

CSE231 – Operating Systems

Assignment 5

Task 1


Writeup

Name: Ansh Arora

Roll no: 2019022

Task: Implement a simple bootloader which boots via legacy BIOS and first switches to protected mode and then displays a Hello World! with the contents of the CR0 register.

Makefile

```
Q1 >  Makefile
1 | make_bin:
2 |     nasm -f bin boot.asm
3 |
4 | run_bin: make_bin
5 |     qemu-system-x86_64 boot
6 |
7 | clear:
8 |     rm boot
```

The make_bin command first creates a binary using the nasm assembler. The run_bin command is then used to run the created bootloader binary in the QEMU emulator. The clear command can be used to reset the whole program.

Code Description and Logic

Line by Line explanation-

The bits 16 line is used to tell the compiler that the code being compiled is a 16 bit one. `0x7C00` offset declared is used to tell the BIOS to transfer control to the bootloader. This happens if the last two bytes of the code equal `0xAA55`. So for this, we add 510 zeroes in the end of the code, followed by these two bytes as can be seen in the code. The line `0x15` is used to enable the A20 bit, so that we can access more than 1MB of memory. This is important if we are going to use the 32 bit protected mode. We also set the `0x10` value to put the VGA into the safe mode in the initial boot section. After this, we make a call to the `protected_mode` section of our code, which puts the program into the protected mode. This is done by changing the value of the `cr0` register, which is done as pointed out in the code. We setup a global descriptor table (which I have hosted in a different `gdt.asm` file). The `lgdt` command is passed the value of the `gdt` pointer so that we can access the `gd` table. This is important if we want to read in 32 bit instructions. The GDT is declared. After this, we can simply move into the 32 bit section of our assembly code, which has also been demarcated. We first point all out segment register to point at data segment of the GDT, which is done by first passing it to `ax`, and then moving the same value to the other registers as well in the `set_segment_registers` section of the code. We are now ready to display stuff in 32-bit mode on the initiated VGA display. For this, we first move to the `say_hello` section of the code. The hello world display works in the following manner – first the value to be displayed, which in this case is "Hello world! Value of `cr0` register: ", is passed to the `esi` register. This is

done because the load string byte instruction `lods`, which is used to print onto the screen in the loop character by character accesses this particular register. This instruction loads individual characters into the `al` register. THE `0x1F00` code informs the VGA that the selected character has to be displayed in white colour(F) with a blue background(1). The loop thus continues to run until the `or al,al jz` command becomes true, which basically happens when `al` is zero, which in turn happens when the string that we needed to print has finished printing by the loop. Thus the loop keeps working until the whole value has been successfully printed to the screen. When `jz` comes true in our code, we jump to the `regs` section, which basically prints out the value of the `cr0` register in ASCII value. For this, we move the values of the `cr0` register into the `eax` register, then write it to the `vga` text buffer, first by giving it the `0x1F00` value for the white text on blue background, and then passing this variable into the `[edx]` location, where the VGA memory buffer location(`0xb8000`) has been memory mapped. On printing this, we finally shift our code to the `halt` section, where the process is interrupted and halted.

Booting Instructions

You can simply run the `makefile` commands given above to boot the bootloader assembly code that we have written into the QEMU emulator. On it's running perfectly, we are greeted by the following screen.



References

1. <http://3zanders.co.uk/2017/10/16/writing-a-bootloader2/>
2. <https://stackoverflow.com/questions/15832893/printing-a-new-line-in-assembly-language-with-ms-dos-int-21h-system-calls>
3. <http://www.on-time.com/rtos-32-docs/rttarget-32/programming-manual/x86-cpu/protected-mode/descriptors-and-descriptor-tables.htm>
4. https://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf
5. <https://medium.com/@g33konaut/writing-an-x86-hello-world-boot-loader-with-assembly-3e4c5bdd96cf>