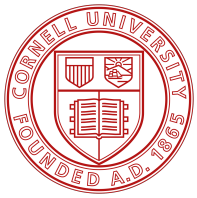
Cornell University



Electrical Computer Engineering

ECE 5725 - Embedded Operating System

LAB1 Report

Lab Section: **Thursday**
Lab Date: 09/09/21 & 09/16/21

Yu Zhang (yz2729) & Xinyu Wu (xw586)

September 24, 2021

1. Introduction

In the lab1, We obtained the initial knowledge of Raspberry Pi and the Linux operating system. The main topic is to let us learn how to setup the Raspberry Pi with a piTFT display, play videos on the piTFT display by applying the `myplayer` application and write Python scripts to control the video playing with the `FiFO` file and physical buttons on the piTFT display. We successfully completed all the parts and were always the first team to demonstrate the result to the TA in this lab section.

2. Design and Testing

In this section, we will introduce our experimental process step by step and describe in detail the phenomena and problems we encountered, the solutions and the final results.

2.1 Assemble the Raspberry Pi and Setup the Linux System

As mentioned in the first section, the first aim of this lab is to assemble the Raspberry Pi kit with a piTFT display and setup the Linux system. To be honest, this part was very convoluted and much harder than we expected.

First, we followed the steps in the manual and quickly assembled the Raspberry Pi with all required cable connected except plugging the power. But when we confidently showed the professor the finished work, the professor pointed out that we had assembled it wrong as the white stripe was installed in the opposite direction. According to the manual, the white strip on the cable should be positioned next to the small "1" on the pin side which is also the direction that the buttons are located. But even though we knew this mistake, it was still hard for us to pull it out by our hands. finally a TA used a screwdriver to help us slowly warp it out. The final assembled Raspberry Pi with the piTFT display is shown in the figure 1. (The photo was taken after the lab.)

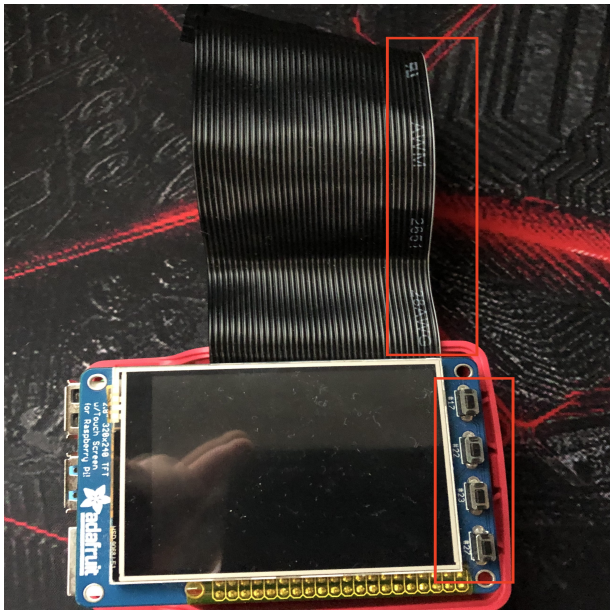


Figure 1: Properly assembled Raspberry Pi and piTFT display

We had got a less-than-stellar start, but the rest of the process was not smooth either.

After booted to the system, we firstly run some commands to get some system information for checking if the system is installed correctly. The commands included `whoami`, `hostname`, `uname -a` and `cat /etc/os-release`. The screen shot of result is shown in the figure 2.

```
pi@raspberrypi:~ $ whoami
pi
pi@raspberrypi:~ $ hostname
raspberrypi
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.10.17-v7l+ #1414 SMP Fri Apr 30 13:20:47 BST 2021 armv7l GNU/Linux
pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@raspberrypi:~ $
```

Figure 2: System Information Result

After ensured that the system information was correct, we started customizing the system. According to the manual, we should use the command `sudo raspi-config` to change system options, like **password**, **hostname** and **WI-FI**, and localization options, like **locale**, **timezone** and **keyboard**. We thought it would be easy because we had experience configuring Linux systems before, but we didn't expect that we would encounter many problems. Some of them were pretty funny.

The first problem was about the WI-FI setting. After we finished the customization of the system and tried to execute the command `sudo apt update` to update the Linux kernel, we got wrong message returned which noticed that remote packages resources could not be accessed by this device. This drove us to check our WIFI settings. And then, we found that the SSID we inputted was "**Redrover**" which should be "**RedRover**". After fixing the typo and rebooting the system, we were able to execute the command and update the linux kernel without any problems.

The next problem is the funny one. When we were setting the local option, we were supposed to unclick the **"en_GB.UTF-8 UTF-8"** option and select the **"en_us.UTF-8 UTF-8"** option, which represents English. But because of a misreading, we set it to the **"es_US.UTF-8 UTF-8"** option, which represents Spanish. However, we didn't know what happened at that time. We just thought the system text became a liitle strange after the reboot, but didn't take much attention on it.

Things got even stranger when we started setting up the keyboard. As shown in the manual, we should set the keyboard type as **"Generic 101 Key PC"**. Of course we just couldn't find that option as the language had turned to Spanish and none of us was able to understand this situation. So,

we asked the professor for help and found that the local setting was the root cause. After fixed this problem, we were able to finish the keyboard setting.

The last problem we encountered during configuring the system is the `ssh` setting. After confirmed that our Raspberry Pi had connected to the Internet (testing process has mentioned above), we executed the command `ifconfig wlan0` for inquiring the IP address. But we failed to connect it with `ssh` command. Our first thought is that may be the `ssh` command had not been installed yet so we used the command `sudo apt install openssh-server` to install the package. But the result noticed that this package had already been installed. Then, we used the command `sudo systemctl status ssh` to check the status of the `ssh` service. In this way we found the real reason that the `ssh` service had not been activated. Once again, we asked the professor for help and he solved this by using the command `sudo raspi-config` to enable the `ssh` service in the Interface Options. The screenshot of the setting page is shown in figure 3.

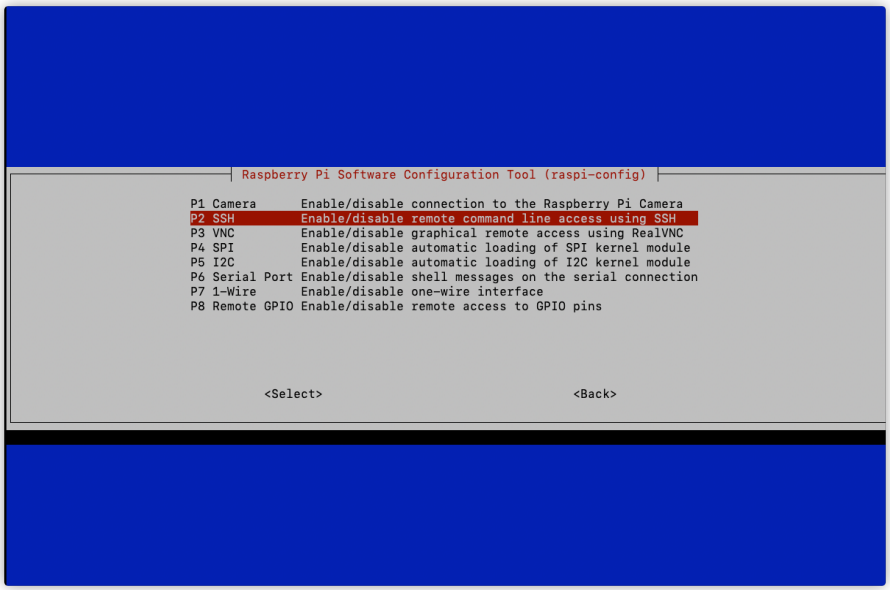


Figure 3: SSH Setting Page

Finally, we managed to complete the system setup.

2.2 Configure piTFT display and play the video

In this part, we aimed to setup the piTFT display and play video on it.

The configuration process of the piTFT display went very smoothly and we didn't encounter any problems. The first step was downloading essential packages by the command `sudo apt-get install -y bc fbi git python-pip python-smbus python-spidev evtest libts-bin`.

In order to change the boot sequence, we edited the file `/boot/config.txt` with `vim` and added lines shown in the manual into it. According to the Raspberry Pi documen, the `/boot/config.txt` stores configuration for booting and will be read it by the early-stage firmware. The `dtoverlay` option make the piTFT be integrated into the boot sequence as an overlay. After edited the

`/boot/config.txt`, we rebooted the system and observed phenomena that the piTFT screen started out white and then switched to black at the end of boot sequences. Tis phenomena indicated that our initial configuration worked correctly.

Then we run the command `dmesg` to test if the touch screen of the piTFT was working. The screenshot of similar result is shown in figure 4. (The screenshot was taken when the Raspberry Pi did not connect with HMDI. But it is weird that the chip can not be recognized.)

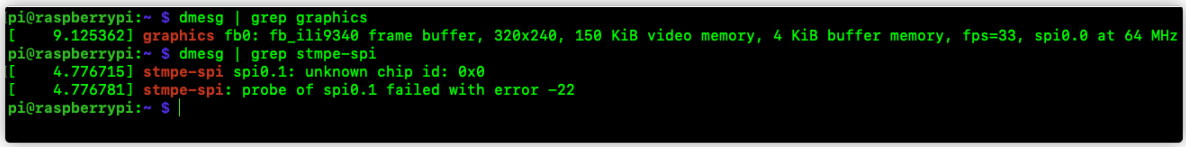


Figure 4: dmesg result

Then, we followed the manual to add a `udev` rule by modifying a bunch of files (copy & paste) and run some commands to stop and restart the touchscreen module. Finally, we could run the command `sudo evtest /dev/input/touchscreen` to start testing the touchscreen. The result is shown in figure 5.

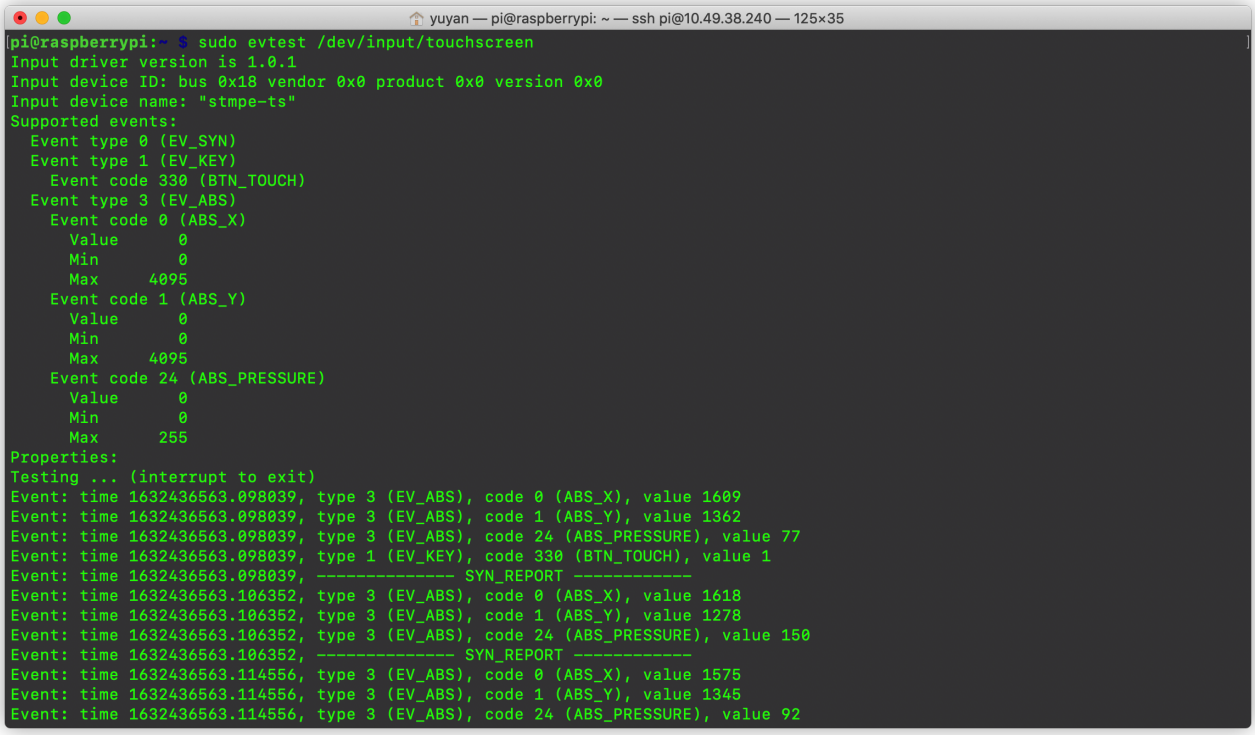


Figure 5: Touchscreen Test

After confirmed that the touchscreen could be used normally, we turned to calibrate the piTFT display and then make the Linux console window start on the piTFT by modifying a bunch of files(copy & paste again). Eventually, we saw the console displayed on the piTFT after the reboot. The result is shown in figure 6.

Figure 6: The terminal on the piTFT

We started to check the audio setting and tried to play the video with `mylpayer`. We got a JBL speaker from TA and connected it to the Raspberry Pi with an audio cable. Next, we executed the command `aplay /usr/share/sounds/alsa/Noise.wav` to test the audio and heard the noise that indicated the audio worked correctly. And then, we used the `myplayer` applocation to the video on the different platform, including the piTFT, the monitor and `ssh shell` with the command `sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop bigbuckbunny320p.mp4` (The option `/dev/db1` specifies the playback device which can be looked up by the command `ls /dev/fb*`). This command leads us to the most important problem of this part.

The root cause of the problem is whether to carry the `sudo` option when executing the command. If we run the command with `sudo` in the terminal on the piTFT, there would be no sound. Only when we run the command without `sudo` will we hear the soud. The situation in the monitor was similar to the piTFT. However, things changed when we run the command through `ssh shell`. If we didn't attach the `sudo` option in the head of the command, we could only heard the sound without the image of the video. Only if we did attach the `sudo` could the video and sound play simultaneously.

So at this point, we've successfully deployed piTFT and played the video. Further, we would control the video by pressing the buttons.

2.3 Control the video with Python Scripts and the FIFO File

In this part, we will introduce the experimental process with the main concepts that need to be grasped, and then, focus on describing the only problem we encountered.

The experimental process mainly consists of three steps:

1. Understand the options of the `mplayer` apllication (pause, seek and quit)
2. Establish the communication between Python scripts and the `mplayer` process through the created `FIFO` file.
3. Integrated the GPIO into the Python script for getting input from pressing buttons.

For the step 1, according to the reference in the manual, we found that, in order to fast forward or rewind the video, we should use the `seek` option with positive number or negative number (e.g., `seek 10`). The sign of the number determines whether to play forward or backward. As for the quit and pause function, the commands are just as same as their words.

For the step2, we created a `FIFO` file called `video_fifo` under the home directory. In fact, the `FIFo` file works as a pipe which can be leveraged for exchanging messages between different processes. In this specific case, the `FIFO` file we created would collect messages sent by the Python process and then deliver them to the `mplayer` process. The message generated by the Python script should be like `echo "seek 10 " > /home/pi/video_fifo` . Once this information channel was established, this step had almost been completed.

For the last step, we should get the basic understanding of GPIO at first. The full name of GPIO is **General Purpose Input/Output**. We found the GPIO pins of four buttons on the piTFT according to the piTFT with R-Pi Schematic. The result is shown in figure 7.

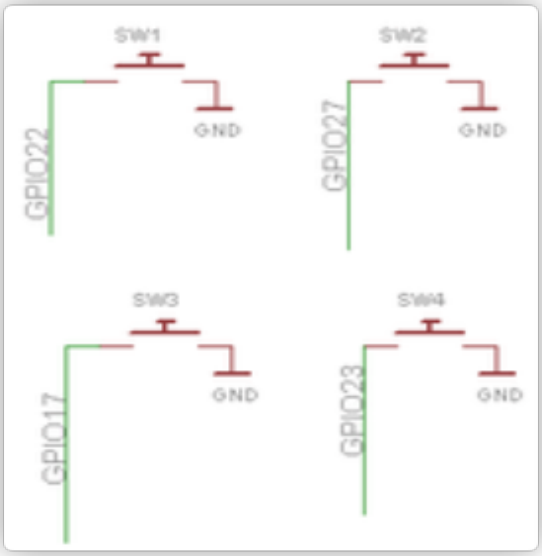


Figure 7: The terminal on the piTFT

Next, we should pay attention to the the Python module `rpi.GPIO`. We mainly used its two functions, `setmode` and `setup`. The `setmode` funtion is used to set up the numbering system. In this specific case, we setup the numbering system as BCM mode, which refers to the channel numbers on the Broadcom SOC. This is a lower level way and if we change the version of the Raspberry PI, we also need to change the pin number in our code. And then, we used the `setup` function to establish channels for each GPIO pin we used. Through the channel, the input of each button could be connected to the Python process. Once we detected a button was being pressed, the corresponding code would be executed. The complete code is shown in page 11.

To finish the requirements of this part, we still need a bash scrip which could simultaneously run the Python process and the `mplayer` process. It was in this step that we ran into the biggest problem. The original version of code is shown in the figure 8.

```
# !/bin/bash
SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop -input file=/home/pi/video_fifo
/home/pi/bigbuckbunny320p.mp4 & python3 /home/pi/lab1_files_f21/py_tests/my_python/video_control.py
```

Figure 8: Original Bash Script Code

When we demonstrated this script to the TA, we were surprised to find that the `myplayer` process did not end automatically when the script process ended. After discussing with the TA, we did not come up with a solution. So, we turned to the professor for help. Of course, the professor figured out this. In our code, we placed the `myplayer` process in front and the Python process behind. When we pressed the quit button, it firstly ended the Python process. And then, the bash script was ended. But the `myplayer` process was leaved as an orphan process because no one noticed the Linux system to end it. So, it would get stuck there, not knowing what to do, until we forced it to stop by pressing `ctrl-c`.

The solution to fix this problem is really easy. Just switch their position. In this case, after the quit button pressed, the Python process would end. And then, it would trigger the end of the bash script which would also end the `myplayer` process. The correct version of the code is shown in Figure 9.

```
# !/bin/bash
python3 /home/pi/lab1_files_f21/py_tests/my_python/video_control.py &
SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop -input file=/home/pi/video_fifo
/home/pi/bigbuckbunny320p.mp4
```

Figure 9: Correct Version of Bash Script Code

3. Conclusions

In this Lab1, we have successfully completed all the parts with good results (the first team passed).

The configuration process for the Raspberry Pi was a lot more convoluted than we expected. However, the complexity of scripts is still within our capacity. Thus, considering the overall workload of the two-week experiment, we would suggest that the workload of the second week could be increased as Python coding involved in this section should be a bit too basic for the students in this class. Personally, I (Yu Zhang) would like to suggest that principle for modifying the boot sequence and other configuration processes could be clearer. Most students, including us, just copied and pasted the commands and configurations from the lab manual. Even though they finished the experiment smoothly, they may still own limited understanding with the Linux system.

The biggest insight we obtained from the whole experiment is that the Linux system is a highly customizable system. Even the boot sequence can be modified at user's will. Of course, users should own the access rights of the super admin. This also places additional demands on the user, requiring a higher level of operating system and programming knowledge background. No wonder that the Linux system has remained viable for so long and is always the dominant operating system in industry.

The most surprising result I got is the process ending sequence mention in page 7. I never thought there would be a time when a process would be orphaned and stuck. This also shows the high tolerance of the Linux system. As the professor said in the class, the Linux system will always execute what you have executed. Be Careful! But **Don't Panic**. The worst case is to start all over again. So, Remember to Make the **Backup**!

Overall, I think this is a very good start experiment for this course. Not only do we get a first glance at the Linux system, but also we can appreciate the Raspberry Pi's charm. It only costs $35 to support most computer functions with the Linux system installed. At last, we must solemnly thank all the **TAs** and **Prof. Skovira** for all the help they provided during the experiment.

4. Code Appendix

1. fifo_test.py

```
# Yu Zhang yz2729
# Xinyu Wu xw586
# Lab 1  Date: 09/16/21
import subprocess
import os
while True:
    # Get inpu from the user
    command = input("Please input a command: ")
    print(command)
    if command == "pause":
        # Pause
        cmd = 'echo "pause" > /home/pi/video_fifo'
        os.system(cmd)
    elif command == "get_file_name":
        # Get File Name
        cmd = 'echo "get_file_name" > /home/pi/video_fifo'
        os.system(cmd)
    elif command == "speed_set":
        # Speed Up
        cmd = 'echo "speed_set 10" > /home/pi/video_fifo'
        os.system(cmd)
    elif command == "quit":
        # Quit
        cmd = 'echo "quit" > /home/pi/video_fifo'
        os.system(cmd)
        break
print("The viedo is end!")
```

2. one_button.py

```
# Yu Zhang yz2729
# Xinyu Wu xw586
# Lab 1  Date: 09/16/21
import RPi.GPIO as GPIO
import os
import time
# Set Numbering System to BCM mode
GPIO.setmode(GPIO.BCM)
# Setup the GPIO channel
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    time.sleep(0.2)
    # Detect input
    if(not GPIO.input(17)):
        print("Button 17 has been pressed")
```

3. four_button.py

```
# Yu Zhang yz2729
# Xinyu Wu xw586
# Lab 1  Date: 09/16/21
import RPi.GPIO as GPIO
import os
import time
# Set Numbering System to BCM mode
GPIO.setmode(GPIO.BCM)
# Setup the GPIO channel
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 True:
    time.sleep(0.2)
    if(not GPIO.input(17)):
        print("Button 17 has been pressed")
    elif( not GPIO.input(22)):
        print("Button 22 has been pressed")
    elif( not GPIO.input(23)):
        print("")
        print("Button 23 has been pressed")
    elif( not GPIO.input(27)):
        print("")
        print("Button 27 has been pressed")
        break
```

4. start_video.sh

```
# !/bin/bash
# Yu Zhang yz2729
# Xinyu Wu xw586
# Lab 1  Date: 09/16/21
python3 /home/pi/lab1_files_f21/py_tests/my_python/video_control.py &
SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl -framedrop -input
   file=/home/pi/video_fifo /home/pi/bigbuckbunny320p.mp4
```

5. video_control.py

```
# Yu Zhang yz2729
# Xinyu Wu xw586
# Lab 1  Date: 09/16/21
import RPi.GPIO as GPIO
import os
import time
# Set Numbering System to BCM mode
GPIO.setmode(GPIO.BCM)
# Setup the GPIO channel
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 True:
    time.sleep(0.2)
    if(not GPIO.input(17)):
        # Pause
        print("Button 17 has been pressed")
        cmd = 'echo "pause" > /home/pi/video_fifo'
        print("Pause the Video")
        os.system(cmd)
    elif( not GPIO.input(22)):
        # Fast Forward
        print("Button 22 has been pressed")
        cmd = 'echo "seek 10 " > /home/pi/video_fifo'
        print("Fast forward 10 seconds")
        os.system(cmd)
    elif( not GPIO.input(23)):
        # Rewind
        print("Button 23 has been pressed")
        cmd = 'echo "seek -10" > /home/pi/video_fifo'
        print("Rewind 10 seconds")
        os.system(cmd)
    elif( not GPIO.input(27)):
        # Quit
        print("Button 27 has been pressed")
        cmd = 'echo "quit" > /home/pi/video_fifo'
        print("Quit the Video")
        os.system(cmd)
        break
```