CIT 595 Spring 2018
Arduino Lab

Introduction

In this lab, you will use the serial communication libraries to set up communication between an Arduino device and an application written in C.

Before you begin

Go to https://www.arduino.cc/en/Main/Software and follow the **"Download the Arduino IDE"** instructions for your platform.

The instructions below assume you are using a Mac or Linux machine. It is strongly recommended that you **not** use Windows; please notify the instructor if that is your only option.

Follow these steps:

1. Attach your Arduino board to your Mac/Linux machine via the USB cable.
2. Launch the Arduino development environment.
3. The first time you start Arduino, it may ask you to create a directory for your "sketches" (Arduino programs). Specify a directory (create a new one if need be) and then click "OK".
4. Once the IDE opens, go to "Tools" on the menu bar, then select "Board" then select the board you are using. It should be "Arduino/Genuino Uno" which should be the default.
5. Then from "Tools" on the menu bar, select "Port" then the device file for your computer. On Linux it should be something like "**/dev/ttyUSB10**" and on Mac it should be something like **"/dev/cu.usbmodem1411"** or similar.
6. Now you will open up one of the sample programs. On the menu bar, select "File", then "Examples", then "01.Basics", then "Blink". This is a simple Arduino program that just makes one of the lights blink (hence the name!).
7. In the new window that opens up with the "Blink" program, click the "Verify" button on the top left (the checkmark icon). This will compile the program for your specified platform. You should see the status message "Compiling sketch" and then "Done compiling" on the bottom of the IDE.
8. Last, click the "Upload" button (arrow pointing right). You should see "Uploading" and then after a few seconds, you should see the RX (receive) and TX (transmit) lights on the board flashing. Then the status bar should read "Done uploading" and then the tiny little orange "L" light on the board will blink on and off once per second.

Part 1

First, you will install an Arduino program that reads from the temperature sensor and writes the current temperature to the serial port.

Exit the Arduino IDE and disconnect the Arduino from your computer.

Restart the Arduino IDE and then download *temperature.ino* from Canvas. This is an Arduino program ("sketch") that reads from the temperature sensor, displays it on the 7-segment display, and then writes the value to the serial port.

Now attach the temperature shield to the Arduino, making sure that all pins are connected, and then re-connect the Arduino to your computer. Be sure the Arduino IDE is running on your computer **before** you attach the Arduino.

Using the Arduino development environment, open *temperature.ino* (it may ask you to create a new folder for it; that is okay), and then compile this program and load it onto the board as described above.

When you do so, you should see the lights on the 7-segment display blink, and then it should show the temperature in degrees Celsius.

Now choose "Tools" from the Arduino IDE's menu bar and then "Serial Monitor". After a few seconds you should see the temperature reading output coming from the board.

At this point, your Arduino board is ready to start communicating with the outside world!

Part 2

Now you will write a C program that gets the input coming from the Arduino board by reading from the serial port.

Download *read_usb.c* from Canvas and make sure you can compile it on your computer using a C compiler.

Then exit the Arduino IDE and disconnect and reconnect the Arduino. The "temperature" program should still be running on the Arduino; you don't need to reinstall it.

Now try to run the *read_usb* program and specify the name of the device file representing the serial port as a runtime argument. Assuming you are using Mac/Linux, you can find which device file it is using by looking in the /dev directory for a recently created file that will probably be named something like **ttyUSB10** on Linux and **tty.usbmodem1411** on Mac (note that this may be slightly different from what you used above when configuring the Arduino IDE).

The program should report that it successfully opened the file for reading/writing.

If it complains that the resource is busy, be sure that the Arduino IDE is not running, and try to disconnect and reconnect the Arduino.

Once your program says that it can open the file, implement the rest of the program so that it reads bytes using the *fd* file descriptor; these bytes are being sent by the Arduino as it reports the temperature.

You will need to use the **read** and **write** system calls to read bytes from and write bytes to the device file:

The **read** system call takes three arguments: the file descriptor; a char buffer/array into which to read the bytes; and the maximum number of bytes to read. This function returns the number of bytes read. It does not block, but will return 0 if there is nothing to read.

The **write** system call also takes three arguments: the file descriptor; a pointer to the char from which to start writing; and the number of bytes to write. This function returns the number of bytes written.

Your program should take the input coming from the board and write it to the screen using *printf*, so that you see the temperature appear once per second, just as in the Serial Monitor.

It is highly recommended that you attempt to store the <u>entire</u> incoming message in a global string in your C program! That is, do not just print one character at a time as it arrives. Although this is not strictly necessary just to mimic the Serial Monitor functionality, you will need to be able to do this for your group project. So you might as well do it now!

This seems really simple, but it can be very tricky!! A few things to keep in mind:

- Do not assume that the "read" command will read the entire message coming from the Arduino as one string; it may be read in small parts. Remember that the "read" system call returns the number of bytes read and does not null-terminate anything.
- Also, the "read" system call will return 0 if there is nothing to read, i.e. it will not (necessarily) block and wait for data.
- Because the Arduino program is using the "println" command to send the message, it will end with a newline character, i.e. '\n'. That is how you will know you've received the end of the message.
- Do not attempt to synchronize your C program and the Arduino by having the C program sleep for one second and then do the read. Just have it continuously looping and reading the bytes as they become available.
- Make sure you have closed the Serial Monitor in the Arduino IDE while attempting to run your program; you can't have two programs reading from the same input simultaneously.
- When you start your program and open the serial connection, this restarts the Arduino program, so you may see some "garbage" appear before the first temperature reading because of buffered data. It is okay if your program prints this out first.

Once you are done with these steps and can see the temperature readings in the terminal as printed out by your C program, please notify a TA and get ready to start your project!