

Project Title:

**Bootloader**

Submitted By:

**Hamza Ikram 2020-CE-06**

**Muhammad Shaff 2020-CE-07**

Submitted To:

**Ma’am Darakshan**

Course:

**CMPE-331L: Operating System**

Semester

**5th**

Date:

**23-12-2022**

**Department of Computer Engineering**

**University of Engineering and Technology, Lahore**

**Table of Contents**

**Problem Statement 3**

**Bootloader 3**

**Explanation 3**

**Working 3**

**Task 4**

Protected Mode 4

**Technologies Used 4**

Languages 4

Tools 4

**Building 4**

print.asm 4

disk\_load.asm 4

gdt.asm 4

print\_pmode.asm 5

switch.asm 5

boot.asm 5

**Flow Diagram 6**

**Conclusion 6**

**Results 7**

**References 8**

**Problem Statement: -**

Our task is to design the bootloader for the x16-bit architecture.

**Bootloader: -**

A boot loader, also called a boot manager, is a small program that places the [operating system](https://www.techtarget.com/whatis/definition/operating-system-OS) (OS) of a computer into [memory](https://www.techtarget.com/searchstorage/definition/memory-card).

**Explanation: -**

When a computer is powered-up or restarted, the basic input/output system ([BIOS](https://www.techtarget.com/whatis/definition/BIOS-basic-input-output-system)) performs some initial tests, and then transfers control to the Master Boot Record ([MBR](https://www.techtarget.com/whatis/definition/Master-Boot-Record-MBR)) where the boot loader resides. Most new computers are shipped with boot loaders for some version of Microsoft Windows or the Mac OS. If a computer is to be used with [Linux](https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system), a special boot loader must be installed.

**Working: -**

After a computer is turned on, information about the installed hardware comes up on the screen. The bootloader places its operating system into the memory. The basic input/output system (BIOS) carries out tests before transferring control to the Master Boot Record (MBR), which contains the boot loader.

A lot of bootloaders are configured to give users different booting options. The options include different operating systems, different versions of the same operating system, operating system loading options, and programs that run without an operating system.

In certain cases, a device may have two operating systems. Bootloaders can be used on these devices to start the correct operating system that users prefer automatically. A bootloader can also be used to boot the operating system into safe mode for recovery.

You can use a bootloader to boot into a program without having to start the operating system. This can be useful with devices such as game consoles. After the game disc is inserted into the console and the console is turned on, the user is taken straight to the game instead of the welcome screen.

**Task: -**

Our task in this project is to make a bootloader for the 16-bit architecture. The task of this bootloader is:

* It tries to read the hard drive in 16-bit mode
* It tries to read from floppy disk in 16-bit mode
* Then it shifts to the 32-bit protected mode
* And tries to the read from the disk

**Protected Mode:**

Protected mode is a 32-bit operating mode found on Intel 80286 or newer processors. It provides the access of addressing virtual memory, extended memory, and multitasking, while protecting programs from overwriting one another in memory.

**Technologies Used: -**

The knowledge, tools and the languages we are going to use to make this bootloader are:

1. **Languages:**

* C
* Assembly

1. **Tools:**

* NASM
* Qemu

**Building: -**

While making a bootloader we need to make different files for the different task of the bootloader.

1. **print.asm:**

The main purpose of this file is to print something on the emulator.

1. **disk\_load.asm:**

The main purpose of this file is to load data from the disk.

1. **gdt.asm:**

The main purpose of this file is to make code and data segments.

1. **print\_pmode.asm:**

The main purpose of this file is to print a message when code enters in protected 32-bit mode.

1. **switch.asm:**

The main purpose of this file is to switch from 16-bit mode to protected 32-bit mode.

1. **boot.asm:**

This is the main file of the program which allocates the memory and run the code at kernel level.

Also, there are some files which we don’t be able to built because we found them on google.

* **kernel\_entry.asm:**

The main purpose of this file is to make program enter in kernel.

* **kernel.c:**

The main purpose of this file is to allocate entry point after entering the kernel with the help of pointer.

Also, there are some other files as well.

1. **kernel.bin:**

This file is compiled from the kernel.c file.

1. **boot.bin:**

This file is compiled from the boot.asm file.

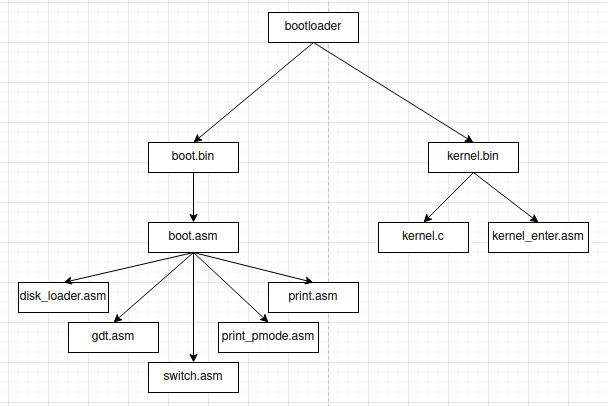
1. **bootloader.bin:**

This file is compiled from the above two files.

1. **Makefile:**

This file has all the commands needs to compile the above listed file.

**Flow Diagram: -**

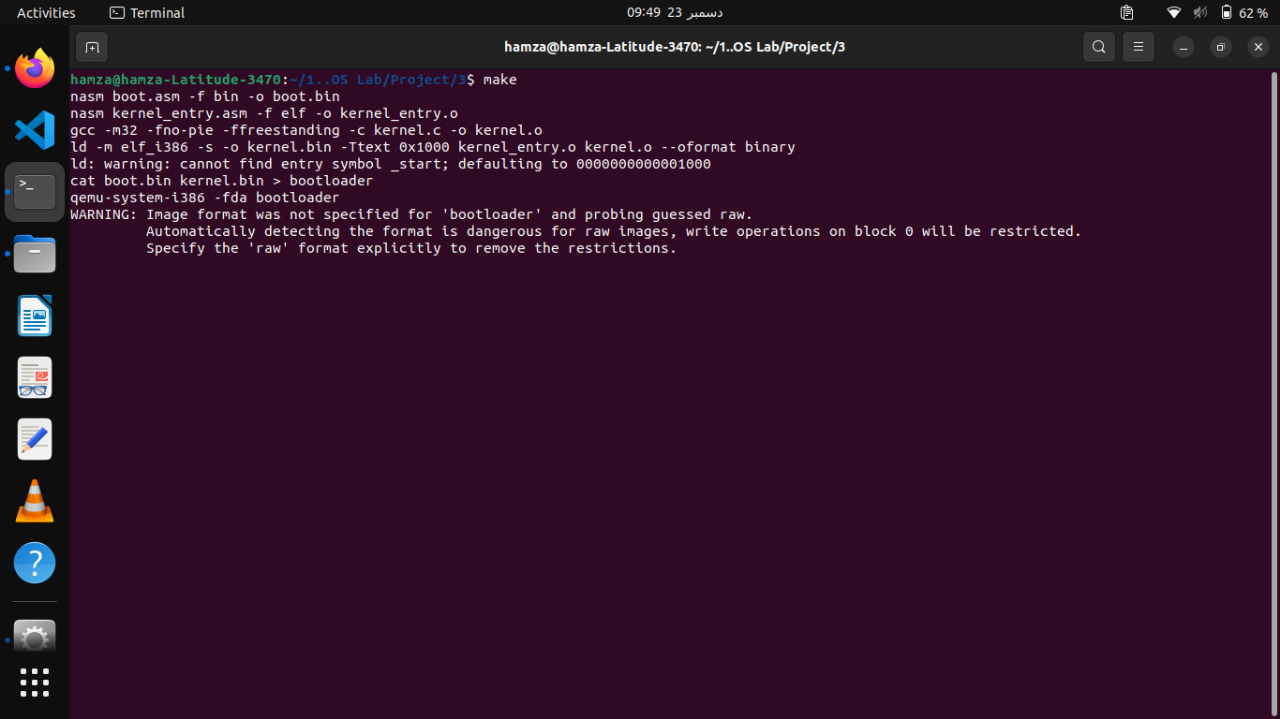


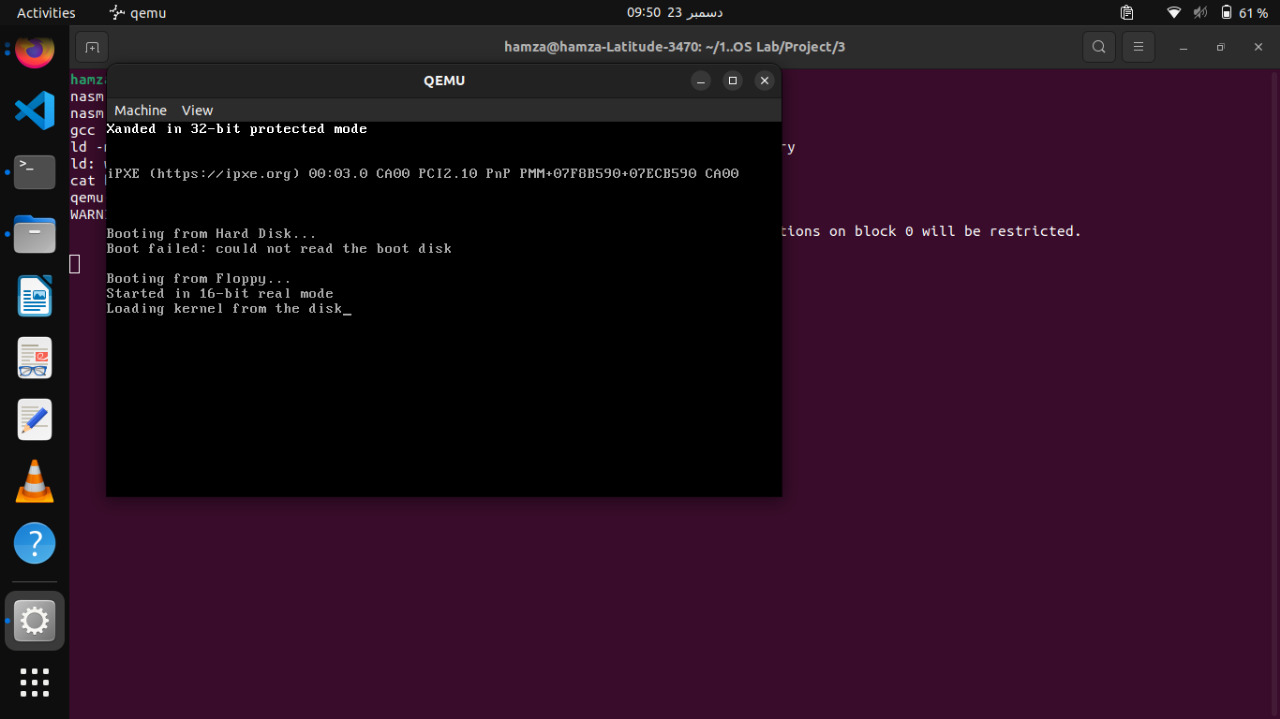
**Conclusion: -**

After making all these files we just need to make their binary file to test our project. Finally, we make our bootloader file and then we run the file on the emulator named QEMU. The Results we obtained after running the files are listed in results section below.

**Results: -**

The results of our implementations are given below:

****

****

**References:**

[1] <https://www.apriorit.com/dev-blog/66-develop-boot-loader>

[2] <https://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf>

[3] <https://www.youtube.com/watch?v=F3XiH78erNM>

[4] <https://www.youtube.com/watch?v=7LTB4aLI7r0>

[5] <https://www.youtube.com/watch?v=48dFP4PBHII>

[6] <https://www.ctyme.com/rbrown.htm>

**Useful Abbreviations**

|  |  |
| --- | --- |
| -f | Format of the file |
| -o | Outfile |
| elf | Format for any object file |
| cat | Concatenate files |
| -m32 | Specify 32-bit code |
| -ffreestanding | Standing environment |
| -c | Specify .c file |
| ld | Binary files and object files linker |
| -m elf\_i386 | Set emulation to 32-bit |
| -s | Strip all symbols |
| -Ttext | Put the text section of your program |
| --oformat | Outfile format |
| -fda | File directory |
| -fno-pie | Mode option |

**Useful Commands**

**nasm boot.asm –f bin –o boot.bin:**

It makes boot.bin file from boot.asm file.

**nasm kernel\_entry.asm -f elf -o kernel\_entry.o:**

It makes the object file from kernel\_entry.asm file.

**gcc -m32 -fno-pie -ffreestanding -c kernel.c -o kernel.o:**

It provides the free standing environment for the kernel.files and also makes kernel.o object file from it.

**ld -m elf\_i386 -s -o kernel.bin -Ttext 0x1000 kernel\_entry.o kernel.o --oformat binary:**

It makes the kernel.bin from the two object files.

**cat boot.bin kernel.bin > bootloader:**

It makes bootloader from boot.bin and kernel.bin file.

**Now, we just need to run the code in qemu emulator.**