**Administrivia**

- Project 4 due in a week
  - Turn in only, no report
- Homework 4 due next Wednesday
- EC

- Today:
  - Project 4 and file system stuff
  - EC questions?

**Project 4**

- Work with a real file system
- Given:
  - cse451fs: simplified file system for Linux
- Goals:
  - Understand how it works
  - Modify implementation to:
    - Increase maximum size of files (currently 13kB)
    - Allow for longer file names (currently 30 chars)

**Linux FS layers**

[Diagram of Linux file system layers]

**File systems in Linux**

- Layered on top of a block device
  - Device provides a big array of blocks
  - Blocks are cached in the buffer cache
- Implement a standard interface
  - file_operations
    - read/write/seek files; read directory
  - inode_operations
    - create / lookup / unlink / mkdir / rmdir / rename
  - super_operations
    - read/write inodes
  - address_space_operations
    - readpage/writepage for memory-mapped I/O
  - file_system_operations
    - read in superblock

**Project 4 Setup**

- Build a kernel module for cse451fs (and a kernel supporting cse451fs)
- Transfer it to VMware
- On VMware, use a ramdisk to test your file system.
  - i.e. create a fake disk in memory, create your FS on top, mount, test.
- load cse451fs
- Make a file system using (modified) mkfs tool
- mount, test
- Step 1: try this procedure with given code
- Step 2: read cse451fs.h, then dir.c

**cse451fs disk structure**

- boot
- superblock: tells where all other things are
  - Contains inode map:
    - Bit array, tracks which inodes are currently in use
    - E.g. for 3 dirs + 4 files, need 7 inodes
- Data map:
  - Bit array, tracks which data blocks are in use
cse451fs structure

#define CSE451_NUMDATAPTRS 13

struct cse451_superblock {
    140 u16 s_magic; // magic number
    char s_imag[0]; // name for inode map
}

struct cse451_inode {
    u16 i_mode; // determines if file or dir
    u16 i_links; (+ protection)
    u16 i_uid;
    u16 i_gid;
    u32 i_filesize;
    u32 i_datablocks[CSE451_NUMDATAPTRS];
};

- Inode size?
- Multiple inodes per block?
- How many for 1K block?
- mksfs decides how many inodes to create
- mksfs.cse451fs.c create an inode for every three data blocks

Data blocks

- Blocks for regular files contain file data
- Blocks for directories contain:

#define CSE451_MAXDIRENTRY 30

struct cse451_dir_entry {
    u16 inode;
    char name[CSE451_MAXDIRENTRY];
};

- Data block for / directory containing:
  
  - etc bin

  - What's this dir's inode number?
  - What is the "file size" field in this dir's inode?

Sample data block usage

For a 4MB file system with 1KB blocks

  - / etc
  - /bin
  - /sh
  - /dev

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Size</th>
<th>Data Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>etc</td>
<td>4 entries + 1 null entry</td>
<td>1</td>
</tr>
<tr>
<td>bin</td>
<td>4 entries + 1 null entry</td>
<td>1</td>
</tr>
<tr>
<td>sh</td>
<td>4 entries + 1 null entry</td>
<td>1</td>
</tr>
<tr>
<td>etc/passwd</td>
<td>1024 bytes</td>
<td>1</td>
</tr>
<tr>
<td>etc/shell</td>
<td>100 bytes</td>
<td>1</td>
</tr>
<tr>
<td>etc/sh</td>
<td>10,000 bytes</td>
<td>10</td>
</tr>
<tr>
<td>dev/char</td>
<td>5,000 bytes</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Project 4 requirements

- Increasing maximum size of files
  - Be efficient for small files but allow large files
  - Changing constant (=13) is not enough.
  - Come up with a better design/structure for locating data blocks.
  - Indirect blocks?
  - Don't have to support arbitrarily large files
    - Fine to have constant new_max (but new_max >> old_max)
  - Allow for longer file names
    - Be efficient for short files names but allow large file names
    - Again, don't just change the constant

Approaches for longer file names

- Store long names in a separate data block, and keep a pointer to that in the directory entry.
  - Short names can be stored as they are.
  - Recommended
- Combine multiple fixed-length dir entries into a single long dir entry (win95)
  - It is easier if the entries are adjacent.
- Put a length field in the dir entry and store variable length strings
  - need to make sure that when reading a directory, that you are positioned at the beginning of an entry.
Getting started with the code

- Understand the source of the limits in the existing implementation
  - Look at the code that manipulates dir entries
  - midx code
  - dir.c in the file system source code
- Longer file names:
  - The code for will largely be in dir.c: add_entry() and find_entry()
  - In mks, change how the first two entries (for "." and ".") are stored
- Bigger files:
  - super.c:get_block()
  - References to _i datablock[] array in an inode will have to change

VFS vs cse451fs

- Don’t conflate VFS structures and cse451fs structures!
  - inodes, superblocks
  - E.g., there are "two" inodes:
    - VFS struct inode
    - Generic inode used in Linux source (works for any FS)
    - Lives in memory
    - cse451 struct cse51_inode
    - Actual inode representation on disk
  - inode: cse451_read_inode converts from cse451_inode to struct inode
    - Copies over mode, size, etc
    - Copies over _i datablock[] to struct inode’s generic_ip field
    - (which will now be used as type cse51_inode_ino)
  - inode: cse451_write_inode converts the other way

Linux Buffer Manager Code

- To manipulate disk blocks, you need to go through the buffer cache
- Linux buffer cache fundamentals:
  - blocks are represented by buffer_heads
  - Just another data structure
  - Actual data is in buffer_head->b_data
  - For a given disk block, buffer manager could be:
    - Complete unaware of it
      - no buffer_head exists, block not in memory
    - Aware of block information
      - buffer_head exists, but block data (b_data) not in memory
    - Aware of block information and data
      - Both the buffer_head and its b_data are valid ("$ hit")

Accessing blocks

- To read a block, FS uses bread(...):
  - Find the corresponding buffer_head
    - Create if doesn’t exist
    - Make sure the data is in memory (read from disk if necessary)
  - To write a block:
    - mark_buffer_dirty() + brelse() - mark buffer as changed and release to kernel (which does the writing)

Some buffer manager functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bread(io, block, create)</td>
<td>Get the buffer_head for the given disk block, ensuring that the data is in memory and make sure the reference count is correct; always pair with a brelse.</td>
</tr>
<tr>
<td>cse451_getblk(io, block, create)</td>
<td>Get the buffer_head for the given disk block. Does not guarantee anything about the state of the actual data. Increment ref count, always pair with a brelse. Check for new blocks, compute the cost incurred.</td>
</tr>
<tr>
<td>brelse(bh)</td>
<td>Decrement the ref. count of the given buffer.</td>
</tr>
<tr>
<td>mark_buffer_dirty(bh)</td>
<td>Mark the buffer modified, meaning needs to be written to disk as soon possible.</td>
</tr>
<tr>
<td>mark_buffer_opdate(bh)</td>
<td>Marks that the data pointed to by bh is valid.</td>
</tr>
</tbody>
</table>

Hints

- Learn how to use bread/brelse and other buffer cache stuff by looking at provided code
  - All printk messages stored in /var/log/messages
  - Can view to examine long debug outputs
- Q: "It is extremely frustrating not to be able to read debug messages because they scroll off screen in vmware so quickly :(")
  - A: Use Shift-pageup and Shift-pagedown
- Q: "How does IS get its entries?"
  - dir.c:readdir()
A gcc warning

- gcc might insert extra space into structs
  - How big do you think this is?
    ```c
    struct test { char a; int b; }
    ```
  - Why is this a problem?
    - What if `test` represents something you want on disk?
      - e.g. directory entries
    - Discrepancy between the disk layout and memory layout
  - Fix:
    ```c
    struct test2 { 
      char a;
      int b;
    } __attribute__((packed));
    ```
  - `sizeof(test2)` is now 5

More hints

- Some stuff in Linux kernel is limited to 256 chars
  - e.g. VFS, fs
  - Be careful when testing long filenames!
- `dd` is useful for creating large test files
  - `dd if=/dev/zero of=200k bs=1024 count=200`
- `du` is useful to check you're freeing everything correctly