Project Zero (4%)

ENEE 447: Operating Systems — Spring 2012

Assigned: Monday, Jan 25; **Due:** Friday, Feb 5

Purpose

This project simply gets you up to speed. There are a number of fundamentals you need to have under your belt before you can even begin any of the projects, and this gets you to that point. Fundamentals include:

- Ownership of one (1) Raspberry Pi with multicore processor
- Ownership of a formatted micro-SD card, SD adapter, and SD card reader
- Ability to cross-compile from whatever operating system you run on your laptop to the ARM platform (there is an *extremely* small likelihood that your laptop runs ARM code natively)

Those are the fundamentals. In this project you will start with them and then put them together to build a kind of "hello, world!" example for the RPi. Once you have gotten that far, you will be ready to start building an operating system.

Get a Raspberry Pi

Go to https://www.raspberrypi.org for information on vendors. The board should cost $35. I have seen it for sale at some vendors for $39 and up. I don't understand how that can happen, but it does. Here is where I purchased mine:

http://www.alliedelec.com/raspberry-pi-raspberry-pi-2-model-b/70465426/

Very Important Note: get the version 2B, which has the quad-core Broadcom BCM2836 chip in it. Don't get a version 1 or a version 0. Those boards are for wimps and sissies, and you are definitely not wimps and sissies, are you? No, I thought not.

You might also want to purchase a version with a power supply, or purchase the correct type of micro-USB cable (note there are like 37,000 different versions of micro-USB out there, so make sure you get the right kind) if you want to power it from your laptop.

You will also want a NOOBS micro-SD card with adapter (comes pre-formatted, so it will save you time and hassle).

If your laptop doesn't have an SD-card slot, you will want a USB SD-card reader.

And, because you don't want to have to go out and buy a *second* Raspberry Pi board because you blew out one of the chips on your *first* Raspberry Pi board (the way I did), you should also purchase one of the many, attractive cases they have for them.

Install a Cross-Compilation Development Environment

The reason that the Raspberry Pi boards have captured the world's attention and imagination is because they started out at $35. That is a fully functional computer — minus only a screen and some form of data entry (keyboard, touchscreen, whatever). This means that you could build a computer in which the computer portion is actually the cheapest component (it's hard to buy a monitor or keyboard or mouse for less than $35). How could they possibly be that cheap? The answer: they are using cellphone chips as the main processor, and whenever a chip sells millions and millions of devices, the cost is low. Cellphones sell in the gazillions, so the components are cheap.

So the Raspberry Pi guys started out with a single-core Broadcom chip, the BCM2835. The first versions of the Raspberry Pi, based on that chip, took the world by storm. But, of course, that was several years ago, and Broadcom has not stood still … they now have a quad-core chip that they made especially for Raspberry Pi, the BCM2836 chip that has four (4) ARM cores in it, not just one (1).

What is the new price? The same as before: $35. So what, you might ask, astutely, are they doing with the old single-core BCM2835 chip, now that nobody wants to buy the old RPi board design based on it? Well, that chip is now cheap enough that they can build a board based on it, and sell it for $5. Yes, that is a single-digit dollar figure. You can buy a completely functional computer for less than the cost of the SD card that holds its operating system. It is called the Raspberry Pi 0.

But I digress. The RPi is cheap because it is based off cellphone chips, and cellphone chips have to be cheap enough to put into, you guessed it, cellphones. Therefore they cannot use Intel chips because Intel chips aren't cheap. So the cellphone manufacturers use ARM chips instead.

The problem is that your laptop uses an Intel chip. Intel chips and ARM chips are not compatible. "Not compatible" means that the code you write and compile on your laptop will run fine on your laptop but will be garbage on any other platform, including an ARM chip.

So what do you do? Answer: you use a cross-compiler. My suggestion: get Yagarto. Look it up, get it, download it, and install it. You need to configure it so that you have the

```
arm-none-eabi
```

tool set on your computer. You can test whether or not it is installed by typing "`arm-none-eabi-gcc`" and "`arm-none-eabi-as`" on the command line, if you are running Linux or MacOS. If you are running Windows, I feel for you.

Read the Baremetal Introduction

On the course website is an introduction to "bare metal" programming, written by David Welch, a professional software developer who happens to do a lot of enthusiast programming on the Raspberry Pi and then shares his code and his thoughts with the world. He has recently started to write code for the RPi2, so some of his examples are multicore-specific.

His introduction is eye-opening in that it shows you how different life is when you decide to write code that runs directly on the CPU, as opposed to running within the context of a particular operating system. For starters, none of your library routines work. That means *read*, *write*, *open*, *close* no longer work, but perhaps more importantly it means that you no longer have *printf* or any other I/O for that matter. You have no *main()* function, either, because *main()* expects a whole lot of stuff to have happened before it ever gets to square one. How do you think the various "hello, world!" examples can do so much with so little code? The answer: there is **tons** of work going on behind the scenes, and before the first line of your code ever comes to life.

Because Welch uses the RPi and ARM as his example, one of the things that his introduction will show you is how to understand what is inside of the binary you create. He will show you how to disassemble files to make sure that everything is where it should be, and he will make you realize how easy it is to screw little things up. He will show you how to get around the lack of *main()* and libraries, and he will show you how little assembly code you really need to get the job done. The good news: very, very little assembly code is needed. You should all be cheering at this point.

Write a Simple Timer Library

Once you have a cross-compiler development environment up and running, and once you have read about how to write code for the ARM architecture, you are ready to build something. On the course website are some documents on how the BCM2836 peripherals are set up, including timers. Timers are

extremely useful, and you cannot really do much in the way of system-level development without them. Your job is to create a function called *wait()* so that the following code example will work as expected:

```
while (1) {

        wait( ONE_SECOND );
        blink_led(GREEN, 10);

}
```

As well as *gettime()* and *timediff()* functions that will work as expected:

```
unsigned int now, then, delta;

then = gettime();
some_long_function_or_operation_that_we_need_to_time();
now = gettime();
delta = timediff(now, then);
```

We will provide you with a Makefile, skeleton code, and the code for the *blink_led()* function. Your job is to write the *wait() gettime()* and *timediff()* functions such that *wait()* exits approximately one second after entering (assuming the input is *ONE_SECOND*). By "approximately" I mean that any time duration in the immediate vicinity of one second (as measured by a stopwatch) is fine. A half a second, two seconds, one millionth of a second, an infinite stall, etc. — those examples count as "non-functional." The functions *gettime()* and *timediff()* can be at any time granularity you want, millisecond or finer.

You will find the ARM documentation on the course website especially useful here, because the "QA7: ARM Quad A7 Core" document describes, among many other things, the various clocks and timers at your disposal.

Build It, Load It, Run It

Once you have it working, show us.

Your Tasks

Your task in this project is to follow the steps detailed above:

- Get an RPi v2
- Get a power adapter or the right type of USB cable
- Get a NOOBS SD card
- Get an SD card reader (if your laptop doesn't have an SD card slot)
- Get a case for the RPi board
- Install Yagarto on your laptop
- Read the bare-metal programming introduction on the course website
- Write a small timer library.

Get it working and demonstrate your working code to the TA during your weekly lab/discussion section.