

Assignment 1

This is the first assignment of several to build ikanOS on a 32-bit x86 machine from source. The subsequent ikanOS assignments build on this assignment. So, you will need to have this assignment fully working to attempt the second assignment, and so on. Turning in assignments late will hurt your ability to stay on track.

Using the base code I provided in Github, as well as the links below (and the assignment hints in the lecture slides), you will need to complete the multi-stage bootloader code. I consider this assignment a warm-up for later, more complex ikanOS assignments, so be sure to completely understand how this code works before moving on.

https://en.wikipedia.org/wiki/BIOS_interrupt_call

https://en.wikipedia.org/wiki/VGA_text_mode

There are several parts you must finish to make the code work:

1. Using the BIOS interrupt link above, load the bootloader-stage2 binary data (i.e., the entire ELF file from fs.img) to RAM starting at address 0x8000. This will need to be split into three parts (each of 128 sectors) since we cannot load more than 128 sectors reliably with BIOS interrupts in one operation. *(This allows our bootloader-stage2 binary to grow to 192 KB and still be loaded correctly.)* For the first part, you will load 128 sectors of 512 bytes from fs.img to memory address 0x8000. The second part will load an additional 128 sectors of 512 bytes from fs.img to memory address 0x18000 and so on for the third part. (Hint: due to segmentation in x86 real mode, you will need to use the “es” segment register to assist as you cannot just load 0x18000 into the bx register.) Your implementation of how to load these sectors using BIOS interrupts will be placed in bootloader-stage1.asm in the locations I specified in the lecture video (the TO DO sections).
2. Thoroughly read the VGA text mode link (above). Using that information, create your implementation of clearScreen(), printCharacter(), and printString() in libc-main.cpp based on the name of those functions and the arguments required. **You cannot modify any of the function signatures or anything else in the files I supplied other than the function implementations.**

Additional Hints: (1) Your BIOS interrupt code should start from sector 2 and use cylinder 0 and head 0. (2) Your clearScreen() function should zero out all of video RAM. (3) Your printCharacter() function should handle Line Feeds correctly (0x0A), Tabs correctly (0x09), and should consume, but ignore 0x0D). (4) Your printString() function should print on a newline for both 0x0A and 0x0D bytes.

3. Be sure to “make” your solution and see that it matches my screenshot below. **You cannot modify the Makefile, either.**

Additionally, be sure to use the constants (from constants.h) whenever possible. You should not be using a hard-coded numeric value if there is a corresponding constant in constants.h that you could use and makes sense for the context of use (but do not modify, add to, or alter constants.h in any way). You cannot create new helper functions and you should not create any new global variables. **Be sure to follow a professional coding standard, including the naming of your variables, structures, etc. You should use descriptive names that are appropriate.**

Nonsensical or whimsical names will lose points. You must also use the custom types where I have supplied them (e.g., uint8_t, uint16_t, uint32_t) unless you need a signed typed or one I haven’t supplied. You will lose points if you don’t follow this convention.

Once complete, your solution should exactly match my screenshot below. When you feel everything is working, submit **ONLY the SUBMIT-ME.zip** file to Canvas for this assignment and nothing else. I will download and test your solution on my QEMU system. It will need to work on my computer to receive credit, so I HIGHLY recommend you use the environment I show in class.

Additionally, you must attest and add the following honor statement to your Canvas submission (in the Canvas submission comments):

I, <state your name>, attest that this submission is my own work. I have not worked with anyone else on my implementation and have not used AI or other unpermitted sources to help me with my solution, in accordance with our definition of class cheating shown in Lecture 1.

Any submission that does not include this attestation will not receive any credit.



The screenshot shows a QEMU window titled "QEMU" with a "Machine View" tab. The main display area is black with white and blue text. It says "Welcome to CSCN-443!" in white, followed by "Bootloader-Stage2" in blue. Below this is a memory map showing hexadecimal addresses from 00 to ff in a grid. The addresses are listed in two columns: 00-1f and 20-3f on the first line, 40-5f on the second, 60-7f on the third, 80-9f on the fourth, a0-bf on the fifth, c0-df on the sixth, and e0-ff on the seventh. The text is in a monospaced font.