INF-2201-1 23V Operating system fundamentals

Pre-1

Amund Strøm

January 27, 2023

1 Introduction

When a PC boots a piece of code in the BIOS will run and try to read the first sector of all the drives the BIOS managed to locate, until it finds a bootable image. The image is loaded onto the memory and is responsible of loading the rest of the operating system onto memory.
In this assignment we are tasked to create such a bootable image. Our job is to create the bootloader, that is the piece of code that is first loaded onto the memory. And create a tool that creates a bootable operating system image on an USB disk, which contains the bootblock and the rest of the operating system.

2 Technical Background

Bootloader

The task of a bootloader is to load the rest of the operating system onto the memory. This is typically done after the BIOS has checked if the hardware is working properly. After these checks the BIOS will search for a drive with a bootloader. There are two things that identifies the bootloader. It will always be stored in the first sector of a drive, which is 512 bytes, and the last two bytes is identified by a magic signature, 0x55aa in hexadecimal. When the BIOS

eventually finds the bootloader on the correct drive, it will load the boot blocks onto memory and handover the control. Since the bootloader is only 512 bytes there is not much code to execute, but the only task is to prepare the memory and load the operating system onto the memory and handover control.

ELF file

The Executable and Linkable Format (ELF) is the product of a compiled program that contains data and headers that tells where and how the data should be handled. This format is typically used for executable files, relocatable object files, shared libraries, and core dumps. In this assignment we focused only on executable files, this file consists of the raw executable data of a program. Such a file will contain:

- a single ELF header that contains the metadata of the file
- Program header structure, zero or more describing memory segments that are only relevant during runtime
- Section header structure, zero or more describing sections only relevant during linking
- Data referred to by these header structures.

A simplified version of a simple hello world ELF file would look like this.

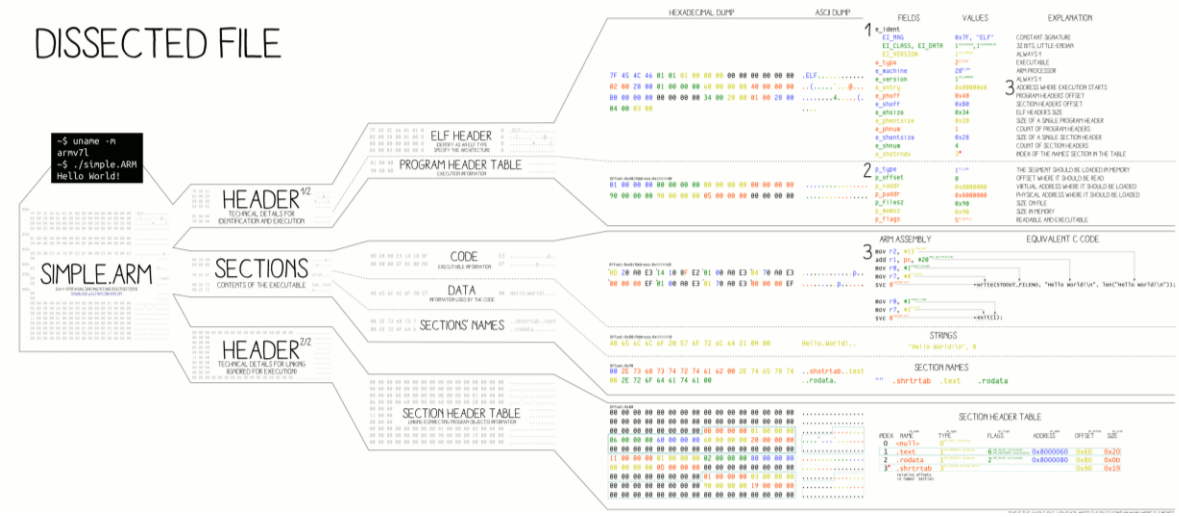

Figure 1: simplified version of ELF executable file.

This file contains more section headers than program headers, and the section header table is located at the end of the file. The section headers will contain metadata about the code and data used in this file, such as the offset value in bytes (how many bytes into the file the section starts), size of each section, flags and some other data.

3 Design and Implementation

There were two things that needed to be implemented, the rest was given as pre-code. This was the bootloader code and the tool that would create a bootable image to be put on a drive.

The bootloader has three tasks that happens in a mixed order, that is allocating memory space for the different segments, a BIOS interrupt call which is a functionality that invokes the facilities of the BIOS firmware. There are several different interrupt calls, but we are only interested in “INT 13h AH=02” which will read form a disk. And finally, a long jump that hands over control to the operating system.
First allocate space for the boot segment, stack segment and stack pointer. Then invoke the BIOS interrupt call, such a call requires some parameters on the different registers:

AH	02h
AL	Sectors To Read Count
CH	Cylinder
CL	Sector
DH	Head
DL	Drive
ES:BX	Buffer Address Pointer

Figure 2: INT 13h AH=02 register parametres.

When the correct parameters are set, we do the interrupt call. After this the final segment is allocated, the kernel data segment. Then perform the last task which is to do a long jump and hand over control to the operating system.

To create a bootable image, you need only the raw executable data of a file since we are working with very minimal space. Therefore, we compile the programs that we are supposed to put on the image and extract the important data. As described in the technical background section, an ELF file is divided into different parts. What we seek is the code and data sections. To access these, we need to know the offset value, which is acquired by sequence of events.

First read the ELF header which is always at the start of the file. The header contains a bunch of different metadata, such as the offset value of the section header table, how many sections there are, and how large the sections are. By using this information, we can acquire the metadata for each section. Again, using the offset value and size of each section, we extract data from the code and data section, also known as the “.text” and “.data” section respectively and place it on the image.

The image also requires correct padding to differentiate the parts of the image. The bootloader will always be at the first sector (512 bytes), and the last two bytes ends with a magic signature, 0x55aa in hexadecimal. Right after this we find the kernel code, or .text section, with padding ending at the address 0x0000127c in hexadecimal, with reasons I am unsure of. Lastly the kernel data or .data section is located at the end with some extra padding ending at address 0x000013ff in hexadecimal. This extra padding is to protect the image for not loading viruses onto the memory, since the bootloader will load everything before the ending address 0x000013ff.

4 Conclusion

The objective for this project was to create a bootloader and a tool that would create a bootable image containing the bootloader and a kernel. To complete this task, I needed to read up on BIOS interrupt calls and ELF files.

Acknowledgement

My code my share some similarities with Iver Mortensen, Tarek Lein and Henrik Haatuft.

5 References

[1] Wikepedia, «Executable and Linkable Format,» 23 December 2022. [Internett]. Available: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format.

[2] Wikepedia, «Bootloader,» 2023 January 2023. [Internett]. Available: https://en.wikipedia.org/wiki/Bootloader.

[3] Wikepedia, «INT 13h,» 23 January 2023. [Internett]. Available: https://en.wikipedia.org/wiki/INT_13H.

[4] Wikepedia, «BIOS interrupt call,» 25 December 2022. [Internett]. Available: https://en.wikipedia.org/wiki/BIOS_interrupt_call.