ECE 391 Discussion Week 9

Announcements & Reminders

- MP3.2 due Oct 29th 6:00pm
- If you haven't, return your TUX controllers at 3.2 handin
- No regrades until the Final Demo!
- Extra Credit worth at most 10% of MP3
  - Must finish MP3 before you can receive extra credit
  - Very difficult to get points (e.g. a start up screen or a fancy BSOD does not count for anything!)
  - Can compete for prizes from Microsoft (top 3 groups)
- Start early, plan ahead for CP3!
- As always, RTDC

MP3.2 Overview

- Terminal Driver
- RTC Driver
- Read Only File System Driver
- In Linux, all devices are treated like files, e.g. there is a file "rtc" in the FS.
- All your drivers should have **read**, **write**, **open**, **close** functions (even if they don't do anything useful right now!)
  - Generalized common interface that MUST match the system call parameters!
  - Encapsulated in a file operations table (a jump table)
  - Definitely read ahead to CP3 to help plan out CP2 functions
  - If not specified, use the convention that return 0 for success, return -1 for failure

Terminal Driver

- Support all alphanumeric keys and symbols (excluding the number pad, home/end section)
  - Shift, Ctrl, Alt, Capslock, Backspace and etc.
  - But you should still have a default case to handle unknown scan codes!
  - Don't output special functionality key presses
- Ctrl-L for clear screen and put cursor to upper left corner
- Scrolling support
  - Scroll to make new space for outputting at the bottom of the screen
  - This is **NOT** page up/down to look at history of commands or the screen history

Terminal Driver (contd.)

- Keyboard buffer is 128 bytes
  - Do **NOT** extend this limit
  - Enter (newline character) is also a character
- Terminal driver is **NOT** the shell
  - Don't implement a prompt such as "user>" in your driver
- Terminal open() initializes terminal stuff (or nothing), return 0
- Terminal close() clears any terminal specific variables (or do nothing), return 0
- Terminal read() reads **FROM** the keyboard buffer into buf, return number of bytes read
- Terminal write() writes **TO** the screen from buf, return number of bytes written or -1

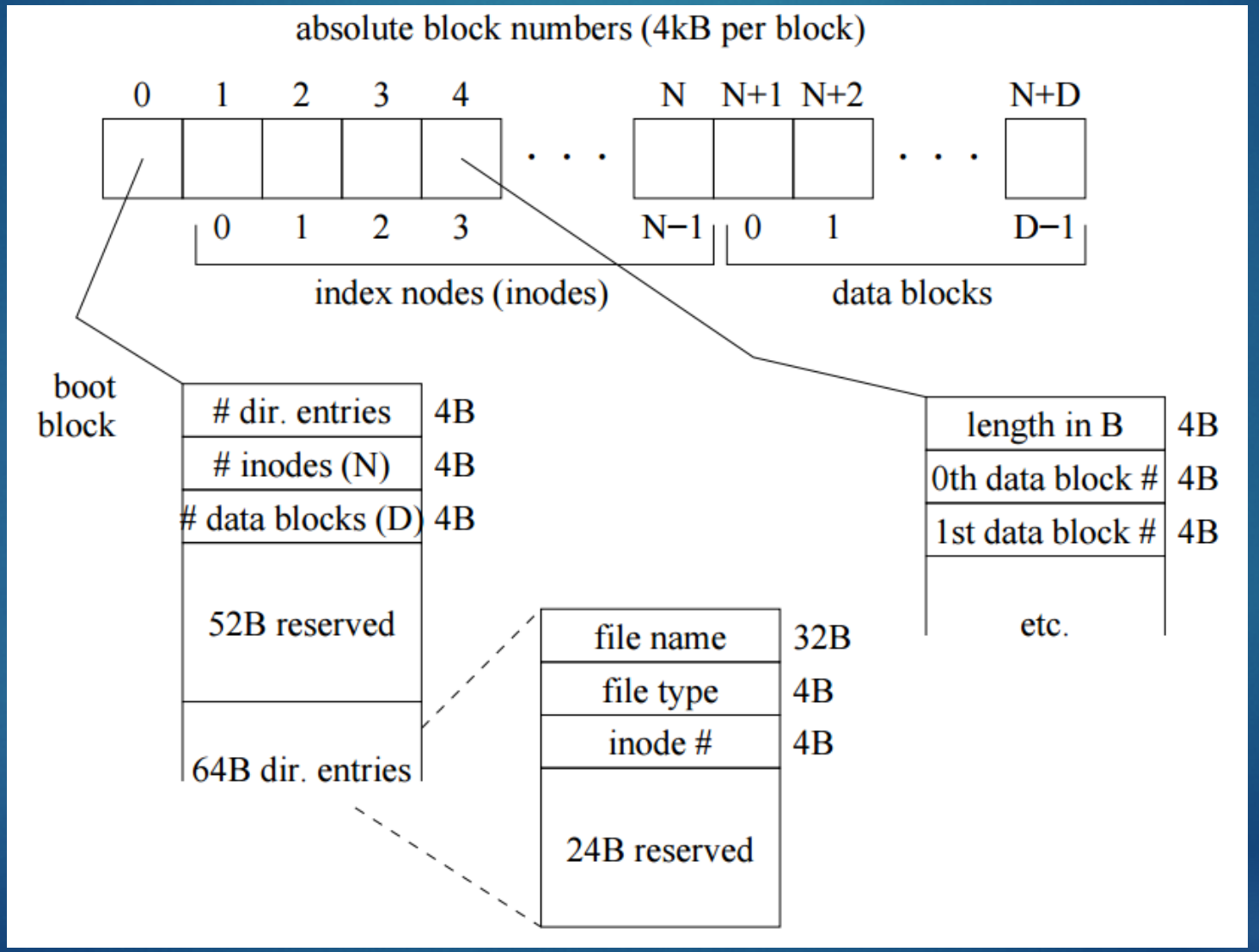
RTC Driver

- RTC open() initializes RTC frequency to 2HZ, return 0
- RTC close() probably does nothing, unless you virtualize RTC, return 0
- RTC read() should block until the next interrupt, return 0
  - **NOT** for reading the RTC frequency
- RTC write() must be able to change frequency, return 0 or -1
  - New frequency passed through buf ptr or through count?
  - Frequencies must be power of 2
- Virtualizing RTC is not required but will be helpful in the future
  - This means RTC can be set to any frequency but the program using RTC will not be affected

File System Driver

- You are given an in-memory filesystem (FS), `filesys_img`, that is compiled into the `mp3.img` and loaded at boot time
  - Look in `kernel.c` to find the address.
  - Do NOT try to hardcode the address
- Read-Only
  - Nothing to code in the write function, but you still need a write function
- Flat structure
  - Only one directory called "."
- You don't need to implement a file descriptor yet, but you should read Appendix A and understand it for CP3

File System Structure



File System Driver (contd.)

- In order to make your design better align with CP3, write separate (open/read/write/close) functions for files and directory.
- E.g. you may have
  - file_read, file_write, file_close, file_open
  - dir_read, dir_write, dir_close, dir_open

File System Driver (files)

- File open() initialize any temporary structures, return 0
- File close() undo what you did in the open function, return 0
- File write() should do nothing, return -1
- File read() reads `count` bytes of data from file into `buf`
  - Uses `read_data`
- Do NOT assume data blocks are contiguous
  - E.g. inode #1 might have its data stored in data blocks 1, 11, 15, 16

File System Driver (directory)

- Directory open() opens a directory file (note file types), return 0
  - `read_dentry_by_name`: name means filename
- Directory close() probably does nothing, return 0
- Directory write() should do nothing, return -1
- Directory read() read files filename by filename, including “.”
  - `read_dentry_by_index`: index is **NOT** inode number