EECS 2031A 2018

Lab #10 An IoT Temperature Sensor in BASH

EECS 2031 Lab #10

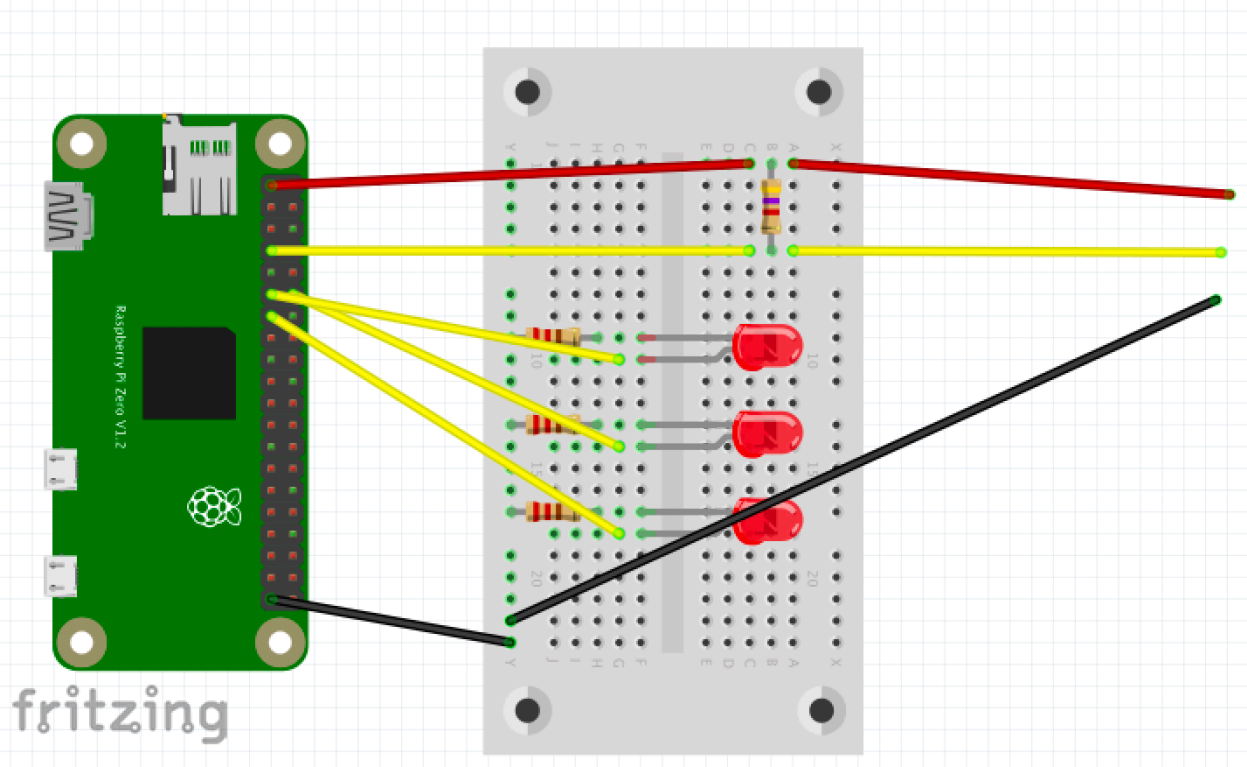
An IoT temperature sensor in BASH

This lab deals with measuring temperature with your Raspberry Pi and processing the data using BASH. This lab builds on a number of different labs, including the code you wrote in Lab 6 and the circuitry you built in the previous lab. You will want to refer to the earlier lab notes for parts of this lab.

Building the circuit

The instructions below assume that you have modified the temperature sensor that you built in lab6 and that in addition to the resistor that you will need for the temperature sensor (a 4.7K Ohm sensor) that you have three resistors and three LED's left in your kit. If you are running out of components you may have to borrow components from your fellow classmates.

The temperature sensor has three wires, a ground (black), a 3.3V (red) and a signal wire (yellow). A 4.7K Ohm resistor must connect the red and signal wires and 1 wire must be enabled in the OS. This will use Wiring Pi pin 4 as a 1wire signal. You will also wire three led's and resistors in series to ground to wiring Pi pins 0, 1 and 2. This complex circuit is essentially a mix of the circuits used in labs 6 and 9. So if you are unsure of what you are doing here you should review the circuit diagrams from those two labs. The red, yellow and black wires that extend off to the right of the image connect to your temperature sensor.



The first computational task

You are going to write a C application tempread that reads from the sensor and prints out the raw (*1000) temperature obtained from the sensor. Your code should take the following optional arguments

-v - which makes the output verbose. This will help in debugging. It will copy the temperature 1wire file to stdout as well as doing the primary task.

-d device - which lets the user specify the device on bus1 to read from. That is, if the hardcoded device in the source file does not match your device id you can input it here. So if your device was numbered 28-0118423e95ff, then you could type

tempread -d 28-0118423e95ff

to cause a specific device to be read from. Your code should assume that the device can be found on /sys/bus/w1/devices.

Your code should properly process the command line arguments and print out a usage message should the user not pass the correct command line arguments. The following code snippet may be helpful.

```
char *device = "28-0118423e95ff";
int verbose = 0;

/* process the arguments */
argc--; argv++;
while(argc > 0) {
  printf("%d %s\n", argc, argv[0]);
  if(!strcmp("-v", argv[0])) {
    verbose = 1;
  } else if(!strcmp("-d", argv[0])) {
    if(argc == 0)
      usage();
    argc--;  argv++;
    device = argv[0];
  } else
    usage();
  argc--; argv++;
}
tempread(device, verbose);
exit(0);
```

It is very important that your code just output the temperature (*1000) when it is run without the verbose flag set. That is, that when run it should output something like what is shown below

```
pi@mypi:~/Documents/lab06$ ./tempread
27937
```

Your code should return an exit code of 0 if it was able to read a temperature and 1 if it is not. You must write a Makefile to make your code and maintain all of your code on a GitHub repository.

Monitoring your temperature sensor from BASH

Now in BASH, write a script that runs forever. It should monitor the temperature of the thermometer relative to the temperature when the code was started. If the current temperature is 1 more but less than 2 degrees C above the starting temperature, 1 LED should be lit. If the current temperature is 2 or more but less than 3 degrees C above the starting temperature, 2 LEDs should be lit. And if the current temperature is 3 or more degrees above the initial temperature then 3 LEDs should be lit. You should use the C code that you have written to monitor the temperature. Note: Your code should do something appropriate if the temperature was not available. You can test this condition by unplugging the power wire from your temperature sensor from the circuit.

Building a manual page for your tempread application

Most LINUX applications have a manual page. Here you will make one for your tempread application. Create a directory (call it man) as a sub-directory of your lab10. In this directory create a file tempread.8 that initially has the contents shown below

```
.TH FOO 8
.SH NAME
foo \- a foo of an application
.SH SYNOPSIS
.B foo
[\fB\-n\fR]
[\fB\-\-bits\fR \fIBITS\fR]
.IR file ...
.SH DESCRIPTION
.B foo
does something
.SH OPTIONS
.TP
.BR \-n ", " \-\-bits =\fIBITS\fR
Some text should go here I expect
```

In this directory you can format your man page using man -l tempread.8

Populate this manual page so that it matches your application. Man pages are written in nroff. Nroff commands start with a period. There are many tutorials online on how to write manual pages (see here for an example). Normally, once complete you would install the manual page in a standard place in the OS. Calling man with the -l option causes the specified manual page to be printed. Add the tempread.8 file to your GitHub repository.

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.