OS LAB1: BOOTLOADER

Miro Mannino

02 September 2022

LAB1 OVERVIEW

Write a bootloader: bootblock.s

- Set up and start running the OS
- Written in x86 assembly

Implement a tool to create a bootable OS image: createimage

- The bootable OS image contains bootloader and kernel
- File structure: become familiar with ELF format

LAB ORGANIZATION

There will be a lab recitation for every lab. Lab submissions will be done in two phases:

- 10-15 minutes to present your design of the project
  - Due date one week before the final project due date
  - You can use flow charts or pseudo code for demonstration
  - By this time you should have very clear idea on what you are going to do.
- Code submission:
  - To submit: `https://www.dropbox.com/request/89Wdp1n7FlPwJYi97qdg`
  - Format: Single zip folder LAB1-netId1-netId2-netId3.zip
  - Include inside README.md with some explanation about the code, the decisions, and the weakness of your solution.

Deadlines:

- Design Review: Sep 9, 2022, 3-5pm
- Due Date Sep 20, 2022, 11:55pm

BOOT PROCESS

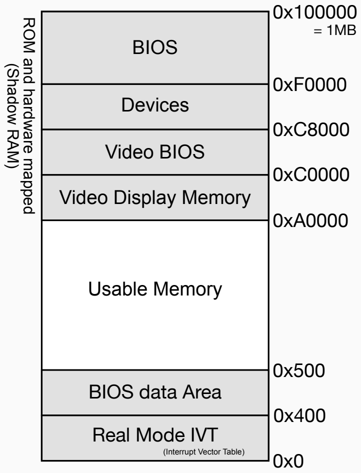
BOOT PROCESS: BIOS

The RAM is empty on startup.

The x86 processor begins executing at the physical address 0xFFFFFFF0.

This address is mapped to a ROM chip that contains the Basic Input/Output System (BIOS) code.

The BIOS performs tests and initializations. Eventually, the BIOS starts the boot loading.

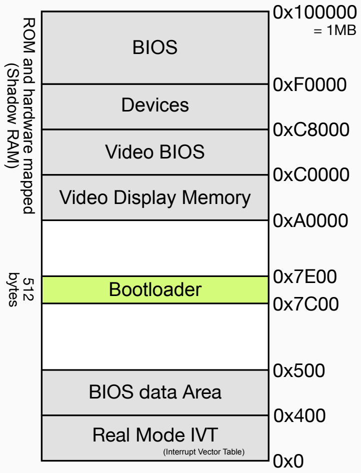


BOOT PROCESS: BIOS BOOT LOADING

The BIOS searches for an available storage media to boot from.

It loads the first sector (i.e. Master Boot Record) of the first bootable device into RAM at 0x7C00.

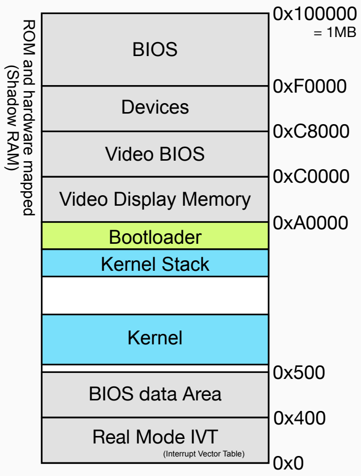
The BIOS transfers control to the bootloader (i.e. JMPs to address 0x7C00).



BOOTLOADER

The bootloader task is to:

- Load the kernel in memory and initialize its stack
- When that is done, transfer control to the kernel
- The loaded kernel can be large enough to overwrite the bootloader. Before loading the kernel it is better to relocate the bootloader to a higher address so it can't be overwritten.



BIOS SERVICES

The BIOS provides services callable through the INT instruction.

For example:

- `INT $0x10` for video services
- `INT $0x13` for disk services
- `INT $0x16` for keyboard services

Each service category has different functions that can be called. For example, function 14 is for displaying a character. The function number goes to register %AH. For example, to print:

```
movb 'A', %al                   # AL = code of character to display
movb $0x0E, %ah                 # Calling BIOS function 14 (print)
movl $0x0002, %ebx              # %bh = DisplayPage = 0,
                                # %bl = Green Color = 0x02

int $0x10                       # Calling BIOS.Teletype

# https://en.wikipedia.org/wiki/INT_10H
```

BIOS SERVICES FOR BOOLOADER

The BIOS provides functions for disk access using the interrupt `INT 0x13`.

Two families of BIOS functions for disk access:

- Using Cylinder, Head, Sector (i.e. CHS), function 2 in this case
- Using Logical Block Addressing (i.e. LBA)

We will use function 2

BIOS INT 0X13

BIOS function 2 to read from disk:

- %ah: register set to 2 to call this function
- %al: is the number of sector to read
- %ch: plus bits 6-7 from %cl: cylinder number
- %cl: bits 0-5: sector number
- %dh: starting head number
- %dl: drive number (e.g. 0 is floppy disk A:, 3 is drive C:)
- %es:%bx: pointer to memory region to place data read from disk. (remember the segment:offset notation)

cy7	cy6	cy5	cy4	cy3	cy2	cy1	cy0	cy9	cy8	se5	se4	se3	se2	se1	se0
			%ch								%cl				

Returns a result in %ah: 0 if successful, 1 if an error occurred.

More info and examples: `https://en.wikipedia.org/wiki/INT_13H`

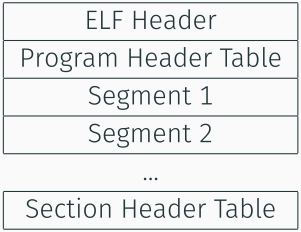
ELF FORMAT

Executable and Linkable Format created after compiling and linking.

It contains all the code compiled and in a specific format readable by an operating system.

But, what we want instead is to extract the code from it to use as Kernel.

You can use readelf to display info about any program headers. For example, `readelf -l touch`



CREATEIMAGE

createimage.given extracts the code from the ELF file. But, during Lab1 you will need to implement createimage to do the same.

- Study the ELF format
- Remember to pad the code to a complete sector (512 btyes), otherwise, two adjacent segments in the executable file may not be contiguous when loaded into memory.
- Mark the image bootable (i.e. write 0x55 0xAA at the end of the sector)
- Compare the behavior of your implementation with crateimage.given
    - Implement `--extended` to print info

Questions?