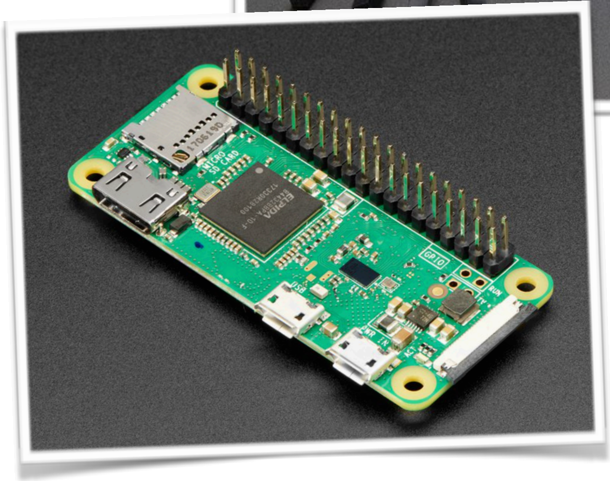
EECS 2031A 2018

Lab #4 Building an IoT device



EECS 2031 Lab #3

Building an IoT device

This lab involves building a sensor that monitors the world and provides this information to a connected node on the Internet. This is the first step in building a device for the Internet of Things (IoT) that uses a standard protocol (IFTTT). This lab thus involves a couple of different technical components

- Building a very simple circuit that connects an infrared sensor to a remote node on the Internet.
- Linking with an external library that connects to the Internet of Things.
- Building and using a more sophisticated Makefile that builds static libraries.
- Maintaining your software on a remote software repository (GitHub).

In order to complete this lab and obtain a grade for it you will have to demonstrate to your TA that you have completed the various steps in this lab. This lab, like all labs in this course, are intended to be experiential. You are encouraged to ask for help from both the TA's and your fellow classmates, and to help you fellow classmates in completing the lab. That being said, you need to master the material being presented here.

Connecting to the IoT

There are many ways of connecting to the IoT, there are many competing standards if you will. Some companies require a site (your home) to have a specific access point and then utilize a local network within the site to connect individual sensors. One advantage of this approach is that this can make the individual sensors or actuators less expensive. It also provides an additional level of security as the bridge between the Internet and your local sensor/actuator network can be used to filter out malicious users. Another approach – and the approach that we will follow in this course – is to put the individual sensors/actuators on the Internet directly. This is the type of approach used by devices such as the Nest, which must be connected to the Internet, typically through something like your home router, in order to work. From the point of view of the course, this is simpler as we already have a device, the Raspberry Pi, that has built in WiFi and we can leverage this. This also means that we do not need a bridge between the internal site network and the Internet, which makes this less expensive.

A very common way of communicating with a device over the Internet is through a URI. This is the common strategy used in modern browsers, where a resource on the internet is described using a string such as

https://foo.bar.com:8080/this/site/stuff/please

Monitoring the world

Updating your RaspberryPi

This lab requires that you install an optional library on your Raspberry Pi. Prior to doing the lab, execute the following commands to add the CURL development library to your Raspberry Pi

```
sudo apt-get update

sudo apt-get upgrade

sudo apt-get install libcurl4-openssl-dev
```

Note: that is libcurl (ending in an l) 4-openssl (ending in an l). Depending on how recently you updated your Raspberry Pi the upgrade may take some time to complete.

Create a GitHub repository and initialize it

Surf to github.com and create an account[1]. Using this account create a new project. Give your repository a name and create a README for it. Remember that this repository is (likely) viewable to the general public. Now there are a number of files on my GitHub repository that you should copy to yours repository. The following sequence of actions will do this (there are other options too).

On your Raspberry Pi check out my repository. (Use the same strategy that you used in the previous lab.) That is

```
git clone https://github.com/michaeljenkin/IoT-alarm.git
```

[1]Resist the urge to re-use passwords on multiple services.

Suppose that this is checked out into IoT-alarm. Now checkout your repository on your Raspberry Pi. Suppose this ends up in lab04. Copy all of the readable files in IoT-alarm to lab04. Then in lab04, execute

```
git add *
```

This will add all of the files to the repository. To push them to your GitHub site type

```
git commit -m “setup”

git push
```

You will be prompted for your name and password of your repository. At this point you should be able to refresh your GitHub site and see that you have copied the material provided to you for this lab.

Updating files on the GitHub repository from your Raspberry Pi

Motion detector

In your parts kit you will find a circuit board with a large white opaque dome, with three connectors on the other side. This a motion detector similar to the sort of detector that you will find in a home security system. This unit also has two tuneable potentiometers. Before using the device you need to determine the correct assignment of the pins. The opaque dome is loose fitted to the circuit board by four small pins. Pry the dome off of the board to expose the wiring. The pins should be labelled vcc (power) and gnd (ground). The third pin is a signal pin. You should connect the power pin to the 5V supply on the Raspberry Pi and the ground pin to ground on the Raspberry Pi. The signal pin should be connected to an input GPIO pin. For the example here we will use the WiringPi 0 pin.

The device also has two small potentiometers that can be adjusted with a small screwdriver. The Lab Monitor will have a few of these to loan. These two potentiometers adjust the sensitivity – how much motion is required to have the detector recognize motion, as well as the duration – how long the device continues to register motion once it does. For testing, it is probably best to set the sensitivity low and the duration short. That is, to set the first potentiometer in the middle of its range or farther clockwise (this makes it less sensitive), and set the second potentiometer as far as you can counter-clockwise (without breaking it). This way you can keep the device from detecting things until you really want to have it detect things, and once you stop waving your hands in front of the device it will quickly return to the idle state. Electrically, this device will signal a 1 when it detects motion, and will continue to signal a 1 until the time set by the second potentiometer is reached. It will then signal a 0 until a new event is detected. There is a small piece of code, in the GitHub repository provided that will exercise your detector for you under the assumption that the device is properly connected to WiringPi pin number 0. This code is printed below for your reference.

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

The practice IFTT server

In the GitHub repository you will find a copy of a pseudo-IFTT server. Basically its a restricted HTTP server that accepts two basic requests:

- A request that pushes data from your application to a database stored within the server.
- A request that dumps out the contents of the database as a web page.

This is all given in the Java code. There is a version of this code running as http://red.eecs.yorku.ca on port 8080, but if it is not working feel free to run it on some other machine. Note: Because of security restrictions in the lab, you cannot run it on a PRISM workstation and access it from the RaspberryPi's on the AirYork network. To run this java code you would type

```
java IFTTTServer 8080
```

Where 8080 is the port you want to use. After successfully compiling it using

```
javac IFTTTServer.java
```

You could run it on some machine connected directly to the Internet.

To view what information is stored on the server, assuming it was running on foo.com on port 8080, you could use a standard browser to surf to http://foo.com:8080/anything

IFTTT Maker platform

The IFTTT Maker platform (see here) is an IoT software system that allows any piece of code that can connect to a web page to signal information to a collection of IoT components. The basic concept in IFTTT is that IoT devices connect via an 'if this <event> then that <action>' There are many, many different events and actions. Here, your code generates an event – a web request – and you can choose an action – such as sending an email. We will use the Maker system in the next lab, but for this one we need to simulate how the IFTTT 'WebHooks' interface works. Basically, the Web Hooks does an HTTP (or HTTPS) post request with up to three parameters given as a JSON string. Within the GitHub repository for this lab you will find a library defining a function ifttt() that does all the work for you in this regard. But for this lab you need a simulator for the IFTTT world to play with, hence the Java code given above.

Static libraries

This lab builds a static library that contains the ifttt() code. Separating large software modules into separate libraries is a common tool in many languages, including C. You have already used libraries (the wiringPi library being one). Here you are going to build your own. The Makefile that builds and uses this static library is shown below

```
CC=gcc
CFLAGS=-lWarn -pedantic
```

```
tester: tester.o libmyifttt.a
     cc tester.o -L. -lmyifttt -lcurl -o tester

libmyifttt.a: ifttt.o
     ar -rcs libmyifttt.a ifttt.o

ifttt.o: ifttt.c ifttt.h
     $(CC) $(CFLAGS) -c -ansi $<

tester.o:tester.c ifttt.h
     $(CC) $(CFLAGS) -c -ansi $<

clean:
     rm tester *.o
```

The library built here is the myfittt library. Libraries end in .a and are included in a build using -lname or -lmyfitt here. The linker looks for libraries in a library path, here given as -L. Library file names start with lib and end in .a, and so -lmyifttt is looking for a file libmyifttt.a in the path specified by -L.

If you look at the Make rule for tester, you will see it depends on tester.o, the library libmyifttt.a and the build rule involves using the myifttt library and the curl library (which you downloaded and installed earlier).

Building a library involves compiling all of the object files that make up the library and then building the library archive using the 'ar' command. (Use man to read about it.)

Note that there is no rule for the sample code to exercise the motion detector. You might want to add one if you are going to use it for testing during development for the lab.

Hint: a 'make all' target is often added towards the end of a Makefile. With a line like

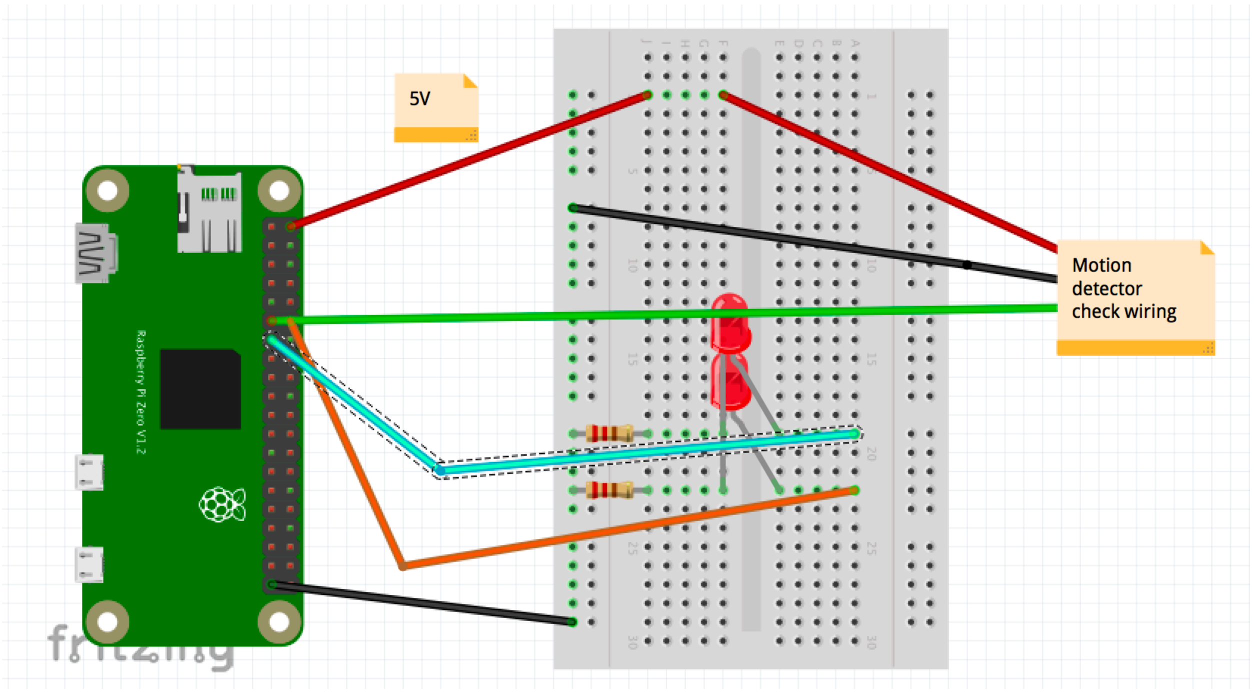
```
all:	tester myothercode stuff
```

Where tester, myothercode, and stuff are targets. This says that all depends tester, myothercode, and stuff.

A simple motion alarm

In this lab you will build a simple motion activated alarm. The alarm utilizes the motion sensor that is described above to monitor motion in the space. It uses two LED's to indicate that it is working - a green flashing LED - and that it is in an alarm state - a red flashing LED. Whenever it triggers an alarm it should connect to the simulated IFTTT server with information about the alarm that has been triggered.

You can do this project in many different ways. But here is a suggested strategy for solving this problem. First, build the circuit. It should look something like the figure shown below



]

Some observations here. The Motion Detector requires 5V and ground. Sending 5V into your Raspberry Pi is contraindicated[2]. The Motion Detector has three wires, vcc (5V), gnd (ground) and an output line. Be careful here. Wire the signal line from the Motion Detector to wiringPi pin 0. Attach an LED to each of wiringPi pins 1 and 2 making sure to put a resistor between the LED and ground. Use code that you have developed in earlier labs to check that your LEDs are properly wired. Use the example code provided on the GitHub to make sure that you are communicating with the Motion detector properly.

Update the Makefile so that it adds a new rule for a target ‘alarm’. ‘alarm’ should build an executable alarm that solves this week’s lab. Note: In a Makefile, the default target is the first one encountered in the Makefile, so placement is important. Also note that in a Makefile a TAB is the separation character between target and dependencies, and between the start of the line and the command required to solve the dependencies.

Start with the code provided to monitor the state of the motion sensor and add on the LED functionality. Print statements are your friend. Once you have that working, you add on the code to deal with talking to the pseudo-IFTTT server. Note that it can take some time to connect to the server (or it may even be down). You can use the web interface to check that the server is working Validate that your code is working. And you are done.

LINUX tools introduced in this lab

- git - further functionality of the git system and GitHub

[2] Doing so can easily make the magic smoke come out of your device requiring you to purchase another one. Once the magic smoke is gone, well its gone.

- make - further functionality of the Make system
- Building and using a static library.

Grading

To obtain a grade for this lab you must show your TA your Raspberry Pi running the security code that you developed here and that your code has been property added your own GitHub repository.

Extras

Next lab will use the IFTTT Maker service to connect a security circuit to the outside world. You may wish to get an IFTTT account now and read up on the Maker service that they offer.

Make GitHub repositories for the other labs. This will result in you having backed up your code base from the earlier labs to GitHub.