ECE 391 Discussion Week 9

Announcements & Reminders

- MP3.2 due Monday Oct 24th 5:59pm
- 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!)
- Start early, plan ahead for CP3!
- Exam 2 on Tuesday November 1st 7-9PM.

MP3.2 Overview

- Terminal (Keyboard) Driver
- RTC Driver
- Read Only File System Driver

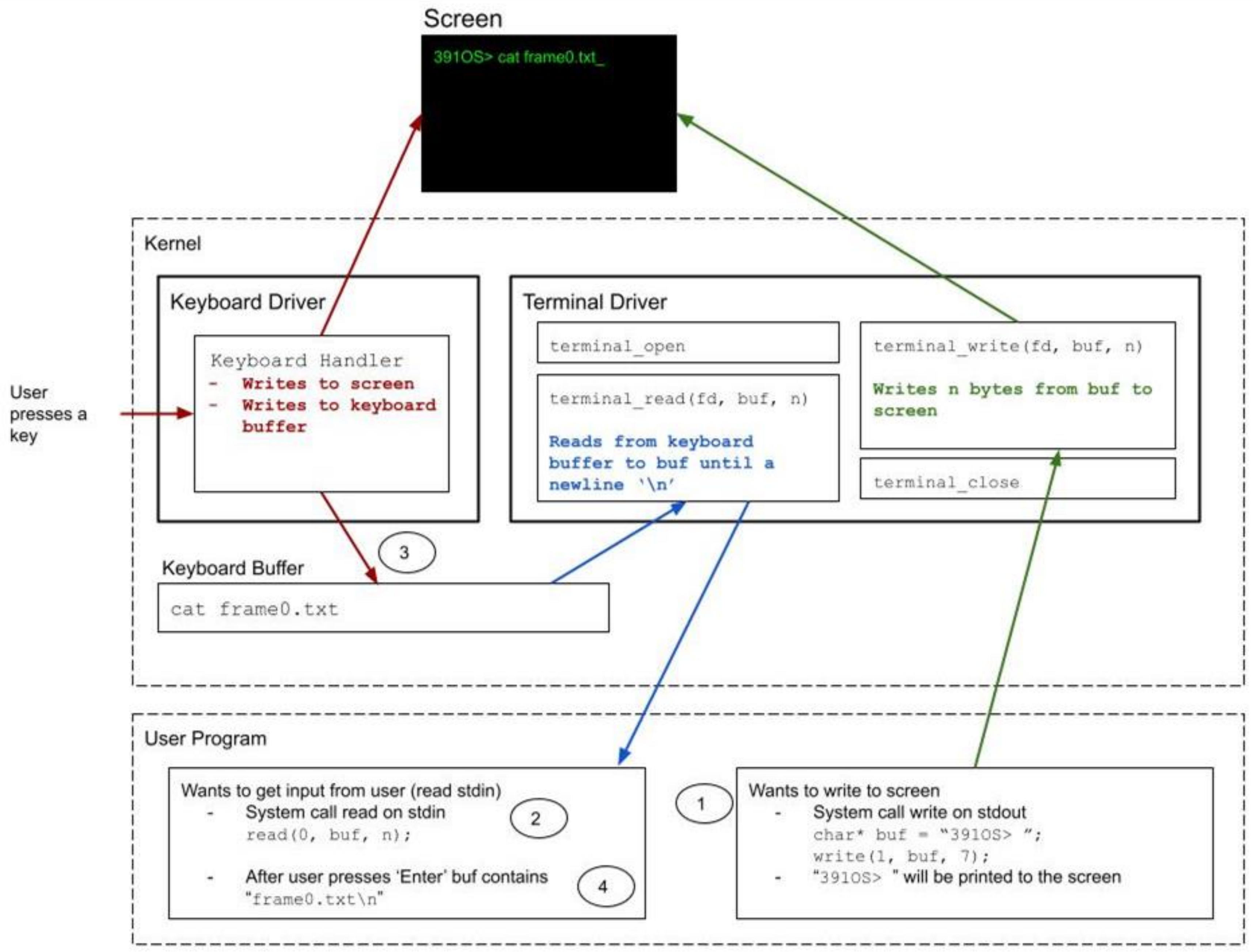
- All your drivers should have **read, write, open, close** functions (even if they don’t do anything useful right now!)
  - These four functions will be mapped to your system call functions in CP3
  - Make sure you are using the parameters correctly
  - If not specified otherwise, they should return 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.
- Ctrl-L for clear screen and put cursor to upper left corner
  - Don’t output special functionality key presses
- 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 (Keyboard) 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


Screen
391OS> cat frame0.txt_
Kernel
Keyboard Driver
Keyboard Handler
- Writes to screen
- Writes to keyboard buffer
User presses a key
Terminal Driver
terminal_open
terminal_read(fd, buf, n)
Reads from keyboard buffer to buf until a newline '\n'
terminal_write(fd, buf, n)
Writes n bytes from buf to screen
terminal_close
3
Keyboard Buffer
cat frame0.txt
User Program
Wants to get input from user (read stdin)
- System call read on stdin
read(0, buf, n);
- After user presses 'Enter' buf contains "frame0.txt\n"
2
4
1
Wants to write to screen
- System call write on stdout
char* buf = "391OS> ";
write(1, buf, 7);
- "391OS> " will be printed to the screen

1. The user program (shell) calls t erminal write to print "391OS> " t o the screen.

2. The user program wants to get input from the user, so it calls ter minal read.

3. The user types "cat frame0.txt" character by character. The keyb oard handler writes this to the scr een and keyboard buffer.

4. The user presses Enter and ter minal_read copies “cat frame0.txt \n" from keyboard buffer to buf.

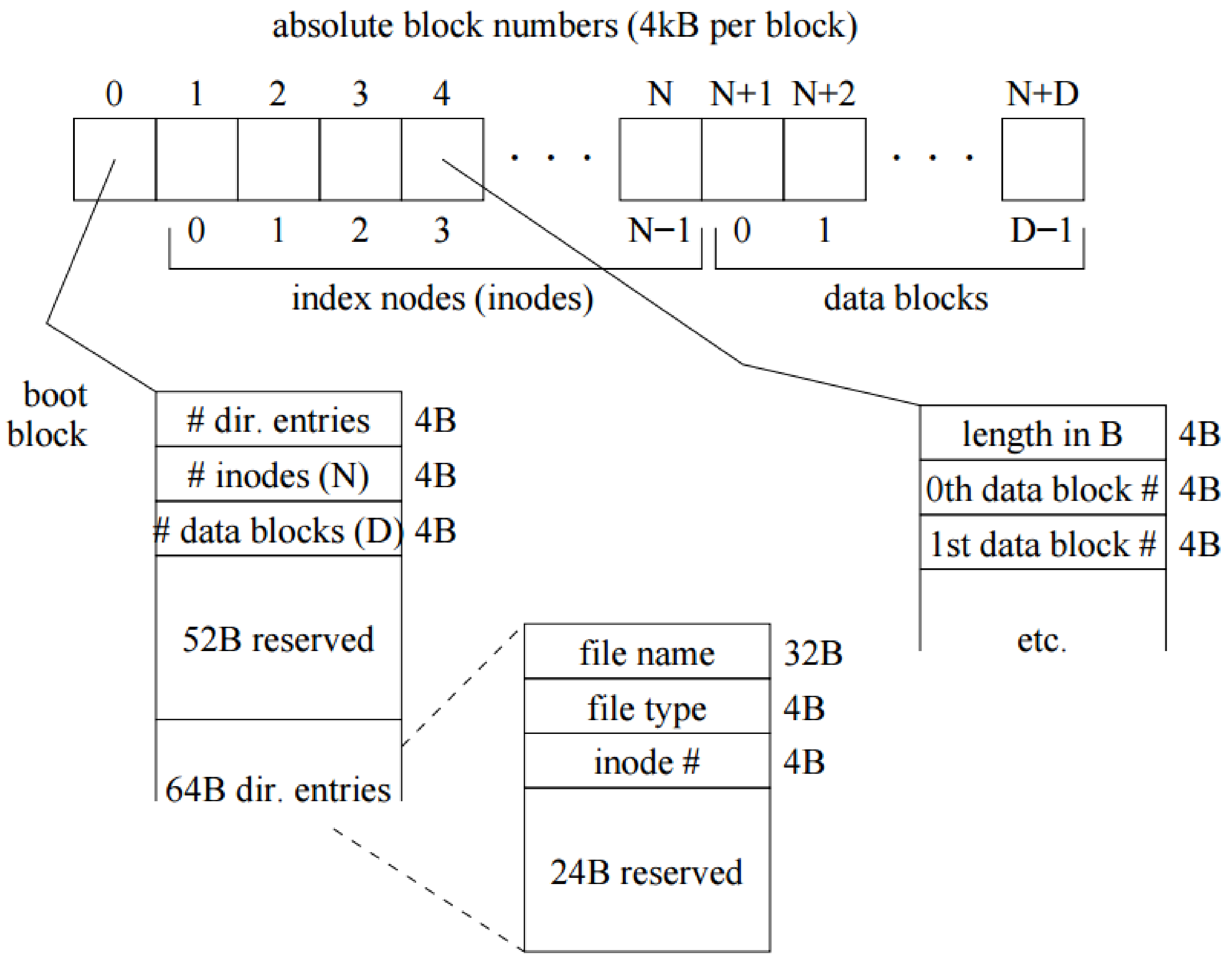
RTC Driver

- 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

- Read Appendix A.
- 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 (files)

- Do NOT assume data blocks are contiguous
  - E.g. inode #1 might have its data stored in data blocks 1, 11, 15, 16
- File open() initialize any temporary structures, return 0
  - Uses `read_dentry_by_name`: name means filename
- 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`

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