ECE 391, Computer Systems Engineering
MP3 Checkpoint 2 Hints

General Guidelines

This document is intended to clarify our expectations for the demo procedure. If you have any feedback to make the document more clear and concise, please let us know on Piazza.

1. During the demo, we will normally run only your tests. If we determine that your tests are inadequate, we will add tests, and you will lose some points.
2. Remember that your tests are running as kernel code.
3. We will be testing different sections independently and will restart your OS multiple times with different tests enabled each time. For example, we may test whether you can read small files once and then large files the next time.
4. We suggest that you get comfortable with function pointers and writing assembly code in separate .S files, as inline assembly can be tricky and lead to subtle bugs in your code.
5. Remember to take advantage of the functionalities C and x86 Assembly offer you, **code smart!**. For example, use structs for the file system structures and use arrays for the terminal.
6. Remember to maintain your **bug log**! While we know that you are capable programmers, we know that everyone has bugs. If you tell us that you had no bugs and hence have none in your bug log, we won't believe you.
7. As always try to use your best style and document code as you write it.
8. All your drivers must have the `open, close, read` and `write` functions defined, even if they don't do anything and just return a constant value. Likewise, all these functions must return an integer and follow the interface provided for the corresponding systems calls.
9. Read ahead for the next checkpoint as a lot of the functionality there is dependent on features that you implement in this checkpoint.
10. Please look at the hints post on Piazza to find a gif of how test cases should look. Remember that the example list is not comprehensive.
11. Remember to validate your input parameters!

RTC Driver

1. `rtc_open` should reset the frequency to 2Hz.
2. Make sure that `rtc_read` must return only after an RTC interrupt has occurred. You might want to use some sort of flag here (you will not need spinlocks. Why?)
3. `rtc_write` must get its input parameter through a buffer and not read the value directly.
4. There is a one line formula that you can use to check if a number is a power of two. You do not need to use a lookup table or switch statement to do so. *Hint: Think binary or in terms of bits.*

5. Your test cases should be able to demonstrate that you are able to change the rtc frequency through all frequencies.

6. Though not required for the MP, it is highly recommended that you virtualize your RTC by the next checkpoint. By virtualizing the RTC you basically give the illusion to the process that the RTC is set to their frequency when in reality the RTC isn't. Instead you set the RTC to the highest possible frequency and wait until you receive x interrupts to return back to the process such that those x interrupts at the highest frequency is equivalent to 1 interrupt at the frequency the process wants.

Terminal Driver

1. All numbers, lower and upper case letters, special characters, the shift, ctrl, alt, capslock, tab, enter and backspace keys should work as intended. You do not need to print anything on hitting a function key, such as shift.

2. You need not care about mapping the arrow keys or the numpad. **As long as pressing these keys does not crash your kernel, you won't lose points for not printing these keys**, these keys should not be counted as part of your input buffer either.

3. `terminal_read` only returns when the enter key is pressed and should always add the newline character at the end of the buffer before returning. Remember that the 128 character limit includes the newline character.

4. `terminal_read` should be able to handle buffer overflow (User types more than 127 characters) and situations where the size sent by the user is smaller than 128 characters.

5. If the user fills up one line by typing and hasn't typed 128 characters yet, you should roll over to the next line. Backspace on the new line should go back to the previous line as well.

6. `terminal_write` should write the number of characters passed in the argument. Do not stop writing at a NUL byte.

7. You may choose to not print the NUL bytes (recommended).

8. Both `ctrl + l` and `ctrl + L` should clear the screen. You should not be resetting the read buffer if the user clears the screen before pressing enter while typing in `terminal_read`.

9. Your keyboard test can just be a calling `terminal_read`, followed by `terminal_write` in a while loop.

File System Driver

1. You must define the `open`, `close`, `read` and `write` functions for both files and directories. So in total you will have 8 functions.

2. `read_dentry_by_name, read_dentry_by_index` and `read_data` are **NOT** provided for you. The documentation says that these functions are provided by the file system module, you are required to write the file system module

3. You should have tests to demonstrate that you can read small files (`frame0.txt, frame1.txt`), executables (`grep, ls`) and large files (`fish, verylargetextwithverylongname.tx(t)`).

4. `directory_read` should only read one file name at a time.

5. The file names in the dentries need not be NUL-terminated.

6. On reading executables, you should see "ELF" in the beginning and the magic string "0123456789ABCDEFGHIJKLMNOPQRSTU" at the end. If you're printing NUL bytes, you might not be able to see both at the same time. If you're using printf to print you might not be able to see the last magic string, as printing will stop at NUL bytes.

7. Your test for `directory_read` should look like the output from ls, look at the gif for more information.