CS3101: Operating System Lab

5th August, 2025

Lab 1: Booting Unraveled

General Instructions

- Assignments should be submitted through a shared link.
- Assignments will not be accepted after the due time.
- In case of plagiarism, zero marks will be awarded.
- Proper indentation and appropriate comments (if necessary) are mandatory in the code.

In today's lab session, you will learn about how a computer boots, and write a dummy operating system.

Booting Unraveled

When a computer starts, a special program called the Basic Input/output System (BIOS) is loaded from a chip into the main memory. The BIOS detects connected hardware devices, resets them, tests them, etc., and also looks for the special sector (the boot sector) on available disks to load the operating system. The BIOS reads the first sector of each disk (one by one) and determines whether it is a boot disk (a disk with an operating system). A boot disk is detected via a magic number 0xaa55, stored as the last two bytes of the boot sector of a disk.

Convert assembly (mnemonics) code to binary using the following:

```
$nasm boot_sect1.asm -f bin -o boot_sect1.bin
```

If you want to see what is exactly inside the bin file, the following command will help you:

```
$od -t x1 -A n boot_sector1.bin
```

The above binary can be used to set up (copy to) the first 512 bytes (the boot sector) of a disk. Instead of writing this boot sector to a physical hard disk, we can use an emulator.

QEMU is a system emulator that provides a simple and nice method to run your boot sector directly from the bin file.

```
$qemu-system-i386 boot_sector1.bin
```

The above command emulates a system using the file provided as the attached disk (which in our case has the first 512 bytes of interest).

Problem 1a

The `boot_sector1.asm` file, in the myos directory, shows a sample assembly code that is supposed to do something. The idea is that this program produces machine instructions that would be copied onto the boot sector and the computer powered-on.

Task: Compare the outputs of the booting process using the two programs, `boot_sector1.asm` and `boot_sector2.asm` , and justify your results.

Submission: Binary files and screenshots of QEMU along with an explanation.

Problem 1b

Let's do something slightly more interesting. On boot, our custom OS should print out a custom message. Write a program, message.asm, that prints a custom message of your choice on the screen during boot-up.
To print a character on the screen, use the following code with appropriate repetitions and changes:

mov ah , 0x0e ; set `tele-type` mode (output to screen)
mov al, 'C' ; one ASCII character hex code in register AL
int, 0x10 ; send content of register to screen via an interrupt

Task: Set up message.bin as the input file for QEMU to use for booting and test output (capture screenshot in message.png).

Problem 1c

Modify message.asm to create a simple bootloader that loads a secondary stage bootloader from disk and prints a message from the secondary bootloader.

1. Task: Write a primary bootloader (bootloader.asm) that loads a secondary bootloader (stage2.asm) from a specific sector on the disk.

• The primary bootloader should print "Loading Stage 2..." and then load and execute the secondary bootloader.

• The secondary bootloader should print "Welcome to Stage 2!" on the screen.

2. Steps:
• Create a disk image with both bootloaders using the dd command or similar.
• Use QEMU to test the booting process.

3. Submission:
• Assembly source files (bootloader.asm and stage2.asm).
• Binary files (bootloader.bin and stage2.bin).
• Disk image.
• Screenshots of QEMU output and explanations of each step.

Problem 2

Create a Dummy Operating System

1. Task: : Develop a dummy operating system that initializes a simple shell environment. The shell should:
• Display a welcome message and a prompt.
• Accept user input for a few basic commands (e.g., echo, clear, help).
• Implement basic functionality for these commands.
• Continuously run, accepting new commands until an exit command is given.

2. Steps:
• Use assembly to write the initial bootloader (os boot.asm) that loads a kernel (kernel.c) from disk.
• The kernel should be written in C and provide the shell environment.
• Create a disk image that contains both the bootloader and kernel.

3. Submission:
• Assembly source file (os boot.asm).
• C source file (kernel.c).
• Binary files (os boot.bin and kernel.bin).
• Disk image.
• Screenshots of QEMU output showing the shell in operation.
• Explanation of how each component works together to form the dummy operating system.