Systems Programming (6CC514) Assessed Component 1

Percentage of final grade: 15%

Date set: Thursday 29th October 2016.

Submission date: 9pm on Wednesday 5th October 2016.

Task

This week, you have seen how the code has been added to the first stage of the bootloader that will load the second stage of the bootloader and transfer control to it.

Your task is to start writing the second stage of the bootloader (boot2.bin) and include the rules in the makefile so that the second stage of the bootloader is built and written to the disk image. The code you produce must do the following:

1. Start execution at memory address 0000:9000h.
2. Use the stack and data segments from stage 1.
3. When stage 2 starts executing, it must display an appropriate message on the screen to indicate that stage 2 of the boot process is running.
4. It must include the code provided in a20.asm and make a call to Enable_A20 to enable the A20 address line. Upon return from Enable_A20, your code must display a message on the screen to indicate how the A20 line was enabled (BIOS, keyboard controller, not enabled, etc). Unfortunately, the code in a20.asm does not follow best-practice for commenting code, so you will have to work out for yourself how it returns a status code to indicate how the A20 line was enabled and what the different code values are.
5. Once stage 2 has completed step 4, it should hlt for now. We will continue the process of switching to protected mode next week.
6. Stage 2 of the boot loader must be exactly 2560 (5 * 512) bytes in size.

You have been provided a starting point that includes stage 1 of the bootloader and the makefile to build stage 1. You need to make the appropriate additions to the makefile to build your stage 2 and to write it to the disk. The command to write boot2.bin to the disk is as follows:

```
dd status=noxfer conv=notrunc seek=1 if=boot2.bin of=$(IMAGE).img
```

This command will write the file boot2.bin to the floppy disk image, starting at the second sector.

You should update the target ‘clean’ if necessary to clean up any additional files.

Finally, different emulators will implement different ways of enabling the A20 line. For example, Bochs enables it by default, so it will not exercise all of the code provided in a20.asm. So you need to think about how you are going to test the code that executes on return from Enable_A20 to ensure that all code paths in your second stage are tested. You should include with your solution, a text file called testing.txt that describes how you tested this.

Submission

You must submit a zip file containing your submission to submission point “Assessment 1 Submission Point” by the due date and time. Any submission that Course Resources indicates as being submitted late will not be marked and a grade of 0 given for this component.

The zip file must contain:

- The contents of the entire folder needed to build the bootloader (i.e. it must contain the code given to you as well as your additional code).
- The folder must be cleaned before you create the zip file. This means that it should be as if you had run ‘make clean’ on the folder. Your submission will be tested by using make to build the disk image containing your code and then it will be executed in Bochs.
- The file texting.txt must be included in the zip file.

The name of the zip file should be Assessment1_xxxxxxxxx.zip where xxxxxxxxx is your student number. For example, if your student number was 123456789, your zip file would be called Assessment1_123456789.zip. Any file that is not named using this convention or if something other than a zip file is submitted will result in the assessment not being marked and a grade of 0 for this assessment being given for this component.