Reminders

- Project 4 due Dec 10
- Turn in only (include writeup)
- If you used your late pass on HW5
  - Turn in to me, by Tue, Dec 7 at the latest.

Today:
- Project 4 and file systems
- HW4 + Project 2 back (finally)
  - HW4 average: 64.8/80
  - Project 2 average: 74/85

Project 4 first steps

- Go through the mechanics
  - Compile the kernel & file system
  - Boot VMWare with new kernel
  - Install ramdisk, install cse451fs, mount, test
- Read the code
  - Start with cse451fs.h
  - Follow with mkfs & dir.c
- Start with increasing filename length

CSE 451 _ Superblock

```
struct cse451_super_block
{
    _u16 s_numInodes;  // inode map is tail of superblock
    _u16 s_dataMapStart; // block # of first data map block
    _u32 s_dataMapBlocks; // data map size, in blocks
    _u32 s_inodeStart;  // block # of first inode block
    _u32 s_numInodes;   // number of inodes
    _u32 s_dataBlocks;  // number of blocks of data
    _u32 s_numBusyInodes; // number of inodes in use
    _u16 s_magic; // magic number
    char s_imap[0]; // name for inode map
};
```

CSE 451 _ Inode Structure

```
#define CSE451_NUMDATAPTRS 13

struct cse451_inode
{
    __u16 i_mode;  // determines if file or dir
    __u16 i_nlinks; (+ protection)
    __u16 i_uid;
    __u16 i_gid;
    __u32 i_filesize;
    __u32 i_datablocks[CSE451_NUMDATAPTRS];
};
```

For a 4mb disk:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_numInodes</td>
<td>1385</td>
</tr>
<tr>
<td>s_dataMapStart</td>
<td>2</td>
</tr>
<tr>
<td>s_dataMapBlocks</td>
<td>1</td>
</tr>
<tr>
<td>s_inodeStart</td>
<td>3</td>
</tr>
<tr>
<td>s_numInodesBlocks</td>
<td>68</td>
</tr>
<tr>
<td>s_dataBlocks</td>
<td>4008</td>
</tr>
<tr>
<td>s_numBusyInodes</td>
<td>7</td>
</tr>
<tr>
<td>s_magic</td>
<td>CSE451_SUPER_MAGIC (0w451f)</td>
</tr>
</tbody>
</table>

Inode structure

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

Blocks for files contain file data
Blocks for directories contain:

```c
#define cse451_MAX_NAMELEN 30
struct cse451_dir_entry {
    _u16 inode;
    char name[cse451_MAX_NAMELEN];
};
```

Data block for / directory containing:
- .. etc bin
- What’s this dir’s inode number?
- What is the “file size” in this dir’s inode?

Sample data block usage

For a 4MB file system with 1KB blocks

```c
struct inoode {
    // File
    ...data...
    // directory
    ...entry...
};
```

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.
- 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

- Combine multiple fixed-length dir entries into a single long dir entry (Win95)
  - It is easier if the entries are adjacent.
- 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.
- 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
  - 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 mfs, 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 cse451_inode`
    - Actual inode representation on disk
- `inode.c:cse451_read_inode` converts from `cse451_inode` to `struct inode`
- `cse451_write_inode` converts the other way
### Linux Buffer Manager Code

- Recall that blocks are cached in buffer cache
- Main block data structure is buffer_head
- 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, block data (b_data) not in memory
  - Aware of block information and data
    - Both the buffer_head and its b_data are valid
- 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)

### Misc tips

- All printk messages stored in `/var/log/messages`
  - Easier to examine long debug outputs
- Learn how to use bread/brelse by looking at provided code
- 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: what is dentry?
  - Directory entry. `dcache` caches recently used dentries (because searching directories is linear and slow)