ECE 391, Computer Systems Engineering
MP3 Checkpoint 1 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 enable the RTC once, test it, and then the next time enable keyboard interrupts to test your keyboard functionality.
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 IDT entries ad Paging structures, and use arrays for the keyboard handler!
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.

Initialize the IDT

1. For this checkpoint, printing a prompt as to which Exception was raised and freezing by using a while loop is sufficient.
2. You will be implementing System Calls in Checkpoints 3 and 4. For now, you may print that a system call was called when the handler associated with 0x80 is executed.
3. It is highly recommended that you have a common assembly linkage. You will be required to do this at a later checkpoint, so you may as well implement it now.
4. For entries 0x14 to 0x1F in the IDT, you need not fill in anything for the IDT.
5. Once an exception has occurred and the processor is executing the handler, your OS should not allow any further interrupts to occur.
6. You must have tests to show that you are able to raise **multiple** exceptions.

Initializing the RTC

1. To show us during demo that the RTC interrupts are being received, you should use the `test_interrupts` handler or a function similar to it. If working correctly, the characters on screen should alternate between a set of different characters at every fixed interval.
2. Though not required for this checkpoint or the next, 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.

Initializing the Keyboard

1. It is sufficient if you show that pressing a key on the keyboard can echo that key on the monitor.
2. We will only test lowercase letters and numbers for this checkpoint and if a single press displays the right character. We will not stress test your keyboard at this checkpoint.
3. You need not care about mapping function keys, special keys such as (but not limited to) alt, ctrl, caps lock, shift, tab, the arrow keys or the numpad for this checkpoint. **As long as pressing these keys does not crash your kernel, you won't lose points for not printing these keys**.
4. Typing to get the right characters on the keyboard is sufficient in terms of testing.

Paging

1. Open the qemu console and use the command `info mem` to see if you have mapped the pages correctly.
2. You can allocate memory for the paging structures in the `x86_desc.S` file, this will place the paging structures in the kernel page. Using structs and arrays will make your job easier.
3. Video memory must only be **one 4KB page**. You may be penalized for allocating more pages than needed for video memory. RTDC and RTCC to see which 4KB page video memory should be at.
4. You must have tests that show that dereferencing locations that shouldn't be accessible aren't accessible. Conversely, you must also have tests that show that dereferencing locations that should be accessible are accessible.
5. Test all edge cases.
6. Data types for pointers are important! Don't overlook them and confuse yourself when a test doesn't work as expected.