Kuwait University, College of Science
Computer Science Department

KernelPi

Written By: Abeer Alsafran

Date: December 27, 2024

Abstract

Kernel Pi is a project focused on developing custom device drivers for hardware components using a Raspberry Pi. By leveraging the Linux kernel and C programming, it aims to enable seamless communication between the Raspberry Pi and the hardware. The main contribution in this work is that we have achieved to deliver for the user a seamless coding experience to make it easy for the user to implement a code that sets up the hardware components. In this work we focused on a water sensor, a LED, and a buzzer to build the circuit. It is an alarm system that senses the water and if it gets to some level then the sensor will trigger the LED and the buzzer to light and make an alarm sound, respectively. We propose a tool that can be utilized to alert clients in case of any emergency related to rainwater or leaking water in general.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Since we are approaching winter and heavy rains are coming, then we need to take care of high possibility to secure and maintain critical places to prevent the rainwater from approaching these places.

1.2 Scope

For that, the proposed solution is a water sensor with a customizable alert system was built from scratch using RaspberryPi to make sure to act on spot.

1.3 Outcomes and Objectives

1- Kernel device driver to communicate with hardware.
2- Water sensor circuit with multiple components.
3- User program to check the status of the security water sensor.

1.4 Methodology

In this project there are 3 phases, the first phase is building the circuit which includes the hardware components. The second phase is to code the device driver that will communicate with the hardware. The last phase is to build the user program to check status of the sensors.

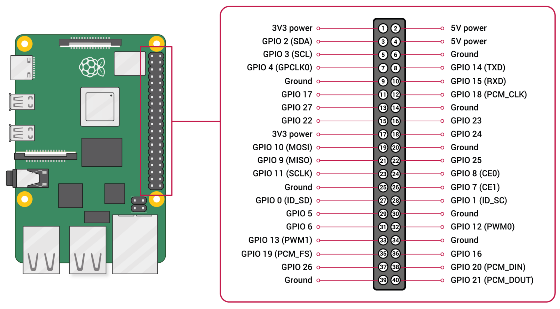


Figure 1.1: GPIO pins

1.4.1 Phase 1: Circuit building

In the first phase building the circuit was the point that we have to do to start this project. It was built after searching and investigating from different resources. In this phase there were a-lot of searching on how to connect the assets i.e. components together. For example, Figure 1.1 shows the GPIO pins and shows each pin and its number.

1.4.2 Phase 2: Kernel code design

In the second phase things has been more clearer since the circuit was built and tested. The first step to be taken in this phase is to know how the ioctl() function works and how does it communicate with the user space [9]. After knowing how does it communicate with the user space, we need to check how to build the GPIO interface to get direct access to the pins from kernel space perspective. While trying to compile the device driver after setting the GPIO pins and setting the flow, some problems occurred. It seems that the GPIO pins numbers in the user space is totally different than the ones that are in the kernel space. There is a command that shows the table of the user space pins numbers and the corresponding pin number in the kernel space.This command is:

```
cat /sys/kernel/debug/gpio
```

1.4.3 Phase 3: User code design

In the last phase we wanted to test the device driver functionality by using the ioctl() function in the user space to trigger the device driver. It was functioning perfectly and it passes the test. The user program was built to create the alert system using the character device.

Chapter 2

Design

2.1 Circuit design

The design of the circuit was designed on a breadboard that is shown in Figure 2.1 [8].

2.2 Code design

2.2.1 Kernel

For each hardware component, a specific kernel code was written in C language. Using the kernel interface for the GPIO[2].

2.2.2 User

A high-level system calls were called to query the status of the components and act as an interface between the hardware and the kernel (piGPIO library) [11].

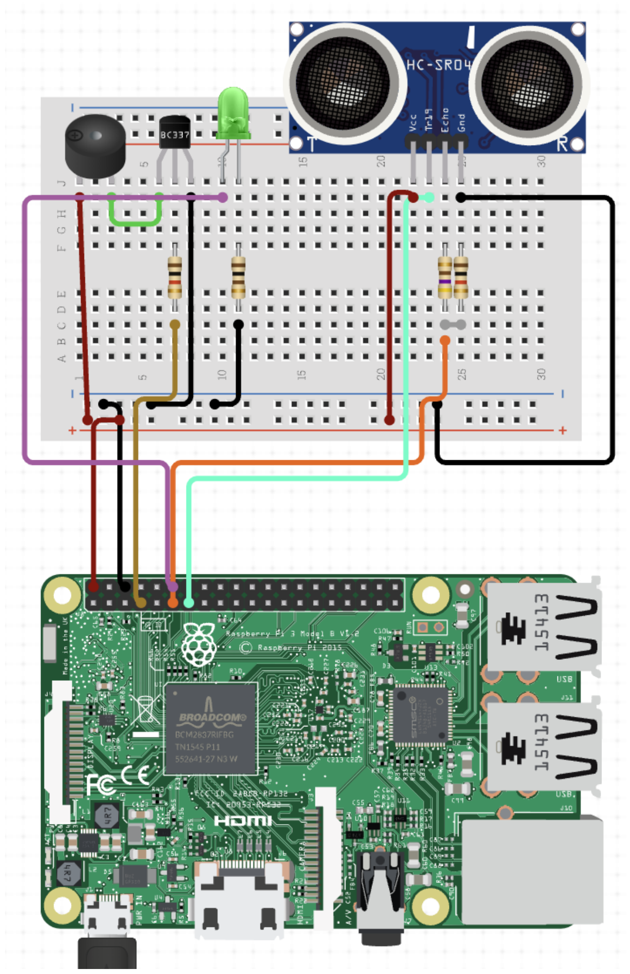


Figure 2.1: Circuit design

Chapter 3

Results and Evaluation

3.1 Sample output

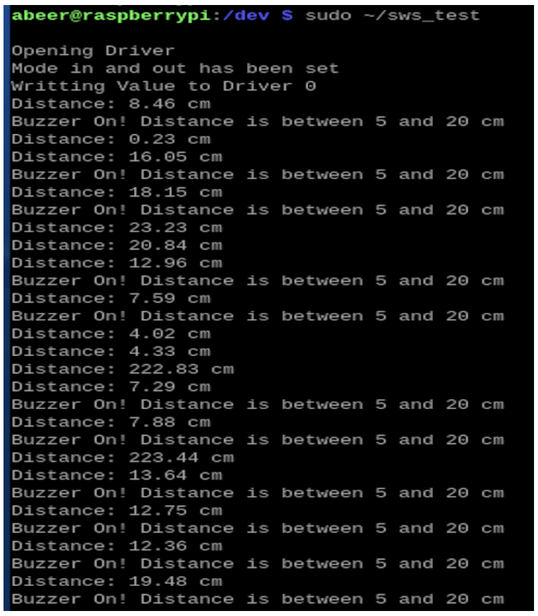
As you can see, when the distance is more than 5 cm it triggers the alarm (Buzzer) and the LED Figure 3.1.

3.2 Comparison between the implemented device driver and the normal code of initializing the GPIO pins using piGPIO library

In this section we have compared the speed and the system calls i.e. cost that was changed from the user code of initializing the GPIO pins from library such as piGPIO and the ioctl functions that has been provided by the character device driver. Using Strace to check for the different system calls that is being triggered.

3.2.1 Top 3 system calls in character device

Systemcall	***Occurrence***
ioctl	1384
write	51
clock_nanosleep	40



```
abeer@raspberrypi:/dev $ sudo ~/sws_test

Opening Driver
Mode in and out has been set
Writting Value to Driver 0
Distance: 8.46 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 0.23 cm
Distance: 16.05 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 18.15 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 23.23 cm
Distance: 20.84 cm
Distance: 12.96 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 7.59 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 4.02 cm
Distance: 4.33 cm
Distance: 222.83 cm
Distance: 7.29 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 7.88 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 223.44 cm
Distance: 13.64 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 12.75 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 12.36 cm
Buzzer On! Distance is between 5 and 20 cm
Distance: 19.48 cm
Buzzer On! Distance is between 5 and 20 cm
```

Figure 3.1: Sample output

3.2.2 Top 3 system calls in piGPIO library

Systemcall	***Occurrence***
munmap	1318
ioctl	96
rt_sigaction	63

The most frequent system call was ioctl() in the device driver implementation, which makes sense since the user space was using it to trigger the device driver functionality (Figure 3.2).
On the other hand, the most frequent system call in the piGPIO library is the munmap() (Figure 3.3) function shall remove any mappings for those entire pages containing any part of the address space of the process starting at addr and continuing for len bytes[13]. We can see that the overall average time for piGPIO library is less that the overall time in Figure 3.10 that the piGPIO library has less latency overall.

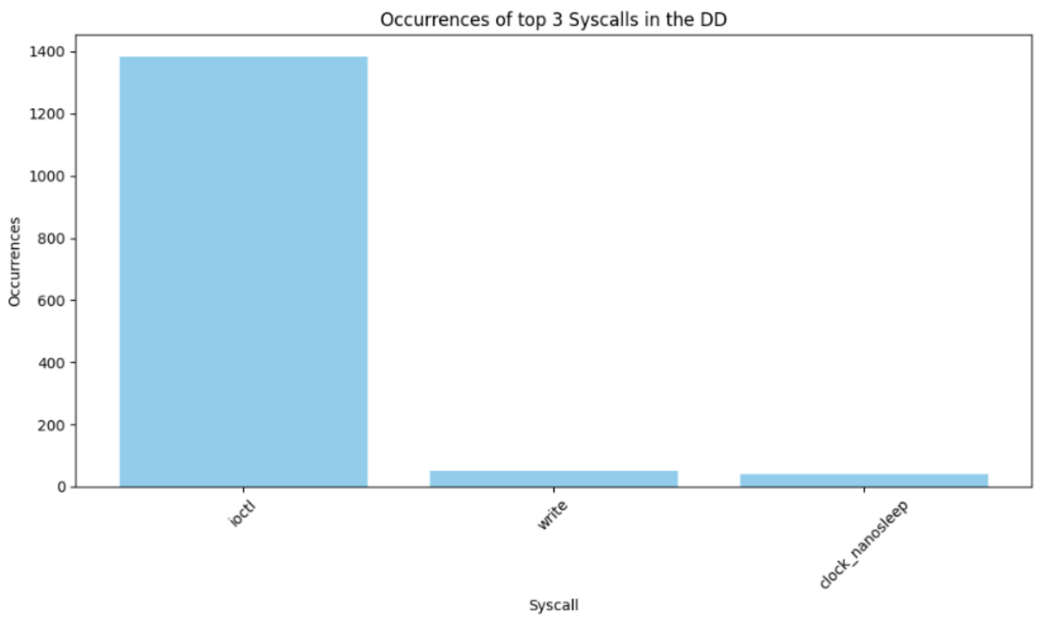


Figure 3.2: Occurence of top 3 of DD

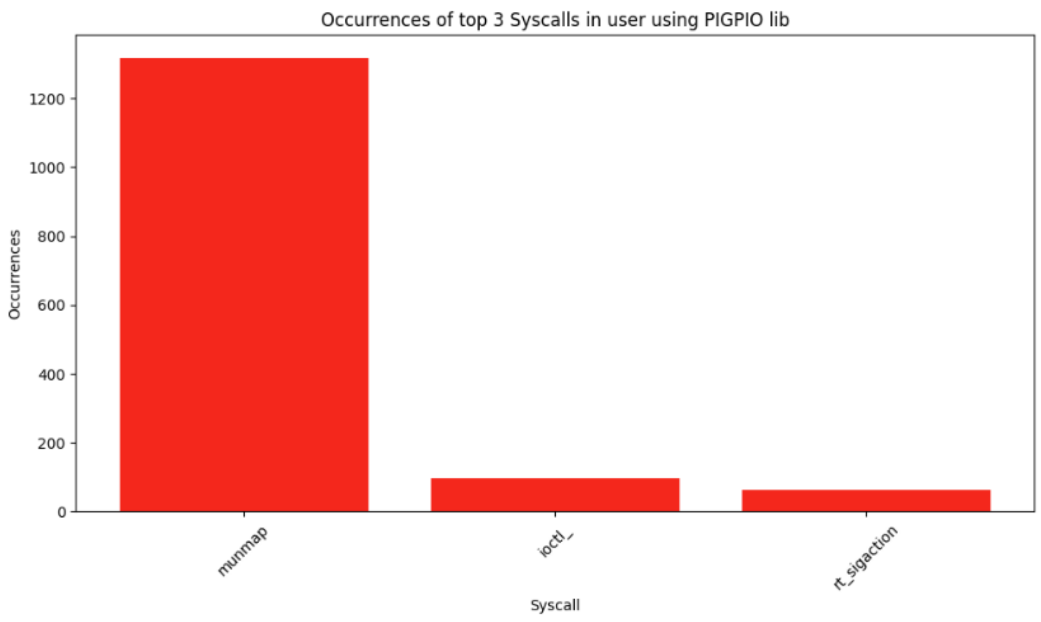


Figure 3.3: Occurence of top 3 of piGPIO lib

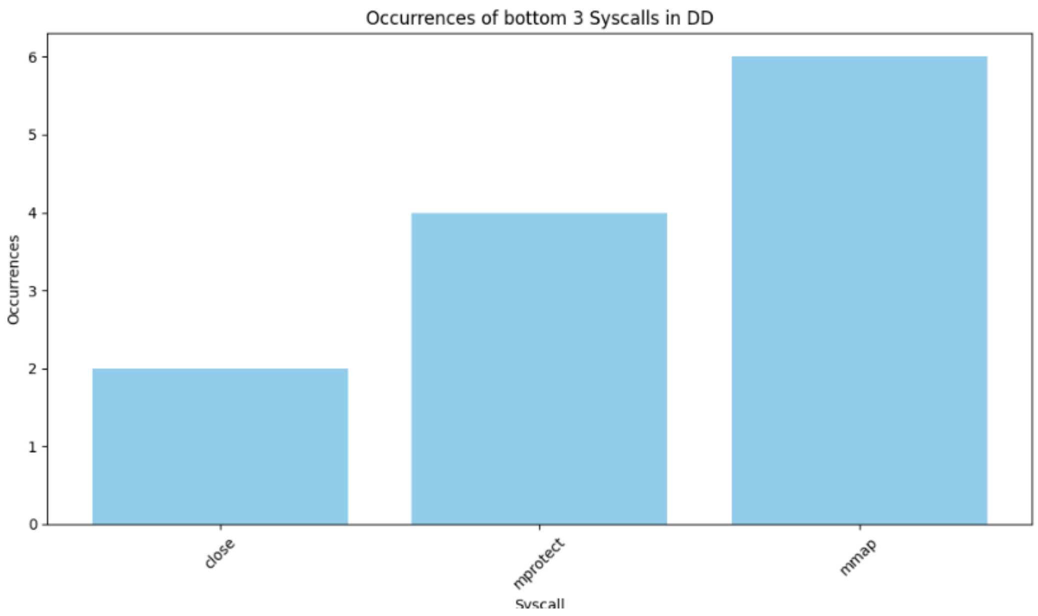


Figure 3.4: Occurence of bottom 3 of DD

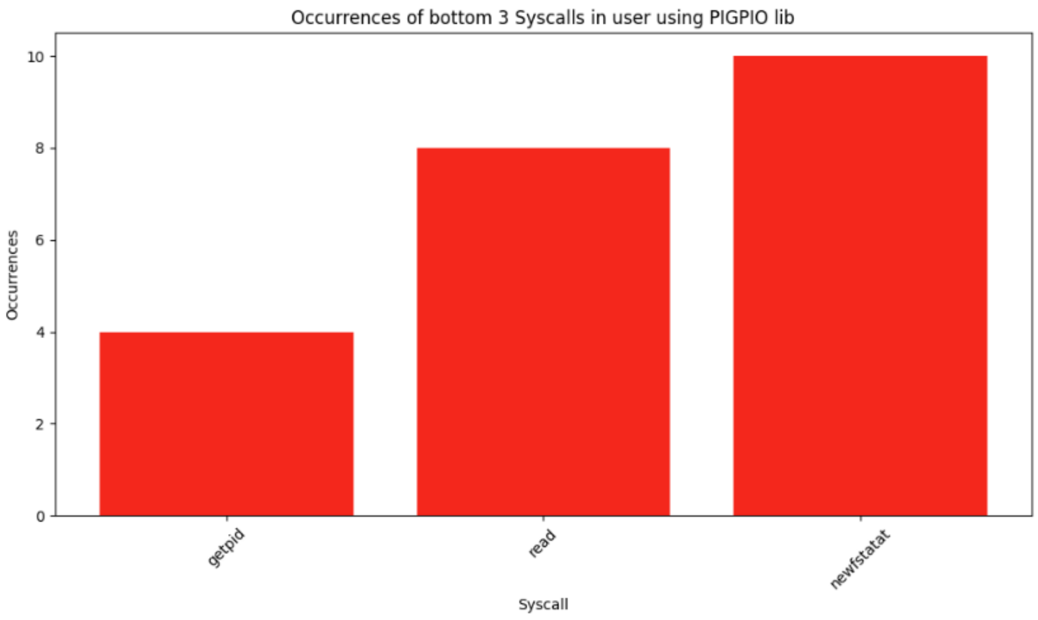


Figure 3.5: Occurence of bottom 3 of piGPIO

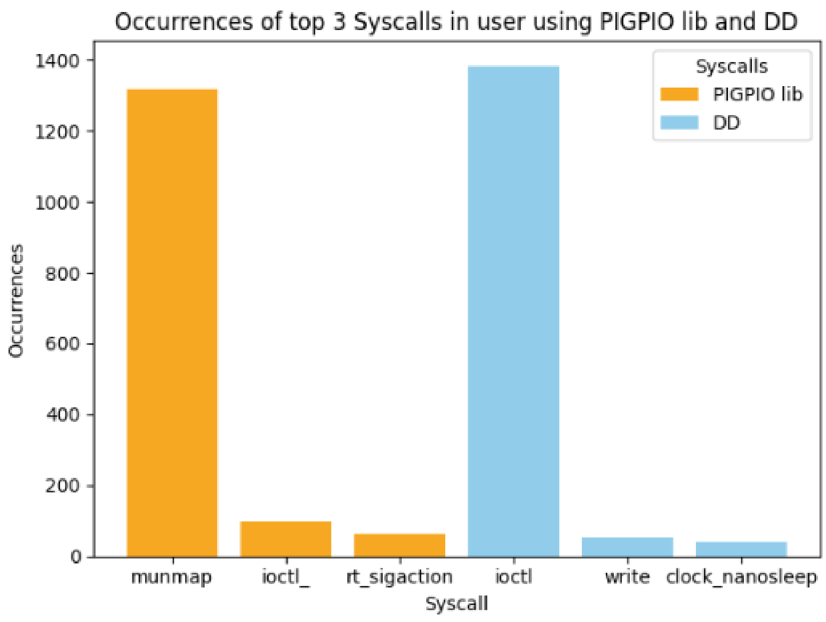


Figure 3.6: DD x piGPIO for top 3

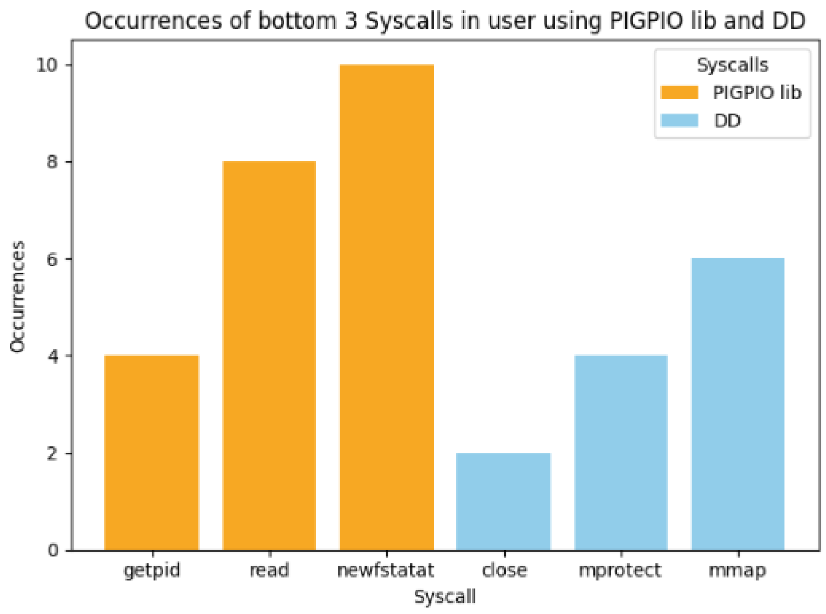


Figure 3.7: DD x piGPIO for bottom 3

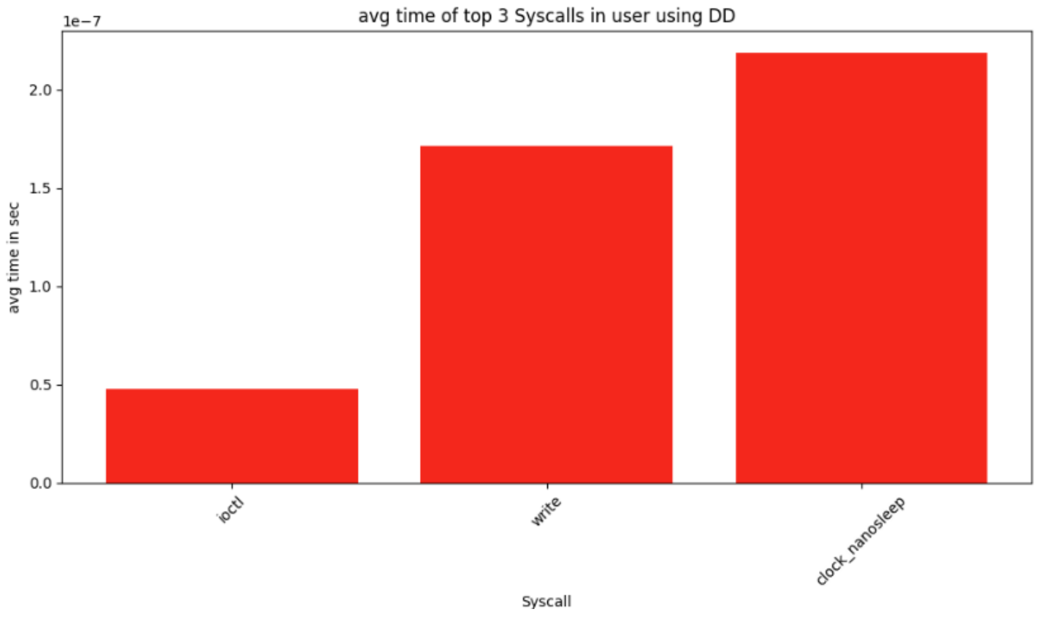


Figure 3.8: Average time of top 3 in DD

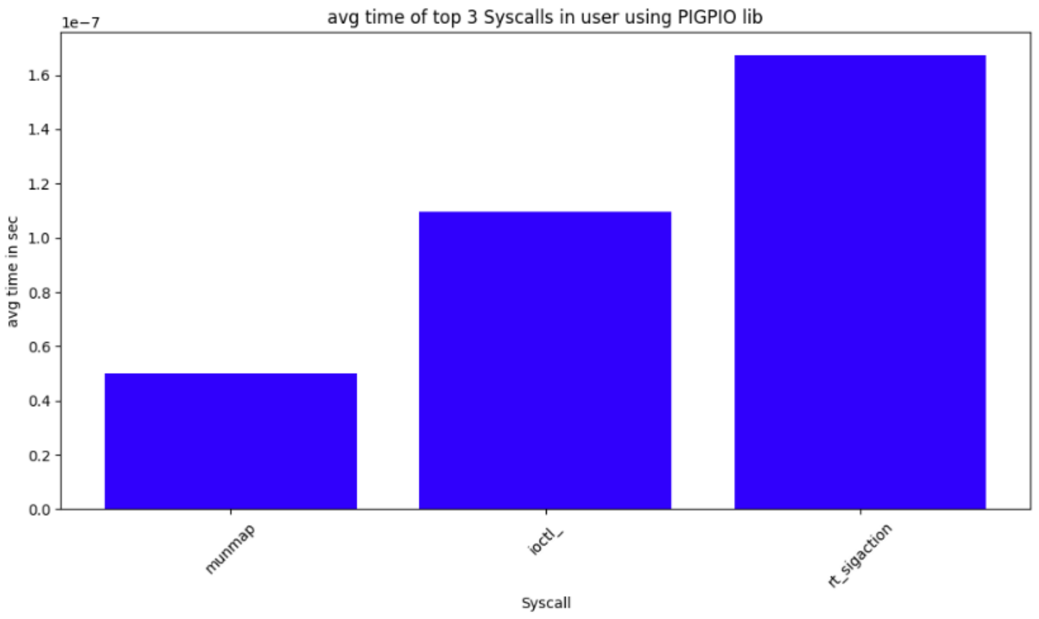


Figure 3.9: Average time of top 3 in piGPIO

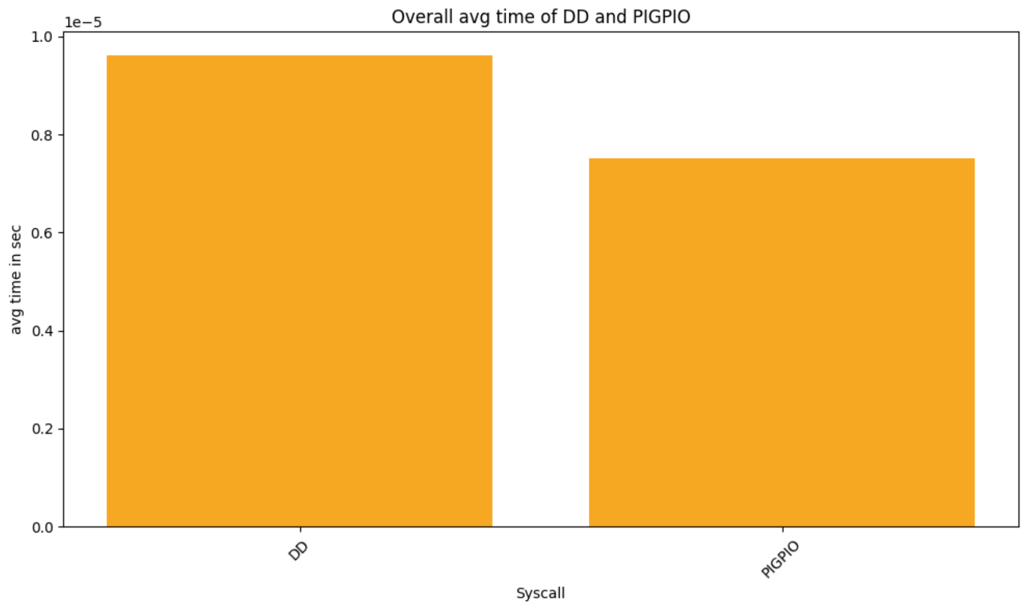


Figure 3.10: Overall Average time

Chapter 4

Conclusions

In the first phase it was hard to start since building circuits is not our special field as computer scientists, yet once getting used to it, it became more easier. The second phase was a very significant phase, since not all the libraries are supported by the C language are still working, some of which get deprecated. So, most of the time was spent exploring new libraries that are supported by Raspberry Pi. The final phase was using the newly created device drivers in a user program in a very abstract way that helps the user use it without the overhead of implementing the code for hardware components and initializing multiple GPIO's. Combining all these three phases, we propose a tool that can be utilized to alert clients in case of any emergency related to rainwater or leaking water in general.

4.1 Recommendations ans lessons learned

- Not to waste to much time searching for GPIO library, check the latest one.
- Use tools and components that suits the requirements.
- Sometimes to make something work you have to try different approaches, but this may affect the work you are currently doing in terms of progress.

4.2 Future Work

Expand the current circuit to add more components and make it more dynamic to adopt more device drivers. To make one large ecosystem. Making the pins dynamic by letting the user set the number of the pin dynamically.

Bibliography

[1] beyondlogic.org. beyondlogic.org libgpiod on raspi. https://www.beyondlogic.org/an-introduction-to-chardev-gpio-and-libgpiod-on-the-raspberry-pi/.

[2] blog.lxsang. Gpio kernel. https://blog.lxsang.me/post/id/33.

[3] circuitbasics. circuitbasics lcd setup. https://www.circuitbasics.com/raspberry-pi-lcd-set-up-and-programming-in-c-with-wiringpi/.

[4] circuitdigest.com. Circuit digest. https://circuitdigest.com/microcontroller-projects/interfacing-water-level-sensor-with-arduino.

[5] circuitdigest.com. Circuit digest. https://circuitdigest.com/microcontroller-projects/raspberry-pi-mcp4725-dac-tutorial.

[6] circuitdigest.com. Circuit digest. https://circuitdigest.com/microcontroller-projects/raspberry-pi-ir-sensor-tutorial.

[7] circuitdigest.com. Circuit digest. https://circuitdigest.com/simple-raspberry-pi-projects-for-beginners?page=12.

[8] circuito.io. Circuit design for makers. https://www.circuito.io/.

[9] embetronicx. embetronicx.com. https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/.

[10] emlogic. emlogic.no. https://emlogic.no/2024/09/linux-drivers-getting-started-with-gpio-on-raspberry-pi-5/.

[11] framagit.org. framagit.org libgpiod. https://framagit.org/cpb/example-programs-using-libgpiod/-/tree/master.

[12] huihoo. docs.huihoo.com. https://docs.huihoo.com/doxygen/linux/kernel/3.7/structfile__operations.html.

[13] IEEE. munmap - unmap pages of memory.

[14] instructables. instructables.com. https://www.instructables.com/Water-Level-Monitor-With-Raspberry-Pi/.

[15] ipilcher. ipilcher modules. https://github.com/ipilcher/n5550/tree/master/modules.

[16] Johannes4Linux. Linux$_{d}river_{t}utorial_{l}egacy$.

[17] Alessandro Rubini Jonathan Corbet and Greg Kroah-Hartman. *Linux Device Driver*. O'Reilly, 2005.

[18] kernel. kerneli2c. https://www.kernel.org/doc/Documentation/i2c/dev-interface.

[19] kernel.org. kernel.org. https://www.kernel.org/doc/html/latest/driver-api/gpio/index.html.

[20] Robert Love. *Linux Kernel Development*. Pearson, 2010.

[21] newbiely. newbiely.com. https://newbiely.com/tutorials/raspberry-pi/raspberry-pi-water-sensor.

[22] ostconf. ostconf.com. https://ostconf.com/system/attachments/files/000/001/532/original/Linux_Piter_2018_-_New_GPIO_interface_for_linux_userspace.pdf?1541021776.

[23] P3Judr4i7QI. Legacy c gpio pins.

[24] raspberrypi. www.raspberrypi.com. https://www.raspberrypi.com/documentation/computers/linux_kernel.html.

[25] raspberrypi.stackexchange. raspberrypi.stackexchange linux headers. https://raspberrypi.stackexchange.com/questions/149113/files-list-file-for-package-linux-headers-6-6-51rpt-rpi-v8-is-missing-final.

[26] raspberrypi.stackexchange.com. raspberrypi.stackexchange gpio pins c. https://raspberrypi.stackexchange.com/questions/101718/control-gpio-pins-from-c.

[27] stackoverflow. stackoverflow-setup-i2c. https://stackoverflow.com/questions/52975817/setup-i2c-reading-and-writing-in-c-language.

[28] techexplorations. techexplorations.com. https://techexplorations.com/raspberry-pi/how-to-compile-a-custom-raspberry-pi-os-kernel/#h-a-use-case-for-a-custom-kernel.