# ECE 531 – Advanced Operating Systems
# Lecture 31

Vince Weaver

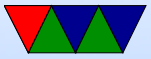
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

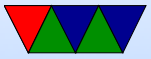
19 November 2025

# Announcements

- Don't forget HW#7
- Don't forget project status update due Friday
- There will eventually be a HW#8 (graphics, filesystems)

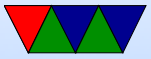


# EXT2 – A "Traditional" UNIX-style Indirect Filesystem



# Ext2 FS – History

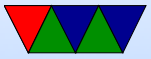
- Linux originally used “minixfs” but it had 16-bit offsets and max size limit of 64MB and filename of 14 chars
- Replacement: ext, but still limits
- Search for a replacement: ext2 (by Rémy Card) vs xiafs (by Ge Xia)
- ext2 won
- Later extended to ext3 (with journaling) and ext4
- Still issues with things like y2038 bug



# Ext2 On-disk Info

- Originally supports 4TB filesystem, 2GB file size
- All structures are little-endian

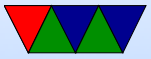
  Learned hard way not specifying, Atari disk images not work on x86



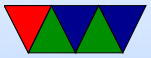
# Ext2 Block Sizes

- Block size 1k-4k
  for various reasons it's complicated on Linux to have a block size greater than the page size
- Does blocksize have to be power of 2?
  Some CD-ROMs had blocksize of 2336 bytes

- Hard-disks traditionally had sizes of 512 bytes (recent push to 4k) (flash is probably much higher but has to fake it for compatibility reasons)

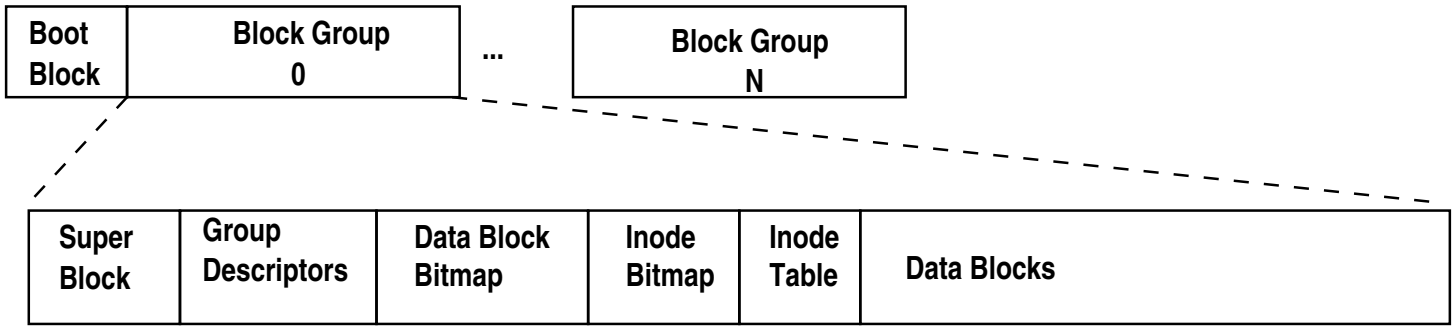


- 5% of blocks reserved for root. Why? Still needed?

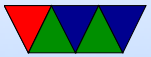


# Overall Layout

- Low-level blocks, grouped together in block groups
- Boot sector, boot block 1, boot block 2, boot block 3

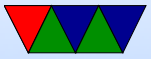



- Block group: superblock, fs descriptor, block bitmap, inode bitmap, inode table, data blocks



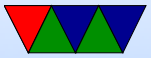
# Block Group Contents

- 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
- Some reserved space to allow filesystem growing



# Superblock

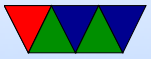
- located at 1k offset, 1k long
- Copies scattered throughout (fewer in later versions)
- Info on all the inode groups, block groups, etc.
- Copy in each block group, but typically only 1st one used



# Superblock Layout

| 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 |

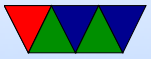
| Offset | Size | Description |
|---|---|---|
| 54 | 2 | mounts between fsck |
| 56 | 2 | ext signature (0xef53) |
| 58 | 2 | fs status (dirty or clean) |
| 60 | 2 | what to do on error |
| 62 | 2 | minor version num |
| 64 | 4 | time of last fsck |
| 68 | 4 | interval between fsck |
| 72 | 4 | OS of 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

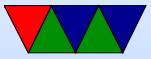
- Follows right after superblock

| 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 |



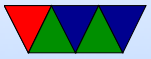
# Block Tables

- Block bitmap
  - bitmap of blocks (1 used, 0 available)
  - block group size based on bits in a bitmap.
- Hypothetical Example
  - If blocksize 4kb, and the block bitmap is 8 blocks then there's room to map 32k blocks (32k* 8 bits per byte) This could map a filesystem of 32k blocks * 4k blocksize = 128MB



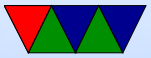
# Inodes (index-nodes)

- All metadata (except filename) for file stored in inode
- inode entries are 128 bytes.



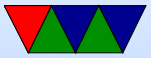
# Inode Table Contents

- Inode bitmap – bitmap of available inodes
- Inode table
- Second entry in inode table points to root directory
- Can you run out of inodes before you run out of disk?
- Can use `df -i` to see free inodes on filesystem



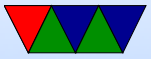
# Inode Layout

| 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 |



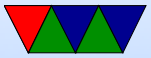
# Inode Types

- FIFO
- char-device
- directory
- block-device
- regular file
- symbolic link
- UNIX socket



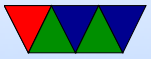
# Inode Permissions

- 8-bits: St rwx rwx rwx
- S = setuid, run as separate user
- t = sticky bit (for /tmp, prevents users from deleting files unless belongs to them)
- read/write/execute for user, group, and everyone



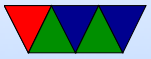
# Inode Finding Blocks

- Addresses of first 12 blocks stored in inode (on 4k filesystem, allows addressing 48k)
- If need more, the next address actually points to an "indirect" block that points to a block full of more pointers. On 4k filesystem this would be 1024 more entries
- If need more than that, the next address in the inode is a pointer to a double-indirect block. It points to an indirect block, but each entry in that points to an

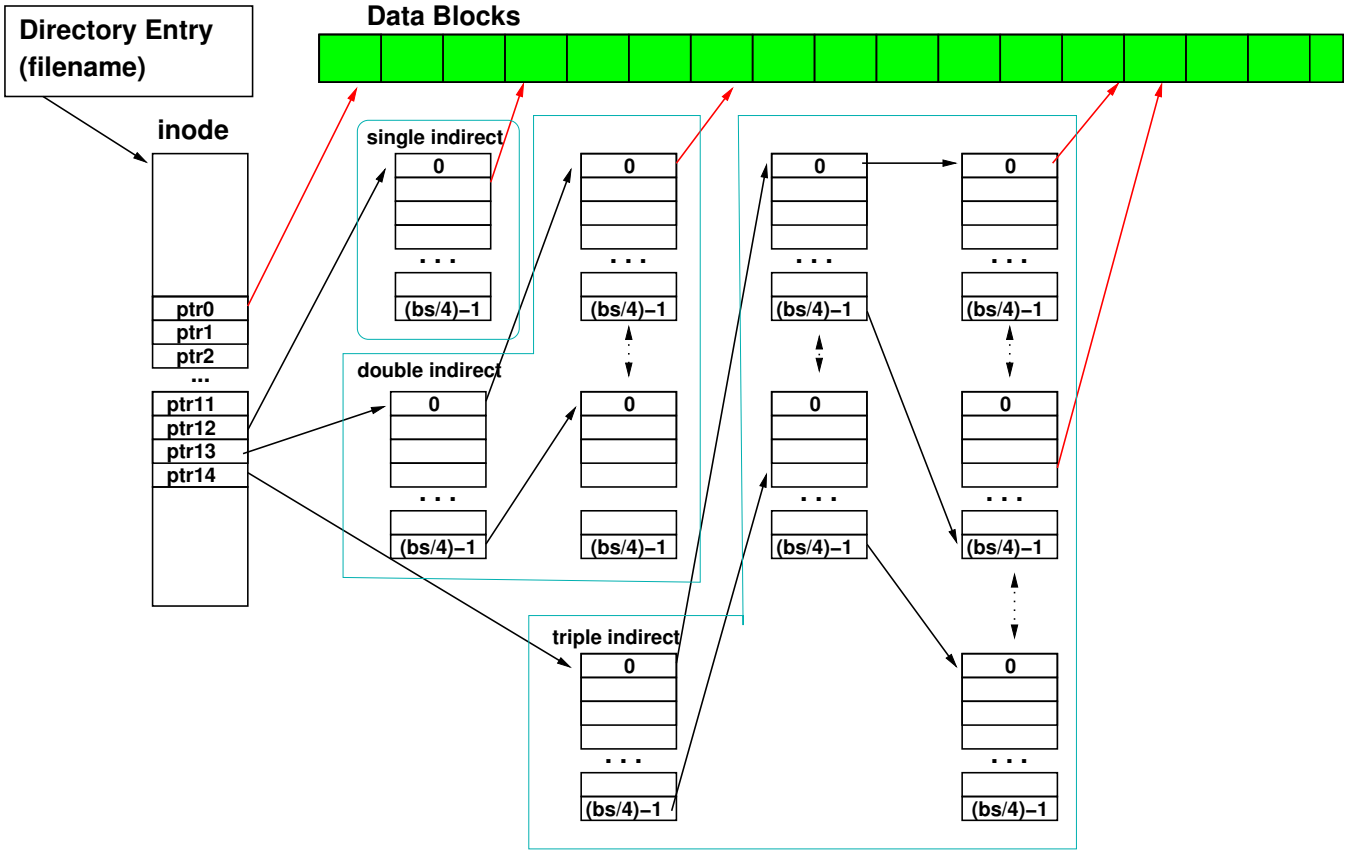


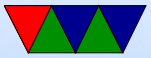
indirect block full of addresses

- Finally, if that's not enough, there's a final triple-indirect block



# Inode Finding Blocks Diagram

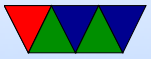


# Directory Info

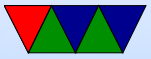
| type | size |
|---|---|
| inode of file | 4 |
| size of entry | 2 |
| length of name | 1 |
| file type | 1 |
| file name | N |

- Directory is just an inode
- Directory inode has info/permissions/etc just like a file
- Directory entries are stored in a data block pointed to



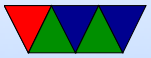
by the indirect blocks

- 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.
- Entries cannot span block boundary (waste space)
- 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



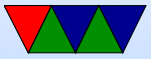
# Finding Root Directory

- Superblock links to root directory, (usually inode 2)



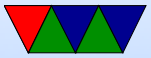
# How to find a file

- Find root directory
- Iterate down subdirectories
  - ext2: linked list
  - ext3/4: hash
- Get inode



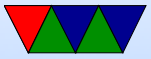
# How to read an inode

- 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

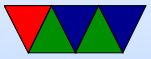


# Ext3

- Excellent (though dated) paper on it here
  `https://www.kernel.org/doc/ols/2005/ols2005v1-p`
  `pdf`
- Backwards/Forwards Compatible with ext2
- Htree instead of linked list in directory search
- Extended attributes (fine-grained permissions)
- Online fs growth
- Journal
  metadata and data written to journal before commit.

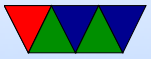


Can be replayed in case of system crash.

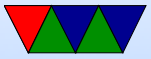


# H-trees, B-trees and others

- B-tree
  - Like a binary tree, but can have many entries in each node
  - Allows finding files much faster than linked-list
  - Can be complex (at time of ext3 paper, the b-tree implementation for XFS alone was more code than all of the total ext2/3 fs code)
- H-tree
  - Variant of a B-tree

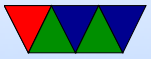


- Uses hash of filename
- Hash down to 32-bit value before doing lookup. With two level-table can lookup 16-millions 52-char filenames very quickly
- 50-100x speedup on some workloads, but it does make readdir() more complicated

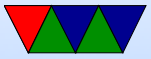


# Ext4

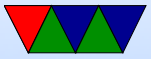
- Filesize up to 1Exabyte, filesize 16TB
- Extents (Rather than blocks)
  - Contiguous area of blocks
  - Only need to store start and how long, much more compact
  - On ext4 an extent can map up to 128MB of contiguous space in one entry
  - Does complicate allocation, fragmentation can limit use



  - Can make truncation/deletion slower
  - Can store extents in an htree?
- Pre-allocate space, without having to fill with zeros at allocation time (which is slow)
- Delayed allocation – only allocate space on flush (wait a bit before lots of writes happen) so data more likely to be contiguous
- Unlimited subdirectories (32k on ext3 and earlier. why? because all files have hard-link to directory and hard-link count was 16 bits. No negative to avoid bugs)
- Checksums on journals



- Improved timestamps, nanosecond resolution, push beyond 2038 limit by grabbing unused bytes elsewhere
- Scalability: superblock locking issues. By avoiding updating block count unless statfs() (df) run no need to try to take lock on whole superblock every time a block allocated



# Why use FAT over ext2?

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