Project 1: Getting Started

Due on 1/25, 11:59 pm CT

Introduction

The objective of this project is to test your development environment. You are provided a simple "kernel", which essentially prints a welcome text and goes into an infinite loop. You are to modify the text on the welcome message to print out your name.

The "kernel" source code for this do-nothing kernel consists of a small collection of source files:

`kernel.C`: This file contains the main entry point for the kernel. For this MP, this is where you need to apply the modifications in order to have the kernel print out your name after the welcome message.

`start.asm`: This assembler file contains the multiboot header, some initial setup, and the call to the main entry point in the kernel.

`utils.H/C`: A selection of utility functions, such as memory copy, simple string operations, port I/O operations, and program termination.

`console.H/C`: Access to the console video output.

`makefile`: Used to easily compile everything to generate the `kernel.bin` file. (There are several versions of the makefile, targeting different development environments.)

Your task is to modify the provided kernel, compile it, and generate a floppy disk image that will boot your kernel. You also need to set up your developing environment,your source revision control repository, and get prepared for the debugging activities needed in the upcoming projects.

The first step is to setup your development environment, i.e., a place where you will find all tools necessary to build your kernel.

Step 1: Setup your Development Execution Environment

We will be using *VirtualBox* to run a fully configured image of Ubuntu Linux with all the development environment tools pre-installed. The image is 64-bit, so it is required to have a processor capable of 64-bit guest OS emulation. Note that the 64-bit requirement does not mean your host image has to be 64 bit. You can have Windows 7 32-bit and still have a 64-bit processor. In the unlikely scenario that your machine does not meet the requirements for VirtualBox, please contact the instructor by e-mail as soon as you find it out. *VirtualBox* is available for the Windows, Linux, and Mac OS environments.

Find the folder with the provided documentation and files at the course's Google Shared Drive. The course Shared Drive is named "CSCE 410 - Students

- Spring 2020". In the sub-directory 'Projects/P1' you will find the 'Resources' sub-directory and a zip file with code.

In Resources, you will find the document "VirtualBoxSetup.pdf". You can follow the instructions in the document to setup your VirtualBox with a Linux virtual machine.

An alternative to using VirtualBox is to work with the Bochs emulator directly in your machine. It is somewhat rare that a student chooses to work on Bochs. Please take a look at that document "Bochs.pdf" in the Resources sub-directory to get an idea of how the emulator works.

The Resources documents are also available on eCampus.

Step 2: Setup your Source Revision Control Environment

You are required to set up a **private** github repository for this class at `http://github.tamu.edu`. You need to name your course repository as CSCE410-2020-A. Your source must be in a sub-directory named P1. Authentication is done using your netid. If you are not familiar with github, there are plenty of good tutorials out there.

The history of your git actions is a documentation of your development process. All intermediate versions of your code need to be captured in the repository. This means that you will be committing your code to the repository often, at least once at any of your working sessions. Even temporary changes (e.g., adding or removing statements for debugging) should be committed. Recall that you do not need to be connect to the internet to commit your code to the repository; the commit operations are captured locally until you push your changes to the repository.

In Project 1, there is nothing much to document: you start by creating a directory for P1 (naming it as specified above) and adding all the provided files you will need there. As you fix problems (e.g., if the provided Makefile did not work in your environment) and develop code (e.g., change `kernel.C` to include your name), you commit new versions of files to the directory. For each action, use meaningful explanations in your commit messages. Use P1 as an opportunity to make sure you know how to commit your changes and push them to your github repository, with appropriate commit messages.

Introducing lots of code and lots of changes in a single *git* action is not, in general, a good software practice. For our projects, it is not an acceptable practice. Again: you need to commit your work (and push it to your remote repository) often. If the project submission system on eCampus is not working you won't need to sweat about proving that you finished the project within the deadline: the git history will prove that you had concluded it.

Your git activity will be used to assess your software development practices and it will have an impact on your project grade. Be aware that the GitHub history may also be used as evidence of your behavior in the development of our code, in particular if your code is flagged as plagiarism by tools that assess code similarity

Step 3: Finally – doing the Assignment

Now you should be ready to change `kernel.C`:

- Download and unzip the provided file `P1.zip` into your development environment
- Look at `README.TXT`
- Work on `kernel.C`

After modifying the provided kernel file, compile it, and copy the resulting `kernel.bin` binary file onto the provided floppy image file, called `dev_kernel_grub.img`. The provided zip file contains a shell command to do this, you invoke it as follows:

```
> ./copykernel.sh
```

This shell command **mounts** the image as a disk, **copies** the file `kernel.bin` onto the mounted disk, and then **unmount** the disk image. In the past some students had to use a lazy umount to get things to work, so as usual, when things don't work, ask around (and in our Piazza forum), look around, and find a way to make it work. Remember to check the course discussion forum, and if you see people stuck with problems that you solved, share your knowledge.

Now you are ready to run your own kernel. In order to avoid the difficulties of debugging your code in a bare metal hardware, you are going to run your kernel in an **emulated** or **simulated** environment, on top of your normal development environment. Many emulation or virtualization environments can be used to accomplish running your "small" kernel on top of the "real" kernel managing the machine you are using for this project. In the provided development environment is set up for your to use Bochs. As mentioned before, utilize the makefile to build your kernel binary. Then call the copy kernel script to copy the `kernel.bin` onto the boot drive image. Finally, use bochs to boot your kernel.

The Assignment

You are to modify the given "kernel" to print out your name on the welcome screen. For this, you modify the provided file `kernel.C`. You then compile the source to generate the kernel executable `kernel.bin`. Preferably you do this by invoking the `make` command. You then copy the kernel onto the provided `.img` file, as described in Step 3.

After testing your code with the Bochs emulator (the one that is ready for you on VirtualBox or the one you build yourself, if you are not using VirtualBox), you rename your `.img` file to `p1.img` and compress it into a ZIP file called `p1.zip`. You are to turn in the ZIP file on eCampus.

Failure to follow the handing instructions will result in lost points, as it hinders the grading.

Project Grading Criteria for MP1

- Your kernel boots: 50%.
- Feature completeness and functional correctness: 50%. (This means your kernel prints the expected welcome message.)
- **If you did not follow the github requirement specified in Step 2, your grade is zero.**

Extra Bonus for P1 - 10 points

As you will experience, debugging operating system code may be tricky. You can position yourself for an easier time later on by working now on setting up a debugger in your environment.

The Bochs handout has information on how to set up and use Bochs with support for `gdb`.

If you pursue this integration successfully, submit as part of your zip file a document with a short description of how you accomplished the integration. Also include a picture of your `gdb` tool in action.

Helpful Links

https://www.cs.princeton.edu/courses/archive/fall09/cos318/precepts/bochs_gdb.html https://www.cs.princeton.edu/courses/archive/fall09/cos318/precepts/bochs_setup.html http://bochs.sourceforge.net/doc/docbook/user/debugging-with-gdb.html http://heim.ifi.uio.no/~inf3150/doc/tips_n_tricks/bochsd.html http://wiki.yak.net/746