CS 343 Operating Systems, Winter 2020
Getting Started Lab

1 Introduction

The purpose of this tiny lab is to make sure that you have everything set up so that you can do class labs. You will get, build, and run an operating system kernel, plus attach a remote debugger to it. This lab must be done individually. Clarifications and revisions will be posted to the course discussion group.

2 Setup

You can work on this lab on any modern Linux system, but we strongly suggest you do this lab on our class server(s) first.[1] We will describe the details of how to access the lab repo via github classroom in lecture and on the discussion group. You will use this information to clone the assignment repo using a command like this:

```
server> git clone [url]
```

This will give you the entire codebase and history of the Nautilus kernel framework ("NK"). This is an actively developed research tool among several institutions, including Northwestern. You will find CS343-specific instructions in the file `README.cs343`.

3 Task 1: Remote display

We need you to have remote display capability from the server. There are several ways to do this, including FastX, VNC, or X11. We will post more details about this on the discussion group. When you have this set up, you'll be able to do something like this:

```
client> ssh -Y you@server
server> emacs &
```

with the result that a new text editor window pops up on your client.

We also assume you are using the `bash` shell on the server. If you aren't, you can start it by just running `bash`.

[1] For students who definitely want to work on their own machines, we will give guidance on the discussion group.

4 Task 2: Build it

To start, you will need to configure the kernel:

```
server> cd [assignment directory]
server> cp configs/cs343-base-config .config
```

To compile the kernel and produce a bootable disk, do the following in your assignment directory:

```
server> source ENV
server> make -j 8 isoimage
```

The end result of this should be a file `nautilus.bin`, which is the kernel, and `nautilus.iso`, which is the bootable disk. Run these commands and capture the results:

```
server> ls -ltr | tail -5
server> md5sum nautilus.bin
```

5 Task 3: Run it

To run the kernel in emulation, execute:

```
server> source ENV
server> ./run
```

You should see a new window pop up, with content that looks like a computer booting. You will also see a bunch of output (boot messages) show up in the terminal where you ran `./run`.

Eventually, the new window will go blue and present you with a prompt that says `root-shell>` This is the command prompt of an NK shell. Type the following commands:

```
root-shell> cpuid 0 3
root-shell> mem fd520 128
```

The first of these commands shows information about your processor. The second dumps out a part of the system firmware (the BIOS). Save the results (you will see everything duplicated in the terminal, where it is easy to copy and paste).

Feel free to play with it more. Type `help` to see commands. Note that this is not a Linux shell. This is a minimal interactive interface that is running *within* NK. You can do anything with full privilege and very easily nuke the kernel. Try `int 0 23` for an easy panic.

6 Task 4: Run it with a debugger

In this final task, you will attach `gdb` to do remote debugging of the kernel. Start NK using the `run` command. While in the blue window, type CTRL-ALT-2. This will switch to a black screen with a `(qemu)` prompt. This is a command interface for the emulator on which the kernel is running. Run

```
(qemu) gdbserver
```

Now type CTRL-ALT-1. This will switch you back to the blue window (NK). Now, in a separate window on the server, attach to it:

```
server> gdb nautilus.bin
(gdb) target remote localhost:1234
```

Finally, run the following gdb commands and capture what they show:

```
(gdb) info threads
(gdb) bt
(gdb) x/s 0xfd7fd
```

The first command shows the hardware threads of the emulated environment on which the kernel is running. The second shows the the stack trace for the currently selected hardware thread. The last command dumps out memory at the given address as a string. This string is within the system firmware (BIOS).

7 Handin and grading

To hand in your work, create a file called `STATUS`. Place the outputs we told you to capture in Tasks 2, 3, and 4 into this file. Now add the file to your repo, commit it, and push:

```
server> git add STATUS
server> git commit -m "Done!"
server> git push
```

That's it! We will look at your `STATUS` file to confirm you've successfully completed these steps.

The purpose of this tiny lab is to make sure you have everything in place for future labs. We will help you do this if you have issues. The grade for this tiny lab is therefore all-or-nothing.