Week 1: The Science of Computer Science

Barry Rountree

Spring 2019

Live Ubuntu

I have created a custom Ubuntu image[1] that will be slightly easier to work with. There is no persistent storage, however, so unless you have off your work elsewhere, you'll lose it as soon as you log out. Saving your work off to a remote repository will probably be the easiest solution. You might also be able to use a second USB, scp, etc.

[1] `https://github.com/rountree/TSoCS-usb`

To boot off a USB in OSX, put in the USB, hold down on the alt/option key, and reboot. You should be given the choice of an "EFI Boot" device. Choose that.

I assume networking will not be a problem, but we will figure that out once we get into the lab. Choosing the network at the EFI screen is not necessary (at least not on my machine).

When the grub screen appears ("Try Ubuntu without installing"), you can either select that or, if you would like a bit more performance, hit 'e'. This allows you to edit the boot script. Adding "toram" (no quotes) before the "—" will load the entire USB image into RAM. Responsiveness should be much better, but you won't have as much space for other things. Boot time will also take a couple of minutes. If you go this route, hit F10 to continue the boot process.

byobu/tmux

Most of our time will be spent in the terminal, often editing multiple files with another window open for running programs. One way to handle this is multiple windows. Another is multiple tabs. A third is running a terminal multiplexor like tmux or byobu (byobu is what I use; the two are mostly indistiguishable). Briefly: `Ctrl-a c` creates a new window, `Ctrl-a n` goes to the next window, `Ctrl-a p` goes to the previous window, `Ctrl-a 3` goes to window 3, and most useful, `Ctrl-a A` allows you to label a window.

Github

Syllabus and weekly notes and assignments are at github.[2]

[2] `https://github.com/rountree/TSoCS`

We'll have a very brief introduction on how to set up your own repo, add files, save them back to Github, and pull them down again elsewhere.

LaTeX

A very brief introduction to LaTeXworkflows, including how to embed graphs into LaTeXdocuments.

The `msr` kernel module

We will add the `msr` kernel module into the running kernel and set the permissions of the device files that result using `chown`.

`rdmsr` and `wrmsr`

Download[3] and compile these utilities.

[3] `https://01.org/msr-tools`

Intel Documentation

We'll start from the main page for the Sofware Developers' Manual.[4] For these assignments, we will spend most of our time in Chapter 14 of volume 3B and chapter 2 of volume 4. Intel provides four different formats: all 4 volumes together, 4 volumes across 4 files, and 4 volumes across 10 files. Particularly when you're doing searches, the smaller documents are easier to work with.

[4] `https://software.intel.com/en-us/articles/intel-sdm`

Bash shell scripting

Do a quick sanity check to make sure we can read the time stamp counter using MSRs. Get a bit of data.

R

Pull that data into R and graph it. Save it to a `png` file. At what rate does the TSC increment?

Firestarter

Pull down and compile Firestarter.[5] Note that the build process is a little nonstandard.

[5] `https://github.com/tud-zih-energy/FIRESTARTER`

A few questions to consider

1. Section 14.2 of Volume 3B describes two hardware counters APERF and MPERF. At what rate do these counters increment during the sleep test?

2. Define *effective clock frequency* as $\frac{\delta \texttt{MPERF}}{\delta \texttt{APERF}}$. What is the effective clock frequency during sleep?

3. What is the effective clock frequency during a 10-second run of Firestarter on a single core? On all cores?

4. Use `setitimer`[6] to create a simple program that prints “Hello, world” every second.

[6] Run `man setitimer` for details

5. Tear down the `rdmsr` utility to see how to access MSRs via C. Add the code to the program above to let you read `APERF` and `MPERF` once per millisecond. Graph the results.

Assignment

Modify Firestarter to report back the effective clock frequency at multiple intervals give a fixed amount of work. Create two graphs: one showing wall-clock performance versus sampling rate (Does sampling more often slow down Firestarter? How bad is it?) and a graph that shows effective clock frequency over time. Repeat this for at least two configurations: running all cores and running a single core.