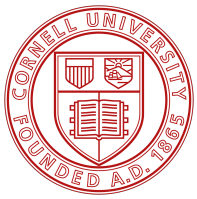
Cornell University



Electrical Computer Engineering

ECE 5725 - Embedded Operating System

LAB1 Report

Lab Section: Wednesday 402
Lab Date: 08/31/22 & 09/07/22

Guangxu Zhai (gz244) & Cynthia Li(xl827)

September 14, 2022

1. Introduction

Lab 1 familiarizes us with basic knowledge and functions of Raspberry Pi and Linux operating system. The main tasks of lab 1 are divided into two parts. First, we focused on setting up Raspberry Pi with a piTFT display and using `mplayer` to play videos on the display and connected monitor. Second, we focused on coding. Specifically, we created a `FIFO` to control the `mplayer` and wrote python scripts to allow users using physical button or keyboard to control the video playing. Though many problems occurred, all tasks were successfully completed and demonstrated to TAs.

2. Design And Testing

In this section, we will elaborate on lab procedures step by step and explain how we addressed issues encountered.

2.1 Physically assemble the Raspberry Pi

Attach the R-pi4 chip, case and also the piTFT screen together, and plug in required wires, including power, mouse, keyboard, HDMI, and SD card. Initially we forgot to plug in the sdcard, and nothing appeared on the screen. We asked TA for help and solved this issue.

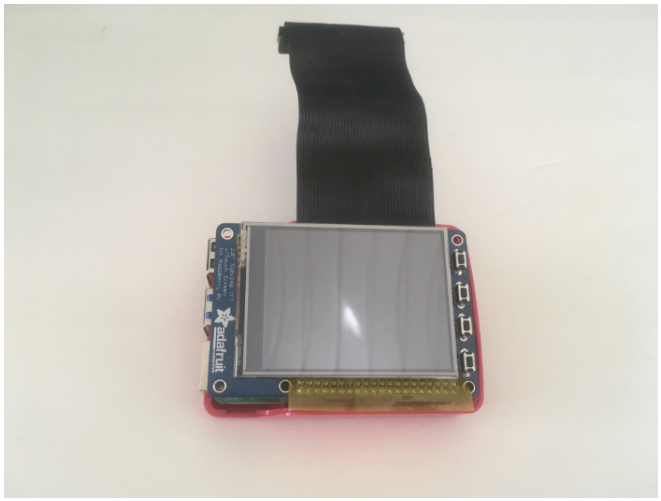


FIGURE 1: Completed Physical Appearance of Rpi4 and TFT Screen

2.2 Install and configure the Raspbian Linux kernel

We used the raspi-config tool to set up the Raspbian kernel by typeing `sudo raspi-config`. Some basic settings here includes time, loaction, language, username, and password. Then we checked and updated the local operating system to the latest version. The whole process of this step is smooth and fast, we just simply followed all the instructions provided in the lab manual and passed all the configuration set-ups.

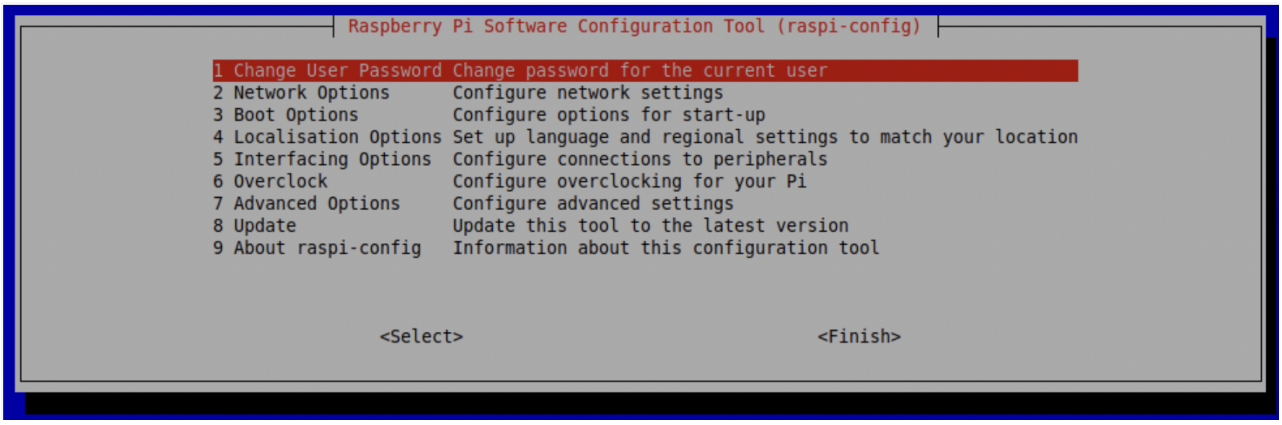


FIGURE 2: Raspberry Pi Configuration Tool Interface

2.3 Install and configure the TFT Screen

Firstly, we installed the required software that supports the piTFT by command `"sudo apt-get install -y bc fbi git python3-pip python3-smbus evtest libts-bin"`.

Secondly we used the administrator permission to edit the `"config.txt"` so as to change the boot sequence.

```
# Added for piTFT
[pi0]
device_tree=bcm2708-rpi-0-w.dtb
[pi1]
device_tree=bcm2708-rpi-b-plus.dtb
[pi2]
device_tree=bcm2709-rpi-2-b.dtb
[pi3]
device_tree=bcm2710-rpi-3-b.dtb
[all]
dtparam=spi=on
dtparam=i2c1=on
dtparam=i2c_arm=on
dtoverlay=pitft28-resistive,rotate=90,speed=64000000,fps=30
```

FIGURE 3: Add piTFT Info to config.txt

Thirdly, we run dmseg to check the installation of modules for piTFT and also added a udev rule so that we could run the command line on the touchscreen. We typed `sudo evtest /dev/input/touchscreen` to test the functionality of the touchscreen.

2.4 Install mplayer and play the video

We downloaded the bigbunny video by command `wget http://adafruit-download.s3.amazonaws.com/bigbuckbunny320p.mp4`. We tested the mplayer by `mplayer -h`, and then we boosted the audio volume for testing the sound.

2.5 Final tests of playing the video

We tested our functionality of playing the video through three ways: starting mplayer directly on piTFT screen; runing desktop and playing video on the PC screen; and remotely using laptop to type in commands and playing the video. The command code for these three are very close. The only difference is changing between `/dev/fb1` and `/dev/fb0` since fb1 is for piTFT and fb0 is for HDMI monitor. We found this description on the lab manual so we smoothly passed all the final tests here for week one.

2.6 Backup SDcard For Week 1

We used the `ApplePiBaker` on Mac to backup all the information in the SD card for our lab progress twice. One is at the mid of the lab, and another one is at the end of week 1.

2.7 Exploring mPlayer

This is a simple preparation step of week 2 procedure. As start of week 2, we restored the backup to the second SD card, and checked that the video can be played in the same way as in week 1, indicating that the back up from the first week was successful. Then, since week 2 mainly focuses on video control, we went to check the video control option list in the terminal by simply typing "mplayer" in the terminal.

2.8 Video Control Using FIFO

Starting from this step, we will need multiple terminal windows to play the video and input command to `FIFO` simultaneously.

First, we created a `FIFO` in the terminal named as "video_fifo" as manual instructed. To test whether video_fifo can work properly, we had two tests. First, on one terminal window, we used `echo` command to pop input text into `FIFO`, and then on the other terminal, we used `cat` command to see the content in the `FIFO`. Since same text was shown, we knew that video_fifo can operate correctly. The second test is to see whether the `FIFO` can give command to control the `mPlayer`. We first used the command `"mplayer -input file=/home/pi/video_fifo bigbuckbunny320p.mp4`, which allows the mplayer to receive input command from video_fifo. Then on one terminal window we play the video, and on the other terminal we can use `echo` commands to give commands from the option list explored in a. to the `mPlayer`. Some example commands we used were `pause`, `mute`, and `quit`. Since we observed that the video responded the same as our expectation of each command, we know that the video_fifo can control the `mPlayer` properly.

One issue occurred during the previous testing was that we knew that the `FIFO` was working correctly from the result of the first test, however, the `mPlayer` is not responding to the video_fifo. Originally we thought the problem was the naming between test_fifo and video_fifo (difference between the example in the reference and what lab manual required), but after trying with a new `FIFO` named by test_fifo again, we still have the same problem. It was then we found the problem was in fact the command line `"mplayer -input file=/home/pi/video_fifo bigbuckbunny320p.mp4`. We thought that the -input is the option argument as in bash commands, and deleted it during testing as we thought it was unnecessary. After adding the `-input` back in, the `mPlayer` can respond correctly. This issue took us some time to debug but it did provide us a better understanding of the command.

2.9 Video Control Using Python Script Controlled FIFO

Known that the `FIFO` works, we also wanted to use a python script to control the `mPlayer`. The difference of using python script and directly using `echo` command in the terminal is that the user won't need to use `echo` and use `> output destination`. The user may just enter "pause" "quit", etc. In other words, it would allow users without bash and linux knowledge to control the video playing as well.

Here, we created a python script named "fifo_test.py". It will works as a transfer station to pass on our command entered in the linux to the video_fifo, and the `FIFO` will control the mPlayer. Therefore, we designed our python script to give a prompt message saying "Enter your command here:" to let the user know that the program is ready. Then, as we want the system to constantly check whether there is input from the user, we want to use a `while` loop that always runs until a "stop" is given. Therefore, we also created a global boolean to control the `while` loop. Inside the `while` loop, it will use a series of `if/elif` statements to check if it is a valid command that we want to take, specifically four commands from the option list were kept: `pause`, `quit`, `run`, and `mute`. This will filter out invalid inputs that may contain typo or incorrect commands. Therefore, if

user enters any command listed above, the python script will `echo` the command to the system video_fifo. `quit` command is the only different one. It will also update the global boolean so the python can exit the `while` loop and end. Code for "fifo_test.py" is under the Code APPENDIX 4.1.

The testing in this part also has two steps. First, we tested the python without the video playing. On one terminal, we executed the python program by simply calling `./fifo_test.py`, which gave the prompt message. Then we typed in our command from the four specified above. Next, on the other terminal, we used `cat` to check whether the `FIFO` received the command. We repeated the type-in and `cat` process for three commands(except quit) and then typed in something that is not specified in the four commands. This time, when we did `cat`, we did not see what we typed in. Last, we repeat the step for quit command and it showed in video_fifo and quit the python. The results showed that the python can work appropriately (as shown in Figure 4). Therefore, we move to the next step testing, which is to test whether the python program can control the video playing. Again, on one terminal, we played the video, and on the other, we run python.

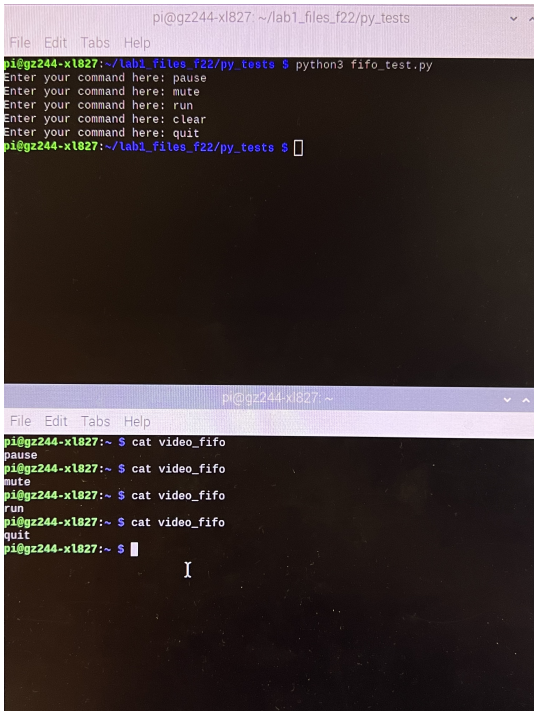


FIGURE 4: fifo_test.py Test Results

We entered the command from the four, and the video can change correspondingly. The result proved that the programs works properly.

The hard part in this step was that we did not know how to read in the input value from the keyboard or output the `echo` commands to the system. This part is solved after looking at Professor Skovira's sample codes "t3_v3.py".

2.10 Buttons Connection to GPIO

The tasks in this section is to configure the buttons on the piTFT to GPIO so the system can read in the on/off signal from the button.

One important thing to know before configuration is the GPIO number that each button should connect to. We found this from the Reference in the Modules named as "piTFT with R-Pi Schematics." In the right upper corner, we found the schematics for the four buttons(shown below). From the schematics, we saw that the correct GPIO pin for each button were 17, 22, 23, and 27.

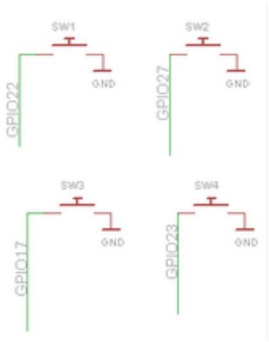


FIGURE 5: piTFT Four Buttons Schematics

Once we know the corresponding GPIO pin for the buttons, we can start to write the python routine. The one-button template code given by professor Skovira was almost complete. It already imported the GPIO and other neccessary modules, set the GPIO number to the Broadcom, initialized a button, set up a timer for the `while` loop to send message showing the program's still running, and used a `if` statement to send a message to the standard out so the user know that the button is pressed. The only adjustment we needed was to change to a correct GPIO pin (which we decided to use the button corresponding to GPIO 22, the second button from the right on the piTFT) and change the boolean variable used for the `while`. The code for this part is named as one_button.py under Code Appendix 4.2.

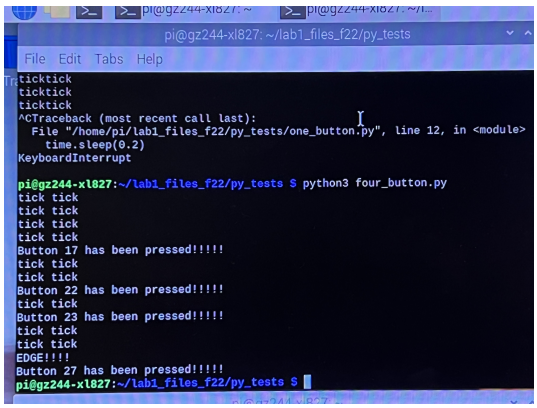


FIGURE 6: one_button.py Test Result

Testing this part was easy. We executed the python program in the terminal, the program will start to print "tick tick" in fast pace and once we pressed the button, correct message was printed to the terminal and we know the program was working properly. Results of this part is shown above in Figure 6.

There was one issue with the program though. As the lab manual did not make it a requirement, we did not write a `while` loop that will exit the program, so we have to manually exit the program.

In the second task, we need to configure four buttons. This is basically the same as one-button, except that we need to initialize four buttons and use four `if` statement to see whether the buttons are pressed. We did make two other changes though. First is that we adjust the sleep timer to 1.0 so the system prints "tick tick" slower (as we experienced a very fast clock in the previous testing). Secondly, we chose the button connected to GPIO pin 27 as the edge button, so it will print "edge" to the standard out and also change the boolean variable used for the `while` loop to end the program. The code for this task is named as four_button.py under the Code Appendix 4.3.

Testing the four-button program was basically the same as the one-button program. We executed the program in the terminal, which started to print "tick tick" but with a slower speed, and started to press the button from the side that we decided not be an edge. Once we pressed a button, the terminal would printed message saying the corresponding button is pressed, until we reached the edge button. When we pressed the edge button, the terminal showed message of "EDGE!!!!" as well as the button pressed message and exit the program. The result of the testing confirms that the four-button program could communicate with the system properly.

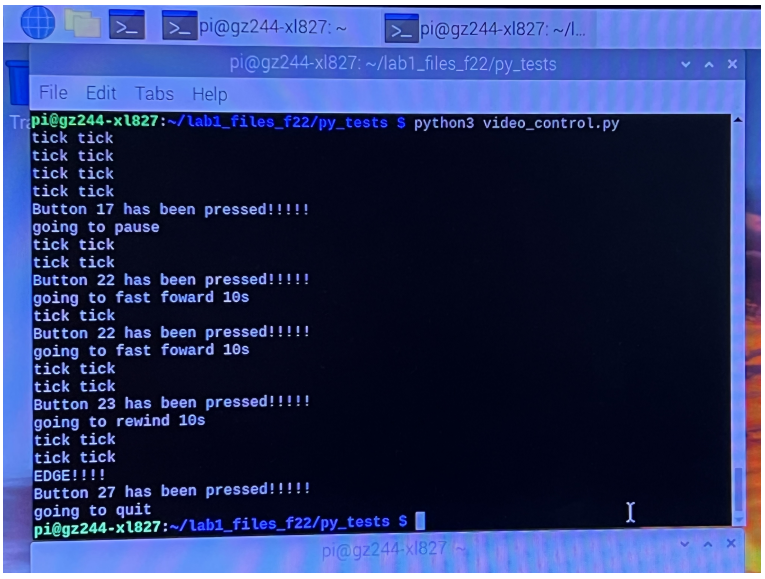


FIGURE 7: four_button.py Test Result

This part was easy and did not took a long time.

2.11 Video Control Using Buttons

The task was to use buttons as input commands to control the `mPlayer` through `FIFO`, which is basically a mix of the four_button.py and fifo_test.py. So we copied four_button.py to a new python script named as video_fifo.py, and added `echo` commands for each function specified by the lab manual(`pause`, `forward`, `rewind`, and `quit`) to the corresponding buttons(17, 22, 23, 27) as well as a `print` of the function message to the standard out. The code for this part is shown in Code Appendix 4.4.

In testing, we ran the video on one terminal and execute the video_fifo.py on the other terminal. On the terminal the ran python, it started to prompt "tick tick." When we pressed button 17, it showed message saying the button was pressed and "going to pause" and paused the video on the other screen. When we pressed button 22, it printed button pressed and forward video messages and continuously forwarded the video as long as we hold the button. Same thing with button 23 except it rewinded video. When we pressed button 27, it printed edge, button pressed, and quit messages, and quited the video as well as the python program. The results showed that the program works properly as expectation (shown below in Figure 7).

This part was smoothly and quickly completed as well.

2.12 Video Control Using Bash Script

This task is to write a bashscript and will start the `FIFO`-controlled video and the video_control.py program so that we can use buttons to control the `mPlayer` in one command. The code is fairly simple, it's just the python execution command video_fifo.py, and the `mPlayer` play video command. The code for this part is named as start_video.sh shown under Code Appendix 4.5.

There were two difficulties we encountered. First, for a long time, our program would only play the video and we have to manually quit the `mPlayer` and then it will run the python program. In the end, we found that this was because we did not add `-input` to the `mPlayer` command, which means that the `mPlayer` would not take any input. After the we added the `-input`, the `mPlayer` would successfully play the video and we can use the buttons to control the video. However, a different problem occurred. The `mPlayer` continues to play after we `quit` in python. After debugging, we realized that the sequence of the commands and what runs in the foreground what runs in the background matters. We ran python first in the foreground and then run `mPlayer` in the background. This means when we press button 27, the python will exit and send `quit` signal to `FIFO` which will quit as well, while the `mPlayer` is still running and waiting for `quit` command.

This part took us a long time because of the `-input` error, but it was a very good practice to learn about foreground and background.

2.13 Backup SD Card and Code

After all tasks we finished, we backed up the SD card and used `scp` to copy our code on the Raspberry Pi to the class server "ece5725-f22."

Backing up the SD card was easy and smooth. However, copying the code using `scp` had a little problem. Because of the internet problem, the kernel died doing `scp` and we could not copy the code to the class server from the R-pi terminal on the monitor. Instead, we had to log in remotely on the class server and transfer file from the class server. In this case, we need the IP address of the R-pi, which was found by typing `ifconfig -a` to R-pi. With all information for the `scp` known, we used `scp -p -r` because we wanted to copy the entire folder and files under it as well as preserving file properties, the source as `pi@IP-of-R-pi` , and the destination as `.` to copy the folder to the home directory of our account in class server (`/home/netID` /). Another note is that sometimes the kernel dies when trying to log in into the class server, the problem can be solved by turning on the school VPN.

3. Conclusions

In this lab, we learned how to set up the basic environment for using Rasberry Pi, how to use cmd to play videos, how to use scripts and codes to play videos. This lab is significant since it taught us the setup of R-pi4 and also how to implement python codes on system piratically.

Although we are almost the last group that finishes both the two weeks' labs, we still successfully passed all the check-ups and backed up our files. The reason why we spent so much time doing the lab is usually we would read the lab manual for long to understand all the concepts and also we often got stuck in one or two minor mistakes so that we have asked for TA's helps, they cannot solve our problem at first glance.

Some interesting facts that we noticed during the labs are the annoying text editor. When we wanted to copy and paste python code from one file to another file, it will automatically caused some indentation errors that prevent the code from normally running.

4. Code Appendix

4.1 fifo_test.py

```
 GNU nano 5.4                                                    fifo_test.py
#ece5725 lab1 week2
#creater: Guangxun Zhai, Cynthia Li
#netid: gz244, xl827
#this is a python file that to gets 4 different input keys from the user, and run different coresponding commands

import subprocess
import os

cmd_run = True


while cmd_run:
        #detect the input key from user
        input_key = input("Enter your command here: ")
        #if the input gets pause, system run command...
        if (input_key == "pause"):
                os.system('echo "pause" > /home/pi/video_fifo')
        #if the input gets quit, system run command..., and also jump out the loop to halt
        elif (input_key == "quit"):
                os.system('echo "quit" > /home/pi/video_fifo')
                cmd_run = False
        #if the input gets run, system run command...
        elif (input_key == "run"):
                os.system('echo "run" > /home/pi/video_fifo')
        #if the input gets run, system run command...
        elif (input_key == "mute"):
                os.system('echo "mute" > /home/pi/video_fifo')
```

4.2 one_button.py

```
 GNU nano 5.4                                                    one_button.py
#ece 5725 lab1 week 2
#name: Guangxun Zhai, Cynthia Li
#netid: gz244, xl827
#this is a python file that takes pressing button 22 as an input and printout


import subprocess
import os
import RPi.GPIO as GPIO
import time

my_name_is_true = True

#set to BCM mode
GPIO.setmode(GPIO.BCM)
#set up gpio
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while my_name_is_true:
        #give it a 0.2 time break
        time.sleep(0.2)
        print("ticktick")
        #if the button 22 is pressed
        if (not GPIO.input(22)):
                print("Button 22 has been pressed!!!!!")
```

4.3 four_button.py

```
 GNU nano 5.4                                          four_button.py *
#ece 5725 lab 1 week 2
#name: Guangxun Zhai, Cynthia Li
#netid: gz244, xl827
#this is a python file that has four pins as the input and print out different statements


import subprocess
import os
import RPi.GPIO as GPIO
import time

my_name_is_true = True

#set to BCM mode
GPIO.setmode(GPIO.BCM)
#set up the gpio
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while my_name_is_true:
        #time break 1.0
        time.sleep(1.0)
        print("tick tick")
        #if button 22 is pressed, print
        if (not GPIO.input(22)):
                print("Button 22 has been pressed!!!!!")
        #if button 17 is pressed, print
        if (not GPIO.input(17)):
                print("Button 17 has been pressed!!!!!")
        #if button 23 is pressed, print
        if (not GPIO.input(23)):
                print("Button 23 has been pressed!!!!!")
        #if button 27 is pressed, print, and jump out of the loop to halt
        if (not GPIO.input(27)):
                print("EDGE!!!!")
                print("Button 27 has been pressed!!!!!")
                my_name_is_true = False
```

4.4 video_control.py

```
 GNU nano 5.4                                          video_control.py *
#ece5725 lab1 week2
#name: Guangxun Zhai, Cynthia Li
#netid: gz244, l827
#this is a python file that takes four buttons as the input and do the different corresponding commands

import subprocess
import os
import RPi.GPIO as GPIO
import time

my_name_is_true = True
#set the mode to BCM and setup gpio
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#set a time break of 0.5; check the input; if find one input, the system runs the corresponding command; when reaches the final, halt
while my_name_is_true:
        time.sleep(0.5)
        print("tick tick")
        if (not GPIO.input(17)):
                print("Button 17 has been pressed!!!!!")
                print("going to pause")
                os.system('echo "pause" > /home/pi/video_fifo')
        if (not GPIO.input(22)):
                print("Button 22 has been pressed!!!!!")
                print("going to fast foward 10s")
                os.system('echo "seek 10" > /home/pi/video_fifo')
        if (not GPIO.input(23)):
                print("Button 23 has been pressed!!!!!")
                print("going to rewind 10s")
                os.system('echo "seek -10" > /home/pi/video_fifo')
        if (not GPIO.input(27)):
                print("EDGE!!!!")
                print("Button 27 has been pressed!!!!!")
                print("going to quit")
                os.system('echo "quit" > /home/pi/video_fifo')
                my_name_is_true = False
```

4.5 start_video.sh

```
  GNU nano 5.4                                                start_video.sh
#!/bin/bash
#ece5725 lab1 week2
#name: Guangxun Zhai, Cynthia Li
#netid: gz244, l827
#script to run bunny on piTFT
#run the videocontrol python code as a background and play the video

python3 /home/pi/lab1_files_f22/py_tests/video_control.py &
sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -input file=/home/pi/video_fifo -vo sdl -framedrop /home/pi/bigbuckbunny320p.mp4
```