Machine Problem 1: Getting Started

Introduction

The objective of this machine problem 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 of 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, and program termination.

`simple_console.H/C`: Primitive access to the console video output. [1]

`makefile`: Used to easily compile everything to generate the kernel binary (the `kernel.bin` file). (The makefile is set up to work both on Linux and on Mac OSX/Silicon.)

Your task is to modify the provided kernel, compile it, and generate kernel binary that can be used to boot your kernel.

The first step is to setup your development environment, i.e., a set of tools necessary to build your kernel.

Step 1: Setup your Development Execution Environment

To generate the kernel binary, you will need an assembler and a compiler for the x86 and the associated toolchain (linkers and stuff) to generate the kernel binary. We will use QEMU to run the binary in an emulated environment. These tools can be easily installed on Linux and on Mac OS. Check out the *Machine Problem Resources* page on the course's canvas site for details on how to install the required software. Windows users will need to install a Linux environment. This can be done by (a) setting up a virtual machine, for example using VirtualBox, and installing Linux on it, or by (b) using the Windows Subsystem for Linux (described in Microsoft's WSL Documentation).

Step 2: Doing the Assignment

Now you should be ready to change `kernel.C` and run your modified kernel:

- Read the provided information under Machine Project Resources on the course canvas site.
- Setup your development environment.

[1] The simple console prints to a separate "graphics" screen. In future MPs, we will use a more sophisticated console, which will allow us to redirect the output to stdout. This will allow you to scroll back and cut-and-paste the output, which may come useful when you debug your code.

- Download and unzip the provided code in file `MP1_Sources.zip` from canvas to your development environment.
- Look at `README.TXT`
- Work on `kernel.C`

After modifying the provided kernel file, compile it. This is very easy to do with the provided makefile. You type

```
> make clean
```

to get rid of old copies of .o files and the old `kernel.bin` file. Then you type

```
> make
```

to start the compilation and linking steps necessary to generate a new `kernel.bin` file.

Now you are ready to test the kernel. For this, you will use QEMU, a very popular and powerful machine emulator. QEMU is very easy to install on both Linux and Mac OS (check out the Machine Problem Resources page on canvas for details). You will be able to boot your kernel by starting QEMU in *system emulation* mode, which emulates a simple x86 system that includes the CPU, memory, and simple devices:

```
> qemu-system-x86_64 -kernel kernel.bin
```

We tell QEMU that we want to boot a kernel called `kernel.bin`. If everything works fine, you should see a window with the output generated by your very own kernel! If you want, you can copy this binary on a USB and boot a PC with it. (Just to be safe, don't boot your kernel on your work machine. Things can go wrong, and you don't want to brick your machine.)

Typing the QEMU command can be tiresome, so we have included it in the makefile. The QEMU command is launched whenever you type

```
> make run
```

You will notice that the makefile also contains the definition of the command `make debug`. This starts QEMU in *debug* mode. In this mode, the emulated system appears to do nothing. Instead, it waits on port 1234 for a debugger to connect. If you use gdb for debugging, for example, you can connect to your kernel as follows:

```
> gdb
> (gdb) target remote localhost:1234
> <here you add whatever instrumentation commands you need>
> continue
```

You start gdb, then connect to port 1234, which is used by QEMU, and then tell QEMU to continue execution. Before typing `continue`, you may want to set breakpoints or do whatever you feel appropriate to instrument the kernel before starting the execution. More details on how to use gdb with QEMU can be found here: QEMU - GDB usage.

when things don't work, ask around, 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.

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. After testing your code using the QEMU emulator, you take a JPEG picture snapshot of the QEMU screen (it should show your name). Rename the picture file to `mp1.jpg`. You are to turn in the JPG file.

What to Hand In

- You are to hand in one file, with name `mp1.jpg`, which contains a picture of the QEMU screen showing your kernel in action, i.e. showing your name.
- Grading of these MPs is a very tedious chore. These handin instructions are meant to mitigate the difficulty of grading, and to ensure that the grader does not overlook any of your efforts.
- **Failure to follow the handing instructions will result in lost points.**

Project Grading Criteria for MP1

- Your kernel boots: 50%.
- Feature completeness and functional correctness: 50%. (This means your kernel prints the expected welcome message.)

Helpful Links

[1] TAMU Canvas: `https://canvas.tamu.edu`

[2] Windows Subsystem for Linux: `https://learn.microsoft.com/en-us/windows/wsl/`

[3] QEMU: `https://www.qemu.org/`

[4] QEMU and gdb: `https://qemu-project.gitlab.io/qemu/system/gdb.html`