Final Project: LEDPi

The University of Iowa
CS:3640 Spring 2016

Introduction

This goal of this project is to show that you understand the fundamentals of networking and can implement these concepts in an applied system. This double as an exercise in developing an Internet of Things (IoT) platform. You will be using a Raspberry Pi to control an array of LEDs by means of network requests. The hardware schematics will be provided, but the actual software implementation is up to you to design and implement. The programming can be done in either C/C++, Java, or Python and the required libraries for controlling the General-purpose Input/Output (**GPIO**) pins on the Raspberry Pi will be discussed later in this document.
Note that **bolded** words and phrases are probably good things to Google if you don't know what they mean.

Skills

This assignment will require you to either have, or develop, a few skills which are directly related to networking. These skills also translate into other areas of programming and software development, so you are expected to be able to troubleshoot most issues on your own and use freely available information on the Internet to augment your skill set. These skills include:

Linux Installation and Interaction You will be expected to install **Raspbian** (a derivative of **Debian** compiled for **ARM** and packaged for the Raspberry Pi) on your assigned Raspberry Pi and be able to interact with the **Bash** command line competently.

SSH (Secure Shell) Configuration and Managment While you will be allowed to use the Raspberry Pi with a keyboard and monitor, you are highly encouraged to configure your Raspberry Pi with SSH and do most of your development from another terminal. This may mean using an IDE of your choice on your own computer and transferring binaries to the Pi or streaming a text editor of your choice over SSH (**X11 Forwarding**). Also learn about setting up **SSH Keys** which use public-key encryption as an alternative to less secure password login. Try to refrain from using the supplied Raspbian desktop environment (called MATE, initialized by executing `startx`) since you are likely to use servers in the future which are headless (no GUI interface).

GPIO/Hardware Usage While you will not be expected to design the circuit layout yourself, the design we are providing is extremely simple and you should understand how it works. You are also allowed to add complexity to your final implementation as long as it follows the spirit of the assignment (which is primarily to control some trivial hardware over a network). If you have an idea, please present and get it approved during office hours.

Networking This is fundamentally a networking exercise. You will be expected to implement, at minimum, a ***REST*ful api** to have fine control over the LEDs. You can extend this by running a webserver with a simple page to allow you to make requests of the LED controller. You can also use the concept of **port knocking** to either protect your REST api, webpage, or SSH connection, or as another api to control the LEDs. If you do extend the project, you need to make sure to have the basic functionality outlined below well implemented before you start on other features. Any extensions will be considered for extra credit. Make sure to discuss it in your final report and README doc!

Materials

You be provided with the following components unless otherwise noted.

- Raspberry Pi (model B+ or 2)
- MicroSD card with $\geq$ 4GB of storage (Class 10 is preferred)
- 3 LEDs & Resistors ($\approx 220\Omega$)
- Breadboard and prototyping wires
- Ethernet Cable
- Monitor & Keyboard (optional, not provided)

Setup

1. Flash Raspbian onto your MicroSD card.
   - Get Raspbian image from `https://www.raspberrypi.org/downloads/raspbian/`. You can use NOOBS, but I suggest learning how to flash a generic image on your platform.
   - The Raspberry Pi Foundation provides tutorials for flashing images to your MicroSD (**installation guide** in the link above). You can also follow the guide provided in the **Intro2RPi** presentation posted in the project repository on GitHub.
2. Connect to the Raspberry Pi either through SSH or with a keyboard and monitor.

   Headless (SSH) The **Intro2RPi** presentation has a guide to how to get started with a headless RPi. You can Google for more tutorials, but there is also a fairly simple tutorial on the RPi forums: `https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=74176`.

   Headed (Keyboard and Monitor) No extra setup is required, but you will need to have a network connection. Ethernet is plug-and-play, but if you want to get a WIFI dongle working you will need to learn about **WPA-Supplicant** and specifically how to configure your network in the **wpa_supplicant.conf** file.
3. Expand the Rasbian image to the size of the SD card and set your environment variables.

   raspi-confi This tool should appear on first boot, but it can also be started by executing `sudo raspi-config`. Use it to expand the file system and change the root password on the device. Also, you can change processor over-clocking to take advantage of CPU capability on the Raspberry Pi 2.

4. Install and verify your libraries compile. Set up your Linux environment to your standards with your preferred language. You can use the `apt-get` package manager to get utilities and other package managers which Raspbian my not be distributed with. You can use various libraries for networking according to your preferred language. We are not going to require a specific build as long as it performs the assignment correctly and your source is documented and reproducible, so Google the networking libraries specific to your language and preference. To get an introduction to using the GPIO pins look at: `http://hertaville.com/introduction-to-accessing-the-raspberry-pis-gpio-in-c.html`. Find below the recommended libraries to use for controlling GPIO on the Raspberry Pi. Some libraries (specifically RPi.GPIO) may already be packaged in Raspbian.

   C/C++ (Wiring Pi) `http://wiringpi.com/`

   Java (Pi4J) `http://pi4j.com/`

   Python (RPi.GPIO) `https://sourceforge.net/projects/raspberry-gpio-python/`

Project Outline

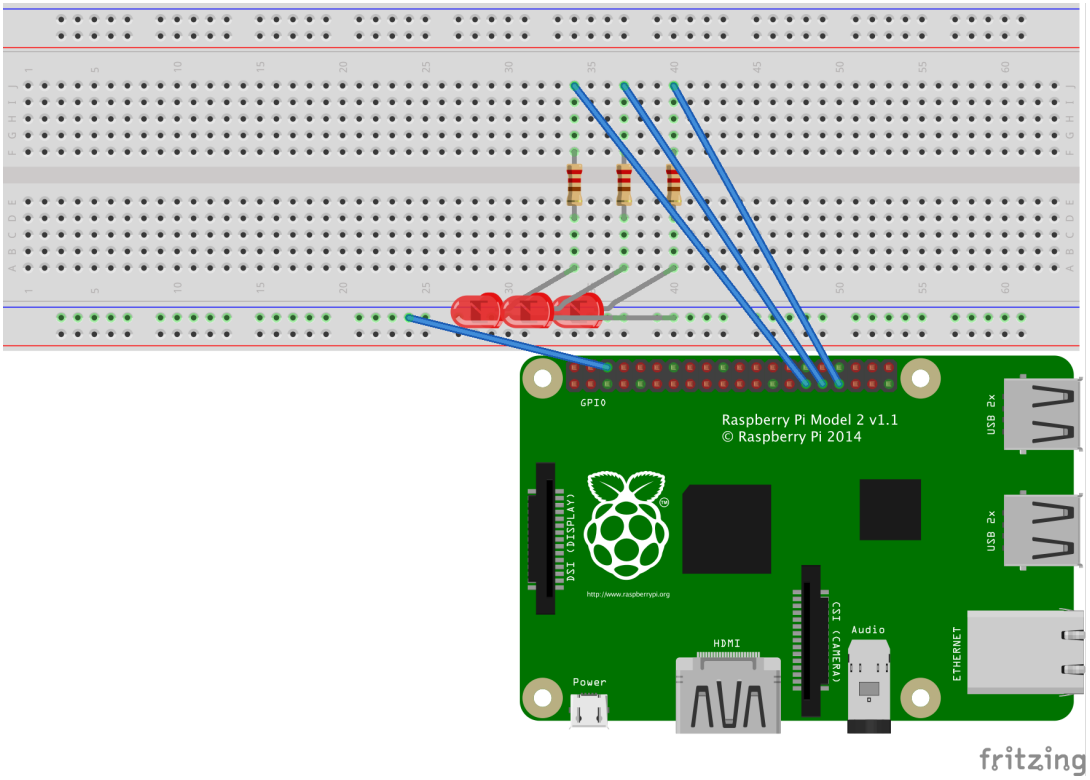
Having set up and configured your Raspberry Pi, hopefully by this point you are comfortable interacting with the Linux environment and have some idea of how to run programs and manage your device. The next step is to configure the hardware. The diagrams on the next page show the schematics of how to set up your circuitry.
You will be controlling the LEDs using what ever pins you prefer, but they will function similarly in all languages. One thing to note is that we can control the LEDs' brightness using **Pulse-Width Modulation** (**PWM**). The Raspberry Pi doesn't support hardware PWM, but all the listed libraries support some form of software PWM.

Program 1 Python, Controlling a Single LED and Cycling Its Brightness with PWM

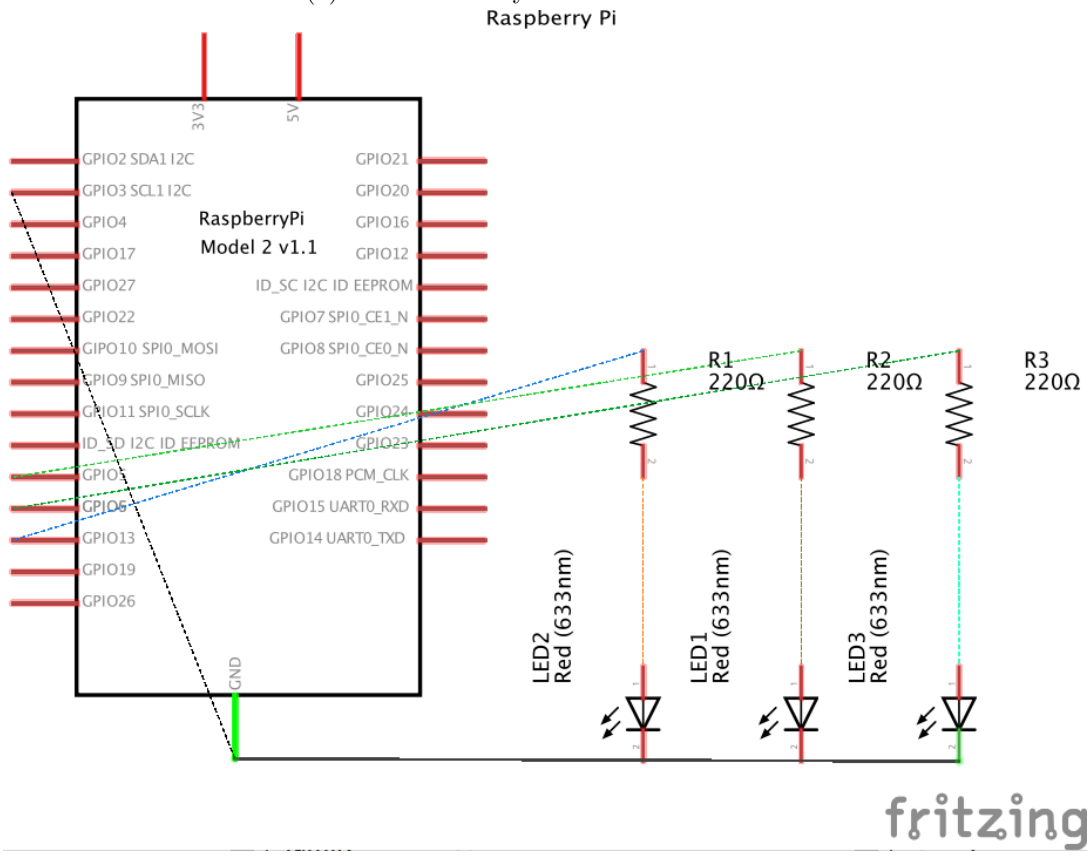
```
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)  # use pin 12 as output

p = GPIO.PWM(12, 50)  # channel=12 frequency=50Hz
p.start(0)
try:
    while 1:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```



(a) Breadboard layout for LEDPi Circuit



(b) Schematic layout for LEDPi Circuit

Project Outline Cont.

Once you have your RPi configured and controlling LEDs, you can start developing your REST api. Your Raspberry Pi should run a simple HTTP server which can receive `POST` requests encoded in **JSON**, **YAML**, or **XML**. The requests should somehow convey the index and brightness of the LED whose state is to be modified. To implement this you are allowed to use a **web framework** to handle your requests. The caveat is that the approved frameworks must be fairly basic and primarily support **routing**. The following frameworks (by language) are approved and have good documentation. Most of them are **micro frameworks** which provide only the basic functions of handling and serving HTTP. You will be expected to learn your chosen framework on your own; the TA and professor will not be able to answer too many questions. Of course, you are always welcome to use lower level utilities in your language of choice to handle HTTP requests.

- C/C++

  Crow `https://github.com/ipkn/crow`

  WT `http://www.webtoolkit.eu/wt`

- Java

  NanoHTTPd `http://nanohttpd.org/`

  Spark `http://sparkjava.com/`

  Jodd `http://jodd.org/`

- Python

  SimpleHTTPServer `https://docs.python.org/2/library/simplehttpserver.html`

  Flask `http://flask.pocoo.org/`

  Web.py `http://webpy.org/`

Example Client `POST` Request (Using Curl and JSON)

- Client sends:

```
clienthost:$ curl \
-H "Content-Type:  application/json" \
-X POST \
-d '"ledindx":0,"brightness":100' \
http://my.rpi.local/led
```

- Server doesn't respond (because REST) and turns the led at index 0 to full brightness.

Project Deliverable

You are expected to provide two primary components in your final project submission. The first component is a report which describes the process you went through to develop your application. This should be submitted in PDF format and have good quality in formatting and presentation. The second component is your source code, with enough documentation and comments to follow your design and be able to reproduce your environment from a clean installation of Raspbian. The source code should include a README (preferably in **Markdown** syntax) which explains

the organization of your projects, the required libraries and dependencies, and how to execute you program. This should be in a separate folder than your report, although the overarching project should be submitted in a single compressed file (**ZIP**, **TAR**, **GZ**). Look further down for an example project submission file structure.

Report Requirements

Your report should include the following sections:

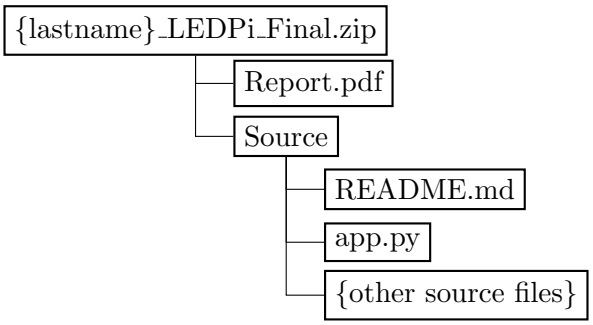
- Individual or team name(s) and title of project
- Introduction to your implementation (language, framework, etc.)
- Development process (including pictures of hardware, reference to source)
- Skills developed (What did you have to learn?)
- Issues encountered (What problems did you run into? How did you overcome, or fail to overcome, these issues?)
- Ideas for project extension (How could you make your application better?)

Source Requirements

Your source folder should include these features:

- README with the following sections:
  - Abstract of Application (a paragraph to explain what your program does)
  - Environment Reproduction (dependencies, compilers, and version information)
  - Compilation Directions (if applicable)
  - Execution Instructions (how to run your program)
- Source code with the following conventions:
  - Name your variables intelligently.
  - Comment your code well! You don't have to be overly verbose, but a reader should be able to identify what each function does and how your program flows.

Example Project Directory Organization



Further Recommendations

- It is highly suggested that you use some sort of **Version Control System** (**VCS**) like Git, Mercurial, or Subversion. The project report should include reference to the *publicly accessible* repository where your project is stored (if you use a VCS).
- If you are new to *nix and the command line environment, here are a few good resources to check out.
  - **Codecadamy** *Learn the Command Line.* `https://www.codecademy.com/learn/learn-the-command-line`
  - **Learn Code the Hard Way** *The Command Line Crash Course* `http://cli.learncodethehardway.org/book/`
- On the note of *nix interaction, it's important to know how to get help. Here are two useful commands:
  - `man` (Manual)
    - The built-in unix command for getting more information about a program.
    - Usage: `man {some command}`
  - `tldr` (To Long; Didn't Read)
    - *Simplified and community-driven man pages.* Not a complete man replacement, but has usage examples for many of the most common unix commands. Much less confusing and verbose than `man`.
    - `https://github.com/tldr-pages/tldr`
    - Usage: `tldr {some command}`
- Getting Help
  - Google is your friend. The Raspberry Pi project is very mature and has a ridiculous amount of documentation floating around online. If you hit a wall, someone probably has hit it too and found a solution.
  - Stack Overflow is a great place to post questions that you might have and have them answered by other programmers and developers. Out of courtesy to their community, please try to Google your issue first and look through the Stack Overflow tags related to your platform before you post a question.
  - Office hours also an option. Because of the flexibility of the project there is a good chance the professor and TA will not know details about the web framework you are using. Because of this fact, you are going to have to be fairly independent in learning your system. General project questions are good things to ask about.
  - There will also be a class discussion site (probably Piazza) where you can ask questions and your classmates and the instructors can reply. This will be especially helpful if several groups use similar **stacks** in their applications.