2
...
ptr11
ptr12
ptr13
ptr14
single indirect
double indirect
triple indirect
0
(bs/4)–1



# Directory Info

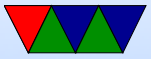
- Directory info –
  Superblock links to root directory (usually inode 2)
  Directory inode has info/permissions/etc just like a file
  The block pointers point to blocks with directory info.
  Initial implementation was single linked list. Newer use hash or tree.
  Holds inode, and name (up to 256 chars).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



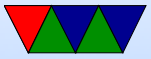
| type | size |
|---|---|
| inode of file | 4 |
| size of entry | 2 |
| length of name | 1 |
| file type | 1 |
| file name | N |

- Hard links – multiple directory entries can point to same inode

- . and .. entries, point to inode of directory entry

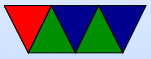


- Subdirectory entries have name, and inode of directory



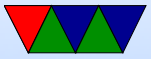
# How to find a file

- Find root directory
- Iterate down subdirectories
- Get inode

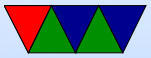


# How to read an inode

- 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



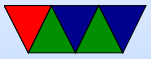
- Use the inode block pointers to read file



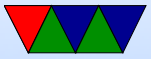
# Ext3/Ext4

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