EE 199: Linux Device Drivers Curriculum

Anthony C. Nguyen[1]
Department of Electrical Engineering, University of California, Los Angeles
[1] anthony.c.nguyen@ucla.edu

***Abstract*—The recent rise of MOOCs (massively online open courseware) allows for the possibilities for the first time to bring a class about Linux Device Drivers to a massive audience. I discuss briefly the current state of MOOCs, focusing in on an existing MOOC whose primary purpose is a lab component. Then I use the framework created by this MOOC and replace the curriculum with my own developed for two USB devices in Linux. As I learned the material to prepare these two devices myself, I picked up additional pre-requisite knowledge along the way. This knowledge I package into a "Background Knowledge" chapter so that the user might become familiar with it before working on the drivers directly.**

***Index Terms*— Curriculum, Device Drivers, Linux, MOOCs, Online Classes.**

I. INTRODUCTION

The recent rapid development of MOOCs, or Massively Open Online Courseware, has the potential to lead to a fundamental change in education. On average, 5% people complete a MOOC on average, but this measly 5% is of an astonishing 100's of thousands of people CITATION NEEDED. At the end of 2013, a study on one of the most famous of all MOOC classes, Harvard's Introduction to Computer Science: CS50x showed that 0.7% of enrolments completed the course for a certificate [1]. Also as part of their polling, Harvard found that the reasons why people signed up for the MOOCs, many of which signed up with no intention of finishing the entire course, but only to learn certain material or to find themselves disinterested in the topic at hand (since many MOOCs are free, it takes no cost in order to view the real course content of many classes so many people just register. More information about the major MOOCs available now is available via many different sources on the web [2] [3]. One other important point is where the money is coming from and going, and that the cost of setting up a new class is quite high, from tens of thousands to hundreds of thousands of dollars.

Given this new online environment, I wanted to take up this EE 199 project for twofold reasoning – to learn about the Linux Device drivers themselves, and to try to "flip the script" and do what teachers do instead of students – create lesson plans and materials with which to teach others. I could not think of the number of times I have been in a class only to think "I could have taught that better myself." This EE 199 project has thus given me the incentive to do exactly that [4].

II. UT AUSTIN 601X

On the edX platform, in January of 2014 there was a class offered on Embedded Systems of which the format seemed similar to what might work well for these Linux Device Drivers. I enrolled in this class in order to see how things went, and I was impressed by what I saw. The format of the class seemed like it would be very conducive to what we want to do here with the Linux Device Drivers, especially because it was also a lab based class. Figure 1 shows the main page of the course. This course was divided up into 3 units which were released at the beginning of the class, 5 weeks into the class, and 10 weeks into the class. The class was graded at your own pace, so as long as you received a grade of at least 70% among all of the assignments before the end of the class, you were awarded a certificate. Piazza was used as a discussion forum where students and professors would communicate openly and thus many questions could be answered by other students, and oftentimes answers to questions that I had would be answered visibly online. The format of the course guides the modules I created.

III. GOALS AND OBJECTIVES

The original project proposal that I submitted was the following:

"Develop modules that can be used in online educational settings. The topics of these modules will be Linux kernel module and device driver development for x86 and ARM architectures, multi-core systems. Also focus on embedded security, low energy computing, collaborative development. Work done under professor guidance together with other undergraduates and graduate students."

Professor Kaiser advised on this project, and I also received guidance from a Ph.D. student, Digvijay Singh. The specific modules I intended to develop in my personal EE 199 were both user-mode drivers for USB devices: the Dream Cheeky Big Red Button and the TEMPer USB Thermometer. The Big Red Button is a useful and simple example to be used for understanding a basic USB device, and then the TEMPer thermometer is a more complicated device which is several steps up in difficulty to understand. Other students worked on device drivers for other kinds of devices.

Of course, to develop a module that another student can learn from, I would need to understand both of these devices myself, so this was of course also one of my project goals. In fact, it should be said that it is **more important** that I understand these things myself because it would be impossible for me to teach something I do not myself understand.

My goal was to follow the following steps for both of the USB devices I worked on:

1. Under Windows, understand what the device was and how to use it.
2. Under Linux, using whatever resources I could find online, get the USB device functioning using any and all means available.

3. Strip those resources down to the core principles behind them, and then create material about them in the format of the UT Austin class.

IV. Big Red Button

A. *What is a Big Red Button?*

The Dream Cheeky Big Red Button is a button covered by a plastic enclosure that can actually sense whether or not it is open or closed. Therefore, it would be more accurate to describe it as a push button and a toggle switch. A picture of the button has been included as Figure 8 in the back. The button is sold as a toy, rather than a productivity aid, but the software that comes with the button does allow it to be programmed to open up a web page when pressed. However, when I first plugged it in, absolutely nothing happened! It did not come with any software in the box, and only by going back to the manufacturer's website did discovered that I had to contact them to obtain a link to the device software. After I heard back from them I ran the download and the installer, then the software was set up and my button ready to fire.

B. *What did I do to it in Linux?*

Misunderstandings and misassumptions caused me to have a lot of unnecessary difficulties with the device in Linux, such that an incredible amount of time was spent working on it.

At the advice of Digvijay Singh, I was told that I should investigate what USB packets the device was sending. Then by observing the appropriate packets, I would identify which packets corresponded with the lid being opened or closed, or the button being pressed or un-pressed. Knowledge about the packet behavior was the first step to writing a driver for the device. However, SnoopUSB tried and failed to read data from the Big Red Button. The USBlyzer tool also failed to get any data from the button; however, that tool was able to analyze a MicroUSB to USB converter just fine. This did not appear to be helpful for now.

Because I was getting nowhere with the USB packet sniffing tools, my next step was to searched the internet to find out if anybody had developed a driver of the device of their own. I was able to find two such drivers intended for this device.

The first driver result was created by Derrick Spell in Ruby [5]. However, there were no detailed instructions for installation, and I have never used Ruby before! Thus, I decided to first see if there were other drivers in C or C++ which I have used before. This brought me to the second driver, which was created in C by Malcolm Sparks [6]. The instructors provided here appeared to be much more simple, and did in fact turn out to be very instructive. However, they also did not actually work! Following the instructions word for word and obtaining the same results as described, I was extremely frustrated. I attempted to contact the author of the article but received no reply. After not getting that to work, I returned to the Ruby driver and tried working on that. I asked multiple computer science majors who have used Ruby before for assistance with the installation and configuration, but even with teamwork going for me I was still not able to get it working. And of course, no reply from this author either.

After being stuck for so long, I decided to go back to search Google in the hopes of finding a less closely related driver for another cheap device and trying to adapt that, or failing so, purchase a different device and call it a day. While searching for the related device, I stumbled upon another person who had the same issue that I did and was able to actually find out what was wrong [7].

C. *What online class modular content did I create?*

After running into all of the aforementioned problems, I wanted to make absolutely sure that the material I created for teaching was clearer than the information I had found online. Additionally, during the course of the learning process, I had felt many times that just getting a button to work was not very interesting. Therefore, I created the assignment regarding the button to have additionally goals beyond "getting the button to work."

Also in my frustration, no longer in the context of developing modules for instruction, I created a one-click installer for the button using the knowledge that I had learned previously while trying to get the button to work on my own. I also put this out online in the hopes that it would save someone else the difficulties I had during this experience.

V. TEMPer USB Thermometer

A. *What is a TEMPer USB Thermometer?*

The TEMPer USB Thermometer, here forth referred to as TEMPer, is actually one of many possible kinds of USB devices that are all sold under the same name, but have completely different internals. For that reason, one driver for one TEMPer device might not work at all for another TEMPer device, and so even in Windows, a customer needs to make sure to hang on to the installation disk for their particular device because they may not be able to find the same driver available online, and instead run into a different device.

The TEMPer device which I purchased appears in Windows Device Manager as a USB Composite Device made of a "HID Keyboard Device" and a "HID-compliant vendor-defined device." The thermometer itself and the program interface are presented in Figure 3A and 3B. A short list of program features: curve plotting, data logging in TXT or CSV, audible high and low temperature alarm, temperature calibration in C and F, Chinese or English language, automatic startup, and remote information by MSN, Email, or Skype.

B. *What did I do to it in Linux?*

This was a more complicated device than the Big Red Button. There was also a lot more buzz around this product, meaning there were a lot more variety of drivers for it that can be useful for different things, and each of the drivers I discovered online had different levels of complexity.

Due to the abundance of drivers online for this TEMPer, unlike the Big Red Button, I was able to get one of the drivers working as is, almost immediately [8]. Unfortunately, this program was written in Python rather than in C. I was able to have it up and running quickly because I had used Python before, knew how to run python code, and the instructions were better, but for the purposes of creating the class content the large inconsistency between Python and C would not be good. So I figured I would need to search out for a functional C driver for this device. It turns out that the same author also had a C version of the program, which was based off of another person's code. I tried to get it to working (and it worked/and it failed). This would then become the cornerstone of the next lab module.

C. *What online class modular content did I create?*

From a perspective of learning about device drivers, the only important function is how the to talk to the device to obtain the temperature; all the other features of the Windows software are very useful to have in practice, but are thus considered bonus goals for this material.

I do not have complete skeleton code for this device.

VI. Conclusion and Future Work

There are many more pathways to continue on in this work. A good next step would be to contact edX and Iversity to see if either one would be willing to host this class. Then the device drivers developed by the other students in the EE 199 would need to be put together so that all of them are in a consistent format between them. The class would need a full syllabus, and hopefully the different modules together would be able to work thematically, or build off of one another in a cohesive way. As a student in UT6.01x, I have had a lot of student perspective and experience with the MOOC class, but it might also be a good idea to contact Professors Jonathan Valvano and Ramesh Yerraballi to find out about how teaching the course went. At that point, you could have a small test run with a couple of students to see how they react to it, and then use their feedback as a means to gather information to help revise the curriculum. Then a first rollout of the class could be launched.

Regarding the two particular USB modules, I feel that the material for the Big Red Button is an excellent, simple introduction to USB devices that can be used to effectively teach other students. For the TEMPer Thermometer, I did not complete the skeleton code and therefore the material is not usable. I now see with the perspective of hindsight that I should have sought out more guidance earlier on in the project so that I did not get stuck for so long working on the button with no progress, then I would have had enough time to do good work on the TEMPer thermometer.

Appendix

My code for this project is available for public download at:
https://github.com/asian701129/EE199

Additionally, I have attached to the back of this document the modules meant for use in an online class. The format that they are based off of can be seen at the following website without registering for edX, so this link can be consulted immediately.
http://users.ece.utexas.edu/~valvano/Volume1/E-Book/

Acknowledgments

I want to give special thanks to Digvijay Singh for all of his direct help throughout the project and to Dr. William Kaiser for providing these ideas and opportunities. Furthermore, without the amazing class created by Dr. Jonathan Valvano and Dr. Ramesh Yerraballi I would not have had such a wonderful format that I could use to adapt my modules to match. Finally, my greatest thanks goes to Derrick Durdle and Phillip Adelt, whom without their original functional drivers I would not have been able to learn about Linux Device Drivers myself and create these lesson materials to teach others.

VII. References

[1] D. J. Malan, "This was CS50," 01 May 2013. [Online]. Available: https://blog.cs50.net/2013/05/01/0/. [Accessed 13 June 2014].

[2] "The Best MOOC Provider: A Review of Coursera, Udacity and Edx," 07 February 2013. [Online]. Available: http://www.skilledup.com/blog/the-best-mooc-provider-a-review-of-coursera-udacity-and-edx/. [Accessed 13 June 2014].

[3] X. HOLDAWAY, "Major Players in the MOOC Universe," 29 April 2013. [Online]. Available: https://chronicle.com/article/Major-Players-in-the-MOOC/138817/?cid=wc. [Accessed 13 June 2014].

[4] R. DeJong, "What Do MOOCs Cost?," 17 September 2013. [Online]. Available: http://www.mindingthecampus.com/originals/2013/09/what_do_moocs_cost.html. [Accessed 13 June 2014].

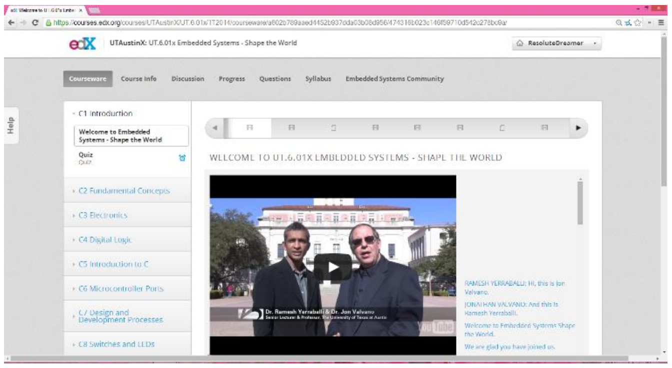
[5] D. Spell, G. Armagno and M. Sephton, "derrick/dream_cheeky," 12 May 2014. [Online]. Available: https://github.com/derrick/dream_cheeky. [Accessed 13 June 2014].

[6] M. Sparks, "The Big Red Button," 15 November 2013. [Online]. Available: blog.opensensors.io/blog/2013/11/25/the-big-red-button/. [Accessed 13 06 2014].

[7] D. Durdle, "Using the USB Big Red Button / Panic Button on the Raspberry Pi [day 10 of 20-days-of-posts series]," 21 December 2013. [Online]. Available: http://ddurdle.blogspot.com/2013/12/using-usb-big-red-button-panic-button.html. [Accessed 13 June 2014].

[8] P. Adelt, B. Cline, J. Monma, A. Schier and J. Stewart, "padelt/temper-python," 10 June 2014. [Online]. Available: https://github.com/padelt/temper-python. [Accessed 13 June 2014].

Figure 1: edX website demonstration

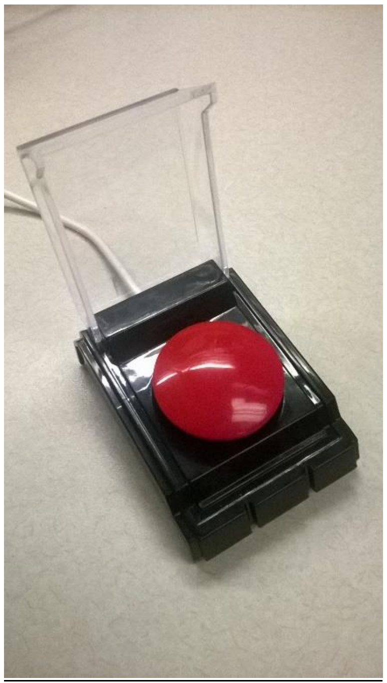

https://courses.edx.org/courses/UTAustinX/UT.6.01x/1T2014/courseware/a602b789aaed4452b937dda03b08d956/474316b023c146f59710d542c278bc9a/
edX
UTAustinX: UT.6.01x Embedded Systems - Shape the World
ResoluteDreamer
Courseware
Course Info
Discussion
Progress
Questions
Syllabus
Embedded Systems Community
Help
C1 Introduction
Welcome to Embedded Systems - Shape the World
Quiz
Quiz
C2 Fundamental Concepts
C3 Electronics
C4 Digital Logic
C5 Introduction to C
C6 Microcontroller Ports
C7 Design and Development Processes
C8 Switches and LEDs
Dr. Ramesh Yerraballi & Dr. Jon Valvano
Senior Lecturer & Professor, The University of Texas at Austin

WELCOME TO UT.6.01X EMBEDDED SYSTEMS - SHAPE THE WORLD

RAMESH YERRABALLI: Hi, this is Jon Valvano.
JONATHAN VALVANO: And this is Ramesh Yerraballi.
Welcome to Embedded Systems Shape the World.
We are glad you have joined us.

Figure 2: The Big Red Button and Software

A) The actual button hardware



B) The software provided with the button

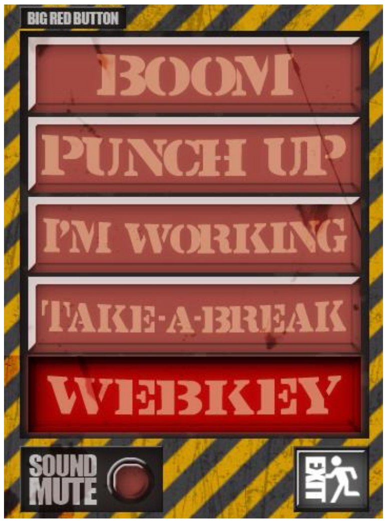
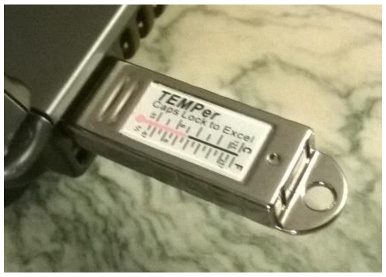
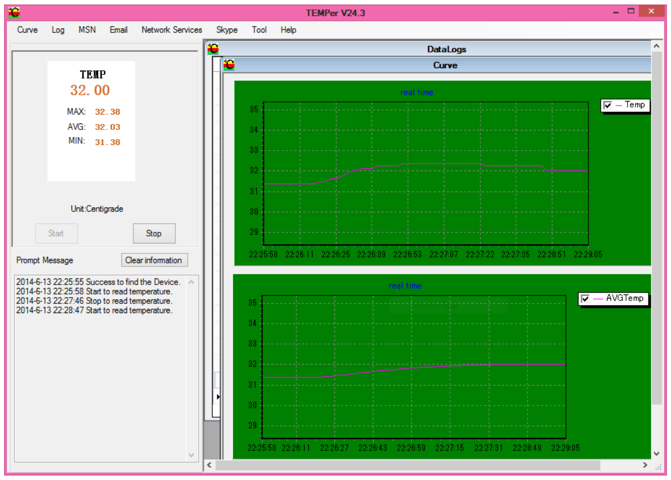


Figure 3: The TEMPer USB Thermometer

A) The actual thermometer hardware



B) The software provided with the thermometer



```
2014-6-13 22:25:55 Success to find the Device.
2014-6-13 22:25:58 Start to read temperature.
2014-6-13 22:27:46 Stop to read temperature.
2014-6-13 22:28:47 Start to read temperature.
```

Linux Device Drivers

[Comments are presented in brackets, like this. Material in bullet points is in outline form to be filled in later. All other material is typed in as it might be presented in actual course material.]

[Any links provided in this document are NOT intended to actually be given to a student to read. Instead, these links would go into the Additional References section at the end of each module as further reading, but the actual module would have text inline teaching the material of that objective. I only put the links here in this case as placeholder, and because I do not know well what constitutes plagiarism for academic teaching materials, thus writing the content in my own words is not a good use of my time right now.]

Module 1: Introduction

Welcome to Linux Device Drivers

Objective: Introduce teaching staff, go over course syllabus and format.

Quiz

This quiz will test you to make sure that you understand what the topics of this course are, and how the class will function procedurally. Unlike the other quizzes, this quiz can be re-taken indefinitely, and must be passed with 100% before further progress can be made. Any emails with questions about course content and structure which are already covered on the syllabus will be ignored. When you click on the next section, the quiz will begin.

Question 1:

What is the timeframe of this course?

A) 8 week course starting in April and ending in June
B) 10 week course starting in April and ending in June
C) 15 week course starting in April and ending in August
D) 5 week course starting in April and ending in May

[Correct Answer: Depends on the format of the course]

Question 2:

When are assignments due for this course?

A) Every Monday at 12:00 UTC
B) At the end of the course
C) Every Monday at 12:00 in your time zone
D) Deadline varies every week

[Correct Answer: Depends on the format of the course]

Question 3:

True or False? A letter grade for this class will be present on certificates for students who pass the class.

A) True
B) False

[Correct Answer: False, at least for the time being edX does not offer this feature.]

Question 4:

The course's *recommended* way to get help when you don't understand something is to:

A) Email the professors directly
B) Read and ask questions openly on the class Piazza discussion forum
C) Ask a friend
D) Ask a question on Stack Exchange

[Correct Answer: Whichever is actually recommended in the course syllabus. (B)]

Module 2: Fundamental Concepts

1. Module Source

2. Makefile

Objective: Learn about what are makefiles, how to write a simple makefile for a regular .c program, and then what needs to go into a makefile for a kernel module in modern Linux and why.

- http://mrbook.org/tutorials/make/
- Video with an example showing a makefile being used.
- http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html

3. Build script

Objective: Learn about script files for the bash terminal, how a script might help in the building of kernel modules and device drivers, provide a few examples.

- This

4. Kernel version

Objective: Learn about what the Linux Kernel is, a small bit about the history of previous versions of the kernel, and what are some major things that you might need to look out for between kernel versions.

5. Atom x86 vs ARM Platforms

Objective: Learn about the differences between these two processor architectures as relates to their origins, instruction sets, speed, power consumption, and security.

Quiz

[Include on the quiz several questions pertaining to each of the previous sections. I have included one or two example questions of each here:]

On this quiz, the number of tries for each question varies from 1 to 3 tries depending on the question. This quiz is not timed. When you click on the next section, the quiz will begin.

Question 3: Makefile pt. 1

Short Answer: What line would you add to the makefile in order to prepare an object file foo.o from the file foo.c using the g++ compiler.

____

[Correct Answer: foo.o: g++ -c foo.c]

Question 4: Makefile pt. 2

Short Answer: Edit the following lines of code so that foo.o is added into the main program when make is invoked:

```
main: main.o

        g++ main.o -o main
```

____

[Correct Answer:

```
main: main.o foo.o

        g++ main.o foo.o -o main]
```

Question 5: Makefile pt. 3

[This question should be about makefiles as pertaining to kernel modules]

Question 10: Atom x86 vs ARM Platforms:

Which of the following is NOT true about ARM processors?

A) Actual ARM processors are made by ARM Corporation
B) ARM processors are more power efficient and scale better than Atom processors
C) ARM processors are able to run Linux
D) ARM processors have all of the same security measures as Atom processors

Module 3: USB Driver

1. What is USB?

Objective: Discuss what USB is, the USB protocol, and how data is transferred using USB.

2. What is USB Sniffing?

Objective: Be able to use a program to inspect USB packets

3. What is the structure of a USB HID driver?

For this USB device, a command to get the current device status must be sent as an appropriately formatted packet. Then, the device will respond back with a data packet which is defined in a certain way.

- https://www.kernel.org/doc/Documentation/hid/hidraw.txt
- http://www.signal11.us/oss/udev/ - Introduction and Motivation Section only

4. udev rules and permissions

Objective: Learn about what are udev rules, when/why you would want to use udev rules, and how to write some common types of rule files.

- Immediately after plugging in a USB device, running the command *dmesg* will print the kernel log buffer, and near the bottom of the log will be an event corresponding to the insertion of the device. This information should be taken down as it is critical for writing the appropriate udev rule to give permissions for the device in user mode.
- reactivated.net/writing_udev_rules.html

Lab 1 – Big Red Button

Requirements

There are three major requirements for this lab:

1) Prepare the device for user-mode access, using an appropriate udev rule.
2) Starting from the skeleton code provided, write a device driver for this button that can request data from the button and receive data back from the button.
3) Interpret the data as it represents the four possible states of the button.
4) Then, using these states, program the button so that the button will perform an action if and only if the correct sequence of states has been accessed. This sequence will be:
    a. {Open Lid, Close Lid, Open Lid, Press Button, Press Button, Close Lid, Open Lid, Close Lid, Press Button, Press Button}

Getting Started

1) Everything needed for this udev rule has been previously covered in Module 2.4 udev rules and permissions. Consider the following:
    a. What is necessary for the device to work in user mode?
    b. How to make sure the rule works on the correct device?
2) Open the skeleton code for this lab. Consider the following:
    a. How would you keep track of the sequence of states?
    b. What would be a good way to debug the program as you go along.

Grading

Open the simulation grading program and select your .c file to be compiled. The simulator will open up and you will be able to toggle the states on a virtual button. When you are ready, press “Grade” and the simulator will go through the testing sequence as provided above. Based on the functionality of your program, the grader will produce a code which can then be sent back to edX.

Interesting Questions

Note that in normal operation, only three of the four button states are possible. What is the fourth unreachable state? What is an example of a function that can be assigned to this fourth state that would make sense if the operation ever reached the fourth state?

Lab 2 – TEMPer USB Thermometer

Requirements

There are three major requirements for this lab:

1) Prepare the device for access, using an appropriate udev rule.
2) Starting from the skeleton code provided, write a device driver for this button that can request data from the thermometer and receive data back from the thermometer.
3) Interpret the data as it represents the current temperature being detected by the thermometer.
4) Record the data over time in a CSV file which could then be used to produce a graph.
5) Actually plot the graph.

Getting Started

1) Everything needed for this udev rule has been previously covered in Module 2.4 udev rules and permissions. Consider the following:
    a. What is necessary for the device to work in user mode?
    b. How to make sure the rule works on the correct device?
2) Open the skeleton code for this lab. Compared to the Big Red Button, this will be more difficult. Consider the following:
    a. For partial credit, make sure that each of the sub functions previously defined work properly. How should you divide the program into functions to assist with debugging?

Grading

Open the simulation grading program and select your .c file to be compiled. The simulator will open up and provide the program with false temperature readings. When you are ready, press "Grade" and the simulator will go through a testing sequence. At the end, it will ask to open the CSV file generated by your code, which should be placed into the same folder as the program and named in the format "YYYY-MM-DD_HH:SS_Temperature". The grader will produce a code which can then be sent back to edX.

Interesting Questions

As discussed in the device description, the TEMPer is a very useful took to monitor the temperature of a server room where the temperature might fluctuate due to the weather on any given day. How would you extend the functionality already created so that an administrator could do more than see a graph of the data?

```c
//
// Code Skeleton created by Anthony Nguyen, based off of code
// from Malcolm Sparks; the original can be found online at
// http://blog.opensensors.io/blog/2013/11/25/the-big-red-button/
//

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//Fill in these defines appropriately
#define LID_CLOSED
#define LID_OPEN
#define CLOSED_BUTTON_PRESSED
#define OPEN_BUTTON_PRESSED

//Place your function declarations here


int main(int argc, char **argv)
{
  //Do not change these variable names
    int character_file, sequence_completed, res;
    int current_read_value;

    //Determine appropriate USB packet size
    char packet[8];
  //Add your own variables here

  //Initialize variables to appropriate values



  //Open the appropriate file and resolve failures


  //Operate the button and detect input into current_read_value,
  //and building up the sequence appropriately in the variable "sequence"

  //use the following command after every input
  usleep(20000); /* Sleep for 20ms*/




}
```

```c
//
// Code Skeleton created by Anthony Nguyen, based off of code
// from Malcolm Sparks; the original can be found online at
// http://blog.opensensors.io/blog/2013/11/25/the-big-red-button/
//
// This is a sample working solution to the Big Red Button Lab
//
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

#define LID_CLOSED 0x15
#define LID_OPEN 0x17
#define CLOSED_BUTTON_PRESSED 0x14
#define OPEN_BUTTON_PRESSED 0x16

//Place your function and variable declarations here

char expectedsequence[9] = {LID_OPEN, LID_CLOSED, LID_OPEN, OPEN_BUTTON_PRESSED,
OPEN_BUTTON_PRESSED, LID_CLOSED, LID_OPEN,OPEN_BUTTON_PRESSED,OPEN_BUTTON_PRESSED};

int VerifySequence(char* sequence, int current_sequence_length);

int main(int argc, char **argv)
{
    //Do not change these variable names
    int character_file, sequence_completed, res;
    int current_read_value;

    //Determine appropriate USB packet size
    char packet[8];

    //Add your own variables here
    int current_sequence_correct;
    int i = 0, j = 0, k = 0;
    char sequence[9];
    int previous_state = LID_CLOSED;

    // if we are currently in a valid sequence, append to sequence

    //Initialize variables to appropriate values
    character_file = 0;
    current_read_value = 0x00;

    //Open the appropriate file and resolve failures
    character_file = open("/dev/big-red-button", O_RDWR|O_NONBLOCK);
    // REPLACE big-red-button with the name of the device named by your udev rule

    if (character_file < 0)
    {
        perror("Unable to open device");
        return 1;
    }
    printf("The open() function assigned the number %d to refer to the big red button.\n",
    character_file);

    //Operate the button and detect input into current_read_value,
    //and building up the sequence appropriately in the variable "sequence"

    //use the following command after every input
    usleep(20000); /* Sleep for 20ms*/

    while (1)
    {
        memset(packet, 0x0, sizeof(packet));
        packet[0] = 0x08;
        packet[7] = 0x02;

        res = write(character_file, packet, 8);
        if (res < 0) {
        perror("write");
        exit(1);
        }

        memset(packet, 0x0, sizeof(packet));
        res = read(character_file, packet, 8);
        if (res >= 0)
        {
            if (j > 9)
            {
            j = 0;
            }
            if (current_read_value == LID_CLOSED && packet[0] == LID_OPEN) {
             printf("LID OPEN\n");

            sequence[j] == LID_OPEN;
            printf("Sequence value %d was added\n",LID_OPEN);
            j++;
            sequence_completed = VerifySequence(sequence);
            if (sequence_completed == 1)
            {
                printf("A correct sequence has been inputted!");
                sequence_completed = 0;
            }

             fflush(stdout);
            } else if (current_read_value != OPEN_BUTTON_PRESSED && packet[0] ==
            OPEN_BUTTON_PRESSED)
            {
             printf("BUTTON PRESSED\n");

            sequence[j] == OPEN_BUTTON_PRESSED;
            printf("Sequence value %d was added\n",OPEN_BUTTON_PRESSED);
            j++;
            sequence_completed = VerifySequence(sequence);
            if (sequence_completed == 1)
            {
                printf("A correct sequence has been inputted!");
            }

            fflush(stdout);
            } else if (current_read_value != LID_CLOSED && packet[0] == LID_CLOSED) {
             printf("Stand down!\n");

            sequence[j] == LID_CLOSED;
            printf("Sequence value %d was added\n",LID_CLOSED);
            j++;
            sequence_completed = VerifySequence(sequence);
            if (sequence_completed == 1)
            {
                printf("A correct sequence has been inputted!");
            }


             fflush(stdout);
            }
            current_read_value = packet[0];

        }
        usleep(20000); /* Sleep for 20ms*/


    }
}


int VerifySequence(char* sequence, int current_sequence_length)
{
    int z = 0;
    for (z = 0; z < current_sequence_length; z++)
    {
        // If we detect an incorrect action, start over from the beginning
        if (sequence[z] != expectedsequence[z])
        {
            for (z = 0; z < 9; z++)
            {
                sequence[z] = 0x00;
            }
            break;
        }
        else if (current_sequence_length == 9)
        {
            // If we have 9 items in the current sequence and no errors, we succeed
            return 1;
        }

    }
    // If we have arrived here, we do not have a valid sequence yet
    return 0;
}
```

```
#!/bin/bash
echo ""
echo "Big Red Button Installer!"
echo ""
#Install Device Rules
echo "Installing Device Rules!"
sudo cp -f ./dev/big-red-button.rules /etc/udev/rules.d/
#Verify
if [ -e /etc/udev/rules.d/big-red-button.rules ];
then
    echo "The udev rule was installed properly"
else
    echo "The udev rule was NOT properly installed"
fi
echo ""
#Check if button is plugged in
echo "Checking for the button"
sudo udevadm control --reload
if [ -e /dev/big-red-button ];
then
    echo "The button is currently being detected by the udev rule"
else
    echo "The button is NOT being detected by the udev rule"
fi
echo ""
#Select Button Functionality, Compile
echo "Would you like to use the button as a shutdown button or to run a custom script?"
select yn in "Yes" "No";
do
    case $yn in
        Yes )
    echo "Compiling"
    gcc ./dev/big-red-button-default.c -o ./dev/big-red-button
    break;;
        No )
    echo "Compiling"
    gcc ./dev/big-red-button-script.c -o ./dev/big-red-button
    sudo cp -f ./resources/big-red-button-command.sh /usr/local/bin/big-red-button-command.sh
    break;;
    esac
done
echo ""
#Verify
if [ -e ./dev/big-red-button ];
then
    echo "The button software has been complied successfully"
else
    echo "The button software has NOT been complied successfully"
fi
echo ""
#Add the program to the user's program directory
echo "Installing 'big-red-button' to /usr/local/bin/"
sudo cp -f ./dev/big-red-button /usr/local/bin/big-red-button
```

```
#Verify
if [ -e /usr/local/bin/big-red-button ];
then
    echo "The button software has been installed successfully"
else
    echo "The button software has NOT been installed successfully"
fi
echo ""
echo "Would you like to run the program now?"
select yn in "Yes" "No";
do
    case $yn in
        Yes )
    echo "Program Start!"
    /usr/local/bin/big-red-button
    break;;
        No )
    echo "Okay, we're done!"
    break;;
    esac
done
exit 0
```