School of Sciences and Engineering

Project Phase 3 Report

By:

Dalia El Naggar ID: 900191234

John El Gallab ID: 900193761

Kirolos Mikhail ID 900191250

CSCE 2303-02 Computer Organization & Assembly Language Programming

Spring 2021

Table of Contents

Assumptions

- We need to mask the PIC's IRQs at the beginning of setting up the IDT and unmask them at the end.

Approach Description

1. Simple Scrollable Video Driver

- The scrolling function works by copying all the lines starting from the second line till the last line to the line directly above the line. Then we clear the last line and move the cursor to the beginning of the last line.
- The clear screen function resets the cursor to the beginning of the screen and loops over the whole video memory and sets it to hold the value 0 which is a null character.
- We used the cited website as a reference. We then calculated the new location of the cursor and borrowed a couple of lines of said website to modify the location of the cursor.
    - https://wiki.osdev.org/Text_Mode_Cursor#Without_the_BIOS

2. Scanning PCI Devices Recursively

We decided to use a recursive approach to scan the PCI devices.

- We call a scan_pci_devices function which simply set the "bus" variable to zero and calls scan_bus to scan bus 0.
- Function scan_bus simply loops over all devices and functions of these devices to scan them using get_pci_device.
- Function get_pic_device loops over configuration addresses for this device to obtain the device's configuration space. If the device's ID is 0xFFFF, we skip it, else we load the

header in physical memory and print a dot to signify that a device has been scanned. We check the device's header; if it is 0x1, then the device is a bridge so we extract the secondary bus number and set the bus variable to it and call scan_bus. When it returns, we return the old value of the bus variable. If the device is not a bus, we check the device's class and sub-class; if they are both 0x1, then the device is an ATA controller so we call ata_copy_pci_header.

3. Setting up the IDT

- We first initialize 4K for the IDT using init_idt.
- We call setup_idt which configures the PIC, masks its IRQs, sets up the exceptions and the IRQs, loads the descriptor, configures the PIT, and unmasks the PIC's IRQs.
- Function setup_idt_exceptions sets the IDT entries for the 32 ISRs.
- Function setup_idt_irqs sets the IDT entries for the 16 IRQs.
- Function setup_idt_entry takes the interrupt number and handler address to set the IDT entry.
- Functions isr_common_stub and irq_common_stub call the registered routine. If there is no registered routine, they call the default. The only difference between them is that irq_common_stub sends EOI to the PIC at the end.
- Function register_idt_handler adds the handler's address in the handler array

4. Setting up PIC

- To begin with we need to clear the PIC, and we do so by firstly disabling the PIC (by loading 1 to all the IRQs. Then we set each of the 4 ICWs to their certain bits. Finally we unmask the IRQs.

- To setup the IRQ mask, we first check the data port, and if we are on the master, we read the IMR. We then shift 1 left by the IRQ value, and OR with the rax, and finally we write it on the data port back again.
- To clear the IRQ mask we do the same as in setting up, but instead of ORing we do not, so we can make all the bits 1, except the ones needed to clear. Finally, we write to the data port.

5. Setting up the PIT Timer

- We call on configure_pit right after we finish initializing and setting up the IDT. In that function we register the PIT handler in order to be invoked through interrupt 32.
- We then, using a simple equation, set the PIT to fire 50 interrupts per second and write this value to the data port of that channel.
- The function handle_pit is called whenever irq0 is invoked. We modified it by allowing it to print the pit counter only once every 1000 times.

6. Identifying ATA disks and load their parameters

- After we are done with scanning all the PCI devices, we start trying to identify the ATA devices if they are present
- We loop over the available channels to start checking the connected devices and whether they have class code 0x01 and subclass code 0x01.
- If it is the case, then we finish identifying the device by printing its attributes.
- The primary function that does these functionalities is the ata_identify_disk that refreshes the channel we are on then start issuing the identify command. If the device is busy or not

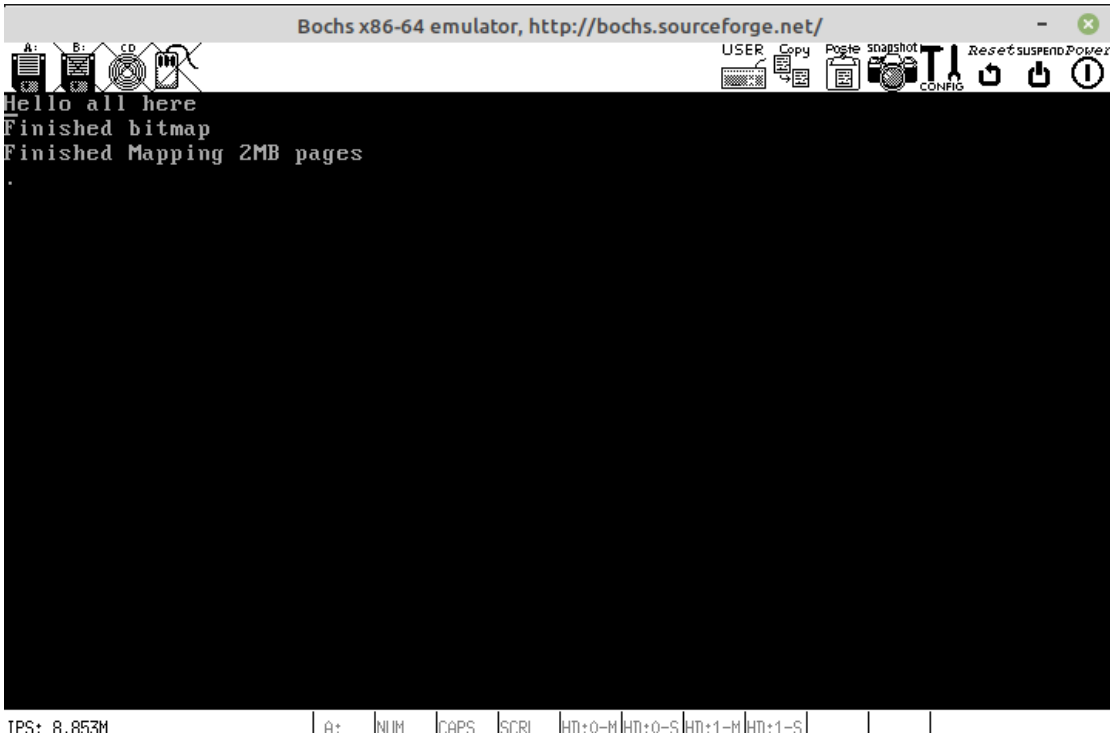
available then, we use a loop to keep on checking it out until it is ready/available. When it is ready, we start printing its attributes.

Steps to run the code:

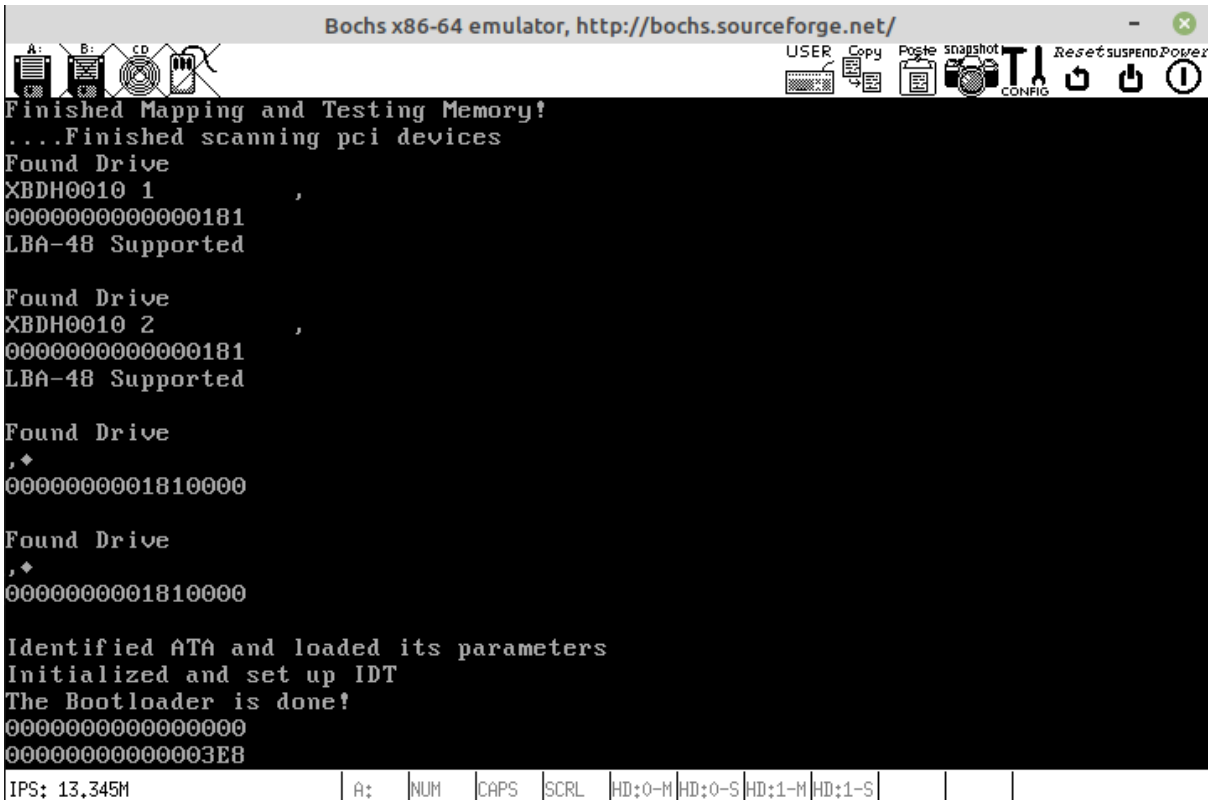
- To run the code, open the terminal in the directory that contains the “MakeFile” and type the command make run_myos.

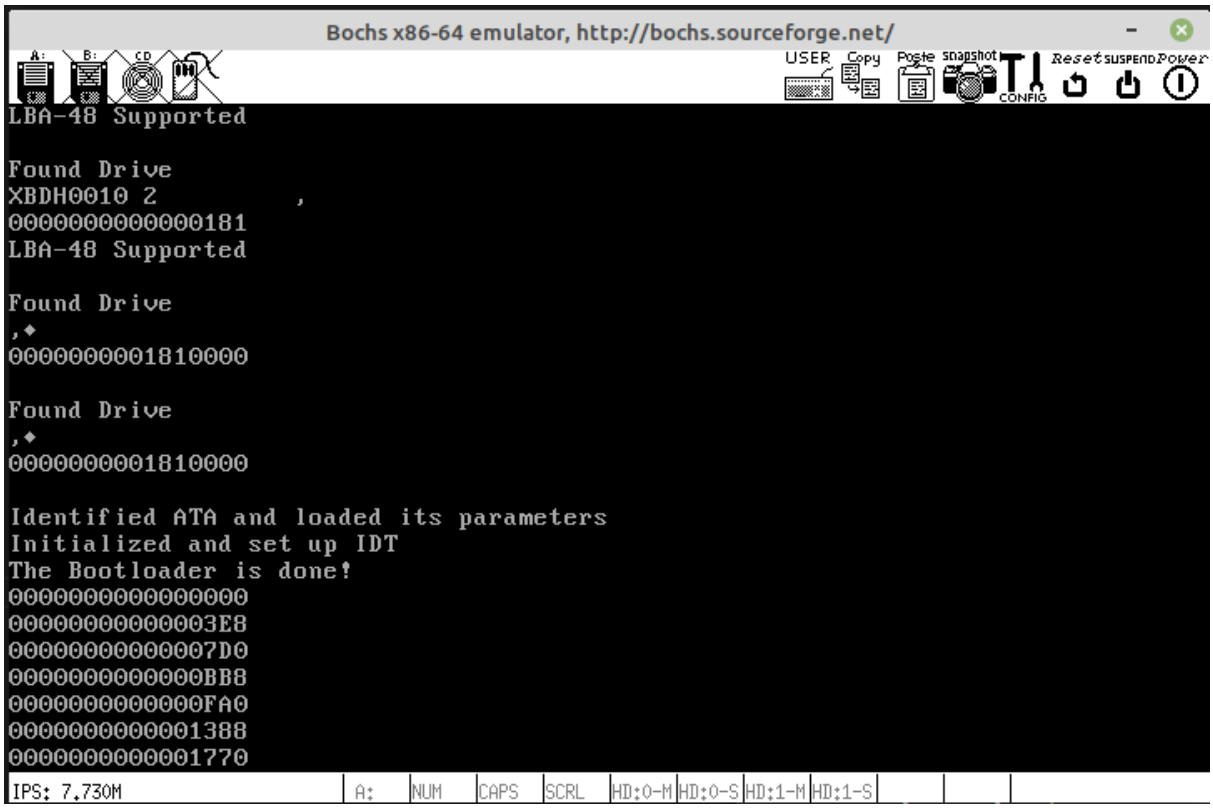
Results:

Screenshot before the screen is cleared:



Screenshot after screen is cleared:


Bochs x86-64 emulator, http://bochs.sourceforge.net/
A:
B:
CD
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
Finished Mapping and Testing Memory!
....Finished scanning pci devices
Found Drive
XBDH0010 1 ,
0000000000000181
LBA-48 Supported
Found Drive
XBDH0010 2 ,
0000000000000181
LBA-48 Supported
Found Drive
,◆
0000000001810000
Found Drive
,◆
0000000001810000
Identified ATA and loaded its parameters
Initialized and set up IDT
The Bootloader is done!
0000000000000000
00000000000003E8
IPS: 13.345M
A:
NUM
CAPS
SCRL
HD:0-M
HD:0-S
HD:1-M
HD:1-S


Bochs x86-64 emulator, http://bochs.sourceforge.net/
A:
B:
CD
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
LBA-48 Supported
Found Drive
XBDH0010 2 ,
0000000000000181
LBA-48 Supported
Found Drive
,◆
0000000001810000
Found Drive
,◆
0000000001810000
Identified ATA and loaded its parameters
Initialized and set up IDT
The Bootloader is done!
0000000000000000
00000000000003E8
00000000000007D0
0000000000000BB8
0000000000000FA0
0000000000001388
0000000000001770
IPS: 7.730M
A:
NUM
CAPS
SCRL
HD:0-M
HD:0-S
HD:1-M
HD:1-S