CSCE 231/2303 Fall 2018
Assignment 9: Memory Page Table and Long Mode

Assigned: Wednesday, November 28th in Class

Due: Tuesday, December 11th at 3:00 pm

No Delayed Submission Allowed

Goals

This assignment is a group assignment (2-3 members/group). The goal of this assignment is to write and implement a group of essential functionalities in x86_64 Long Mode. Precisely, a simple scrollable text mode video driver, scanning installed PCI devices, identify ATA disks and print their attributes, reading sectors from disk using the PIO method, and setup your PIT timer.

Details

In this assignment you will work on the third stage boot loader within the skeleton code tree. You should build on top of the work you have done in assignment 8. Initially, you can use the code snippets presented in the slides and build on top of them. You are required to implement the following five main functionalities:

1. **Simple Scrollable Video Driver:**
   In the code skeleton, you already have a file called **video.asm** that contains two main functions, bios_print_hexa and video_print. The first function name is misleading and erroneous as it is not bios anymore, so I suggest renaming it to video_print_hexa. The two functions writes directly to the video ram. Your first main target is to understand how those two functions work, and then you will need to modify/reuse them in away to allow scrolling in your screen console. As the screen space is finite (80x25 characters) you will need to throw away some old data from the screen to accommodate new data. You are also required to implement a new function that clears the screen and initialize the cursor at the left upper corner location.

   After clearing the screen your screen should be empty and the cursor should be at the upper most left corner location of the screen. Whenever print commands are issued to the screen the data to be printed should be appended to the video ram, which will automatically appear on the screen. If the video ram is full and you keep on advancing the cursor, you will be writing outside the video ram, which is not only dangerous for overwriting other data that you do not know anything about, but also what you are printing will not show on the screen.

As long as there is empty room on the screen you keep on appending data to the video ram. As soon as the video ram is full, you will need to throw away some of the content at the beginning of the ram to make room at the end which will appear as scrolling up the screen. Precisely, you need to free the last line in the video ram by throwing the first one away and copying the content of the video ram up one line; 80 characters. You are also required to handle new lines '\n' correctly and account of the empty spaces in each line accordingly. Moreover, you should be able to account for printing multiple lines at once. Of course, at any point in time the screen is cleared, you start from the initial state where the screen is empty.

Note: you can implement this functionality using a normal mov loop or you can use **MOVSB** coupled with **REP**.

2. **Scanning PCI Devices:**
   You are required to scan all available PCI devices and store their headers in some memory area for future usage. The way you should follow to perform such scan is fully explained and presented in class and you are allowed to use the code for scanning a single device. You have to study **pci.asm** and see how it is invoked from **third_stage.asm** and complete the missing parts to achieve the target functionality.

3. **Identifying ATA disks and load their parameters:**
   I already intentionally left this part implemented for you in **ata.asm** to read and learn from. It is very important to understand this part to be able to do functionality #4. You are required to complete the in-line detailed documentation. You will need also to modify the way the ATA disks attributes are printed to accommodate the new video driver scrolling features.

4. **Read disk sectors:**
   You are required to download the ~1 MB PCI database from this link https://pci-ids.ucw.cz/v2.2/pci.ids, package it into one of the available disks and read it into memory using PIO commands. You are required to add and implement a function in **ata.asm** that reads sectors from a disk based on some parameters (disk #, starting sector, # of sectors, memory location). You will have to use this function to read the sectors containing the PCI database file into memory. You will use the database file to lookup every PCI device that you have scanned in functionality # 2 using the **VENDOR ID** and the **DEVICE ID** and print their meaningful textual names.

   Note: We will discuss in the lab how you can package the database file into one of the VM virtual disks. This means that it is important to attend the lab.

5. **Setup the PIT Timer:**
   Finally, you are required to configure the pit with **mode 3** in pit.asm as explained in class and modify the pit handler routine to use the new video

driver and utilize scrolling instead of printing the pit_counter in-place. You will also need to modify the pit handler to print the pit_counter every 1000 interrupts instead of on every interrupt.

What to submit

1. Your full in-line documented third stage assembly code for all code added to the code tree by you and for already existing undocumented code; especially functionality #4. **It is very important to highlight that you are not allowed to copy the code documentation presented in the slides, and you need to explain the code in your own words. If you copy the documentation you will get ZERO in the assignment.**
2. All the skeleton code with your updates must be submitted on BlackBoard.
3. A PDF report that includes:
    a. A detailed description of any assumptions you have made.
    b. Detailed description of the approach and the steps you have adopted and followed for each functionality.
    c. List all findings that you have came up with from doing this assignment.
    d. The steps needed to run your code.
4. A readme file indicating how to compile and test your code.

IMPORTANT: each group will need to present their work and conduct a demo in front of the professor on the due date. Each group will have 20 minutes to show their work. Each group will need to reserve a time slot in the duration from 4:00 PM – 9:00 pm on this google sheet https://docs.google.com/spreadsheets/d/1ydJLaoTlWZxl-nev6E_r7E63v9Damj9yxLTR2PHd2Mg/edit?usp=sharing, and each group will have to send the professor an email with the time slot as well.
NO DEMO NO GRADE.

How to submit:

Compress all your work: source code of full skeleton source tree, report, readme file, and any extra information into a zip archive. You should name your archive in the specific format <Student_ID>_<Name>_Assignment9.zip. Finally, upload your code to blackboard.

Grade

This assignment is worth 10% of the overall course grade. The assignment will be graded on a 100% grade scale, and then will be scaled down to the 10% its worth. The grading of the assignment will be broken down as follows:

1. 10% for just submitting a meaningful assignment before or on the due date. This 10% does not account for the correctness of your assignment but submitting an empty assignment without code will definitely results in loosing this 10% and consequently the whole grade of this assignment.
2. 70 % for the correctness and the quality of your code.

3. 20 % for the quality of your inline documentation, the report, and the readme file.

Delays

No delays are allowed. **ZERO GRADE IF NOT SUBMITTED ON TIME.**