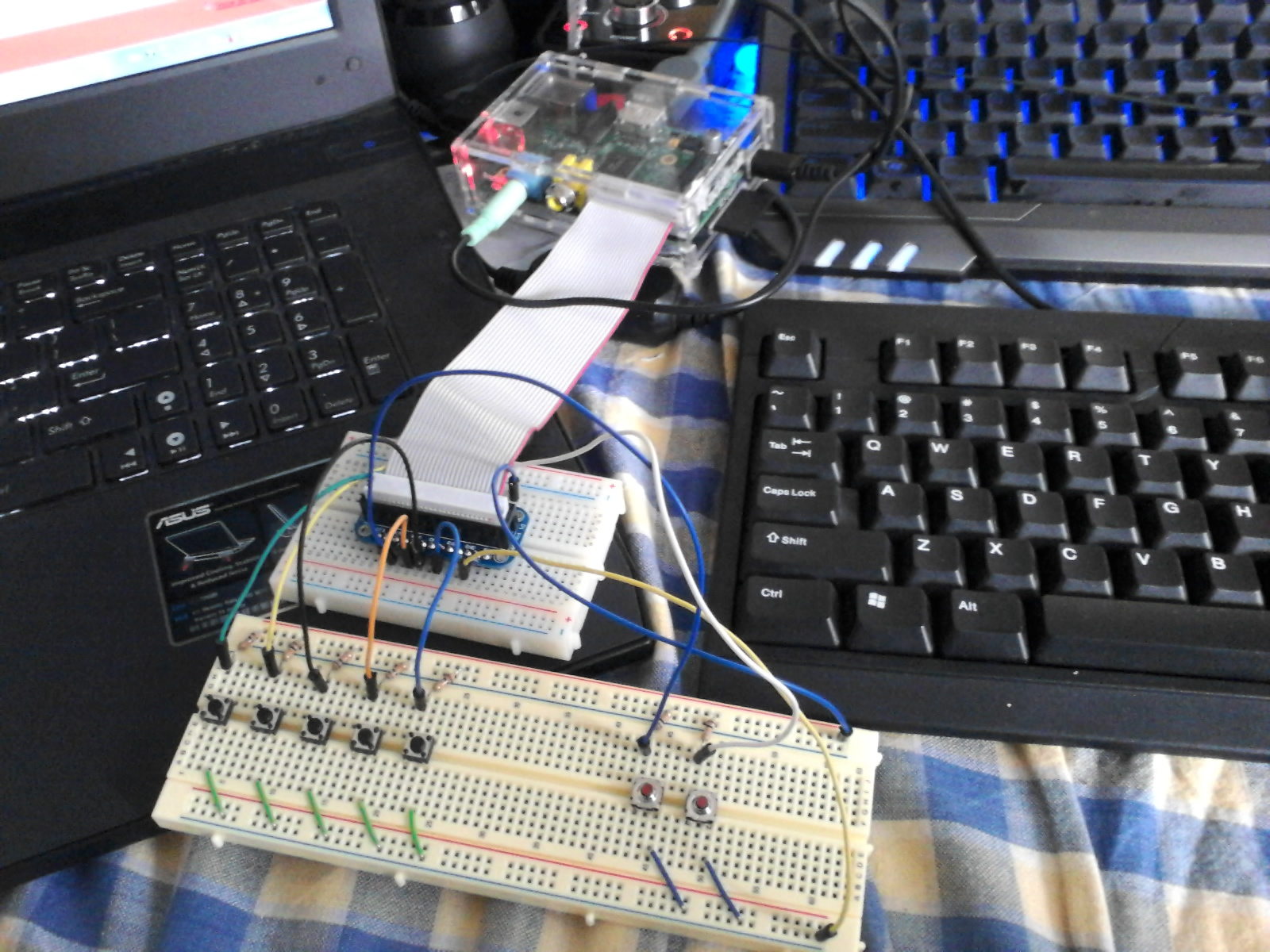
Music Player With External Push Buttons Using Raspberry Pi



By: Paul Baek

Objectives

To create a standalone music player that can be controlled with programmed push buttons. I used the Raspberry Pi, because it is a device I can program to be a music player without having to be plugged into a serial port. It is useful because of its portability and contains all required files and codes in its own “hard drive.”

I started out this project by thinking about what a music player normally has and what it needs. Based on that information I set it up accordingly. There were some limitations to the music decoder I used so I had to take some of that into account. After getting everything I needed delivered to me, I started working on my project and managed to finish it over a weekend. There were a few hiccups with the programming, but with a little research I was able to overcome those troubles and get the python code working the way I wanted it.

Parts list

2 breadboards: $15 each at RadioShack

7 push buttons: $2.50 for a pack of 10

7 10k ohm resistors: $1.19 for a pack of 5 at RadioShack

Raspberry Pi Budget pack from Adafruit.com: $49.95

Budget pack includes the following:

GPIO Kit, Wires, 4 GB SD card, Raspberry Pi enclosure, Raspberry Pi power supply, small breadboard.

Raspberry Pi: $35 at minimum

Generic USB Keyboard: $10

Generic Mouse: $10

Speakers: $15

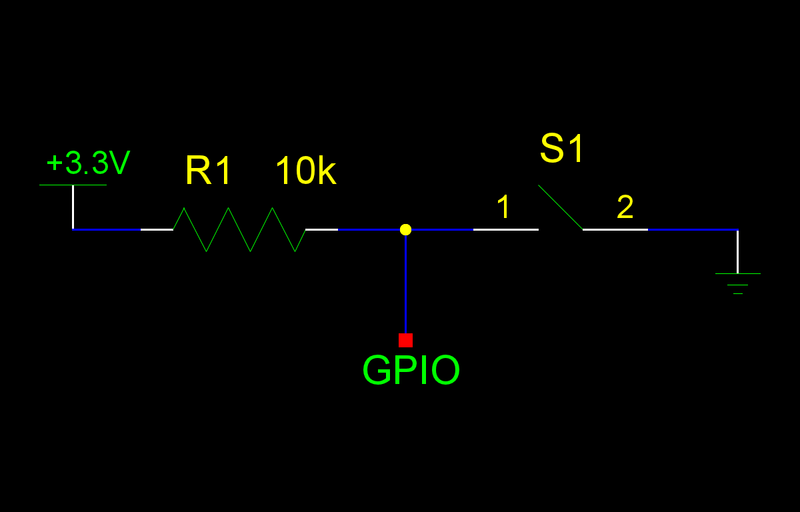
HDMI monitor: $100

The last 4 items are listed because they are necessary, but most people should own these things already. Generic USB keyboard and mice MUST be generic for the Raspberry Pi to function properly. I noticed strange problems with odd inputs and lag when using a keyboard and mouse with extra features.

Procedure

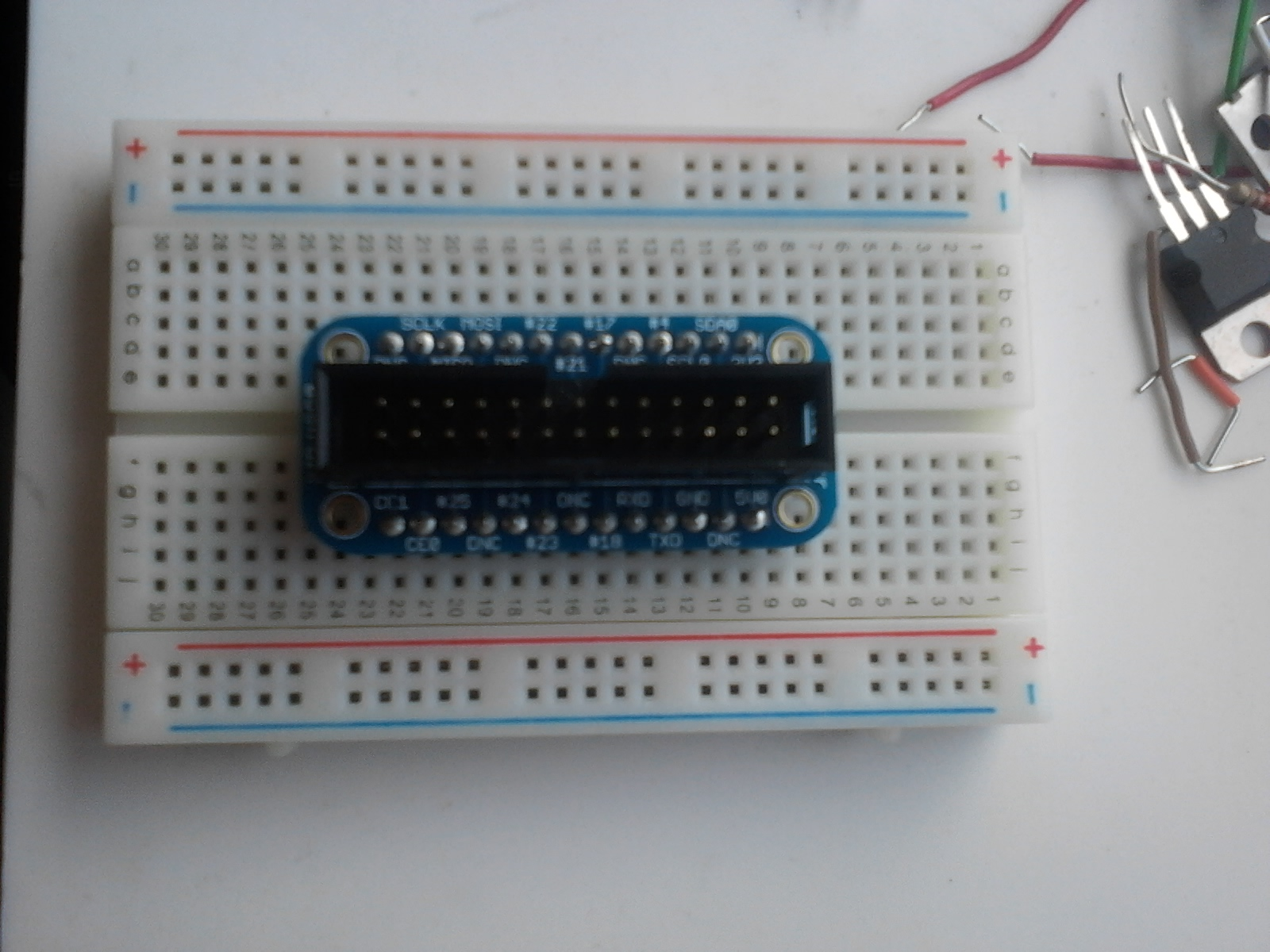
Wiring the Breadboard

The design of the bread board is very simple because all that’s required is that the push buttons are hooked up properly to the GPIO pins of the Raspberry Pi. This is the schematic used to wire the push buttons:



Where GPIO is the pin that will be read by the Raspberry Pi to determine the state of the button.

This is the piece used to connect to the Raspberry Pi’s GPIO pins.



The buttons are wired to the following pins:

Play: Pin 17

Pause: Pin 22

Forward: Pin 23

Random/Normal: Pin 24

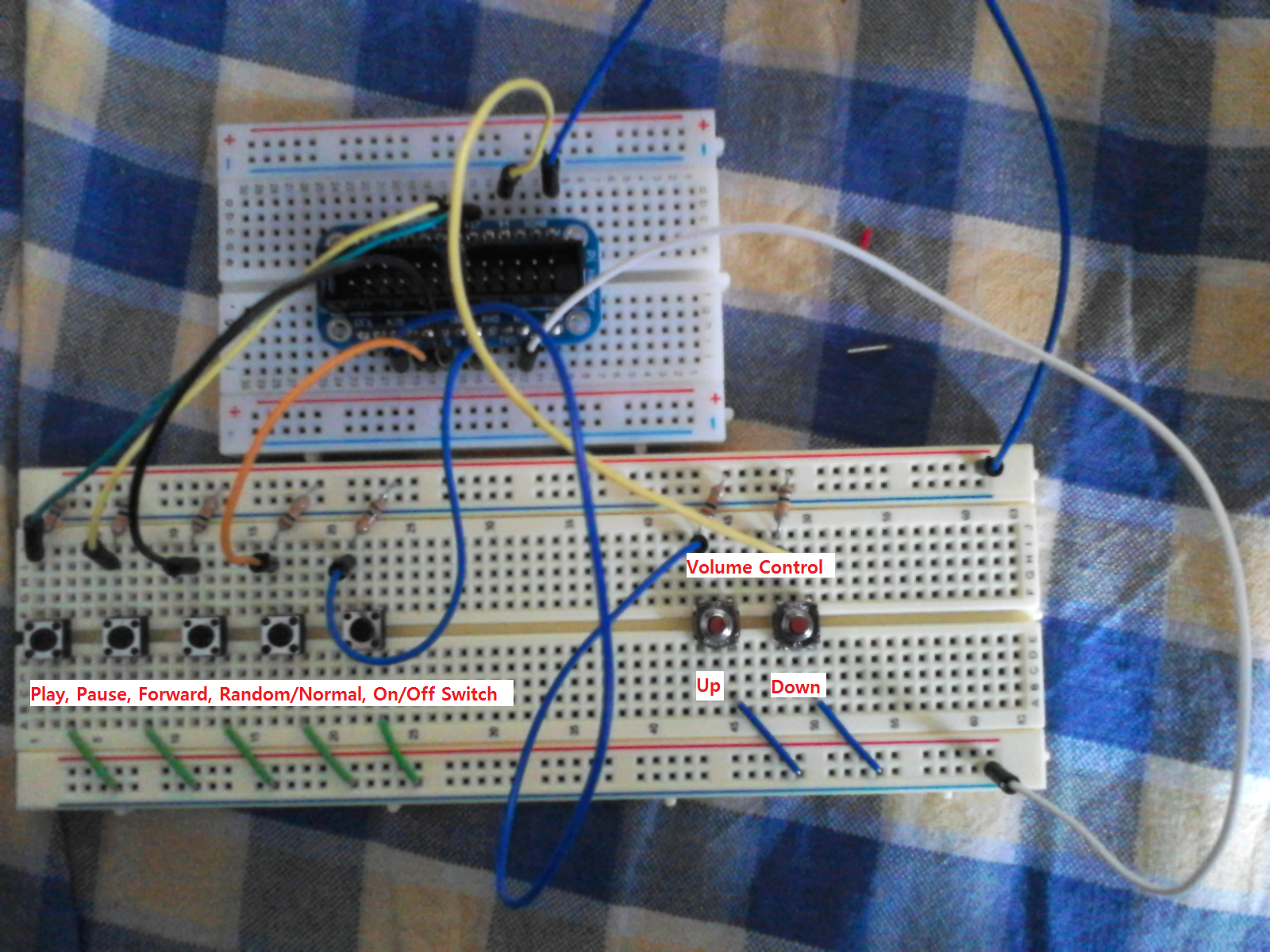
Volume Up: Pin 4

Volume Down: Pin 25

On/Off: Pin 18

3.3V power pin is the top right most pin (bad picture because it’s blocked) will go to the positive (red) strips on the breadboard. The pin labeled GND must be wired to the negative (blue) strip of the breadboard.

This is how the wiring looked after it was completely wired. The blue wire to the right is the 3.3V pin used to power the breadboard. The White wire is the GND pin used to ground the breadboard.



This is the way I set up the pins because I had a problem getting the proper input out of pin 21 rendering it useless.

Python code

The python code was written in Python version 2.7, on the Raspberry Pi itself. Alsa-utils, mpg321, and gpio must be installed onto the Raspberry Pi for the code to run properly. This was all done in the “Wheezy” image of Raspbian OS.

This is the code written in python using the pin set up that I used:

#!/usr/bin/env python

From time import sleep

Import os

Import sys

Import RPi.GPIO as gpio

Import signal

From subprocess import Popen, PIPE

gpio. Setmode.(gpio.BCM)

gpio. setup(17, gpio.IN)

gpio.setup(18, gpio.IN)

gpio.setup(22, gpio.IN)

gpio.setup(23, gpio.IN)

gpio.setup(24, gpio.IN)

gpio.setup(25, gpio.IN)

gpio.setup(4, gpio.IN)

gpio.setwarnings(False)

os.sytem(‘mpg321 -@/home/pi/audio/playlist.m3u &’);

control\_c\_sequence = ‘’’keydown Control\_L

key c

keyup Control\_L

‘’’

Def keypress(sequence):

P=Popen([‘xte’]), stdin=PIPE)

p.communicate(input=sequence)

return 0;

def sigint\_handler(signum, frame):

pass

vol=70

volstr=str(vol)+’%’

os.system(‘sudo amixer cset numid=1 - - ‘+volstr)

x=1

while True:

if (gpio.input(4)==False):

sleep(.5)

vol=vol+5

volstr=str(vol) +’%’

os.system(‘sudo amixer cset numid=1 - - ‘+volstr)

if (gpio.input(17)==False):

sleep(.5)

os.system( ‘ sudo pkill –SIGCONT mpg321’)

if (gpio.input(22)==False):

sleep(.5)

os.system(‘sudo pkill –SIGSTOP mpg321’)

if (gpio.input(23)==False):

sleep(.2)

keypress(control\_c\_sequence);

signal.signal(signal.SIGINT, sigint\_handler)

if (gpio.input(24)==False):

sleep(.5)

if(x==0’):

os.system(‘sudo killall mpg321’);

os.system(‘mpg321 -@/home/pi/audio/playlist.m3u &’);

x=x+1

else:

os.system(‘sudo killall mpg321’);

os.system(‘mpg321 –Z -@/home/pi/audio/playlist.m3u &’);

x=x-1

if (gpio.input(25)==False):

sleep(.5)

vol=vol-5

volstr=str(vol)+’%’

os.system(‘sudo amixer cset numid=1 - - ‘ +volstr)

if (gpio.input(18)==False):

sleep(.5)

os.system(‘sudo killall mpg321’)

sys.exit()

The code was saved as final2.py

This python code uses mpg321 as the music decoder and because mpg321 is a command line music player, this code must be run in command line.

Before this code can be run, a playlist must first be created. I created a folder and named it audio. I then put in the songs I want to listen to into this folder. After that is done, I went to the command line and changed the directory to the audio folder and input the following command: ls –w 1 > playlist.m3u. This will create a list of the songs in the audio folder for mpg321 to open. This command must be run in the audio directory every time new songs are inserted. When recreating the playlist the previous playlist must be deleted to prevent any errors or problems running the program. Once the playlist is created, I change directories to where the python code is saved. Then you need to run the command chmod +x final2.py. This will make the code executable. Then finally, to run the python code I enter the command sudo ./final2.py.

Result

The project worked almost like I had hoped it would. It pretty much functions like a music player, however, some features had to be left out due to my lack of knowledge on programming and the limitations of mpg321. Mpg321 has a skip forward function but not a back function, so I couldn’t replay a song I had just listened to. Another problem with mpg321 is that randomizing the playlist is an option that must be specified when the program is run and cannot be changed while it is running. This forces me to program the random button in a weird way in which the program must be shut off and must be run as random or as normal. Another problem I had is that when increasing the volume, it first goes down for some reason. That may be a problem with the alsa mixer and not the python code. In the end, I am just glad it works and I now have a source of music without having to run a music player on my PC. This helps a lot because it’s one less process that my PC needs to run.