IN3000/4000 – Project 1

Bootup Mechanisms

Don’t panic!

Bootstrapping

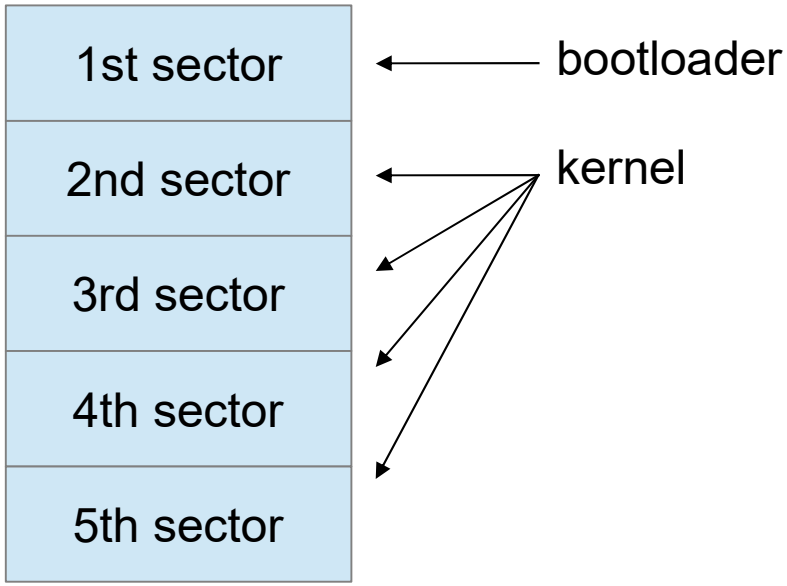
1. Power up computer
2. CPU performs self-test
3. It jumps to ROM code
4. ROM code initializes the hardware
5. Loads in bootloader
6. Jumps to bootloader
7. Loads in OS
8. Jumps to OS
9. OS runs
10. OS crashes...

Where is the bootloader?

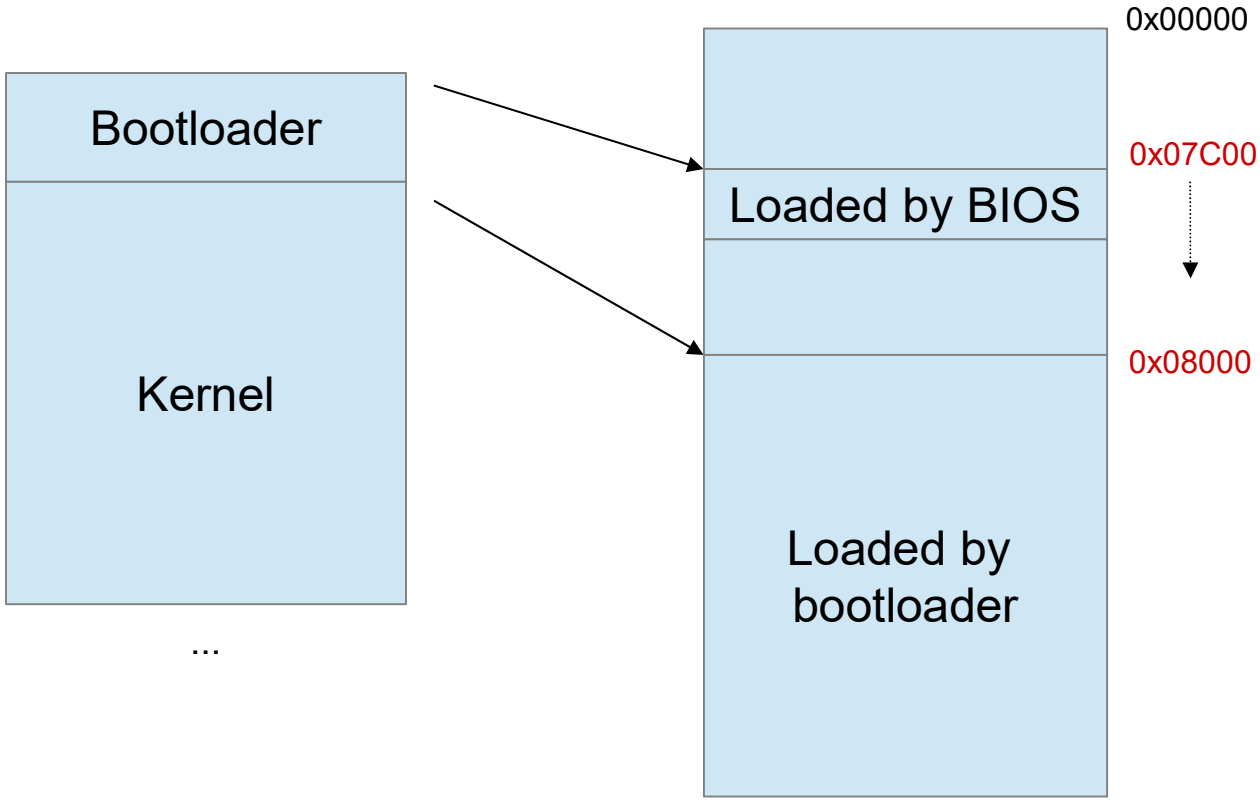
- Modern BIOSes can usually boot from floppies, hard drives, CDs, USB drives, etc.
- located at the first sector (512 bytes only)
- What if it needs to be bigger?

Our USB flash drive

- The bootloader is in the first sector
- The kernel starts at the second sector
  - how many sectors, depends on kernel size



From disk to memory



From C to executable

- Compile from high-level language to assembly
- Assemble from assembly to relocatable object file (object) containing machine code
- Link to put different objects and libraries together and resolve addresses between them

What's wrong with that?

- The linker creates an ELF file by default
- ELF includes headers and meta-information in addition to the executable code
- When there is an OS, it knows how to handle it
- When there is no OS, there should be no ELF headers either!

What to do then?

- Make a C program that takes out the headers

Your assignment (1)

- **bootloader**
  - you should only make changes to *bootblock.s*
  - Make sure it
    - loads the kernel from the USB flash drive
    - sets up the stack for the kernel
    - jumps to the start of the kernel
    - (the kernel does the rest)

Details

- Set up stack and data segment registers
- Get the size of the kernel from a hard-coded location
- Read the kernel starting from the second sector of the USB flash drive and put it into memory starting at 0x8000
- Set data segment register to kernel segment
- Long jump to the kernel code

Your assignment (2)

- **createimage**
  - only make changes to *createimage.c*
  - strip the ELF headers
  - make a bootable operating system image (put the bootloader and kernel on the USB flash drive)
  - always keep in mind one word: *padding*

Is it hard?

- Of course!
- A lot to understand first
- The assembly allows almost no debugging support at all (not even printf)
- Start early!