Instituto Tecnológico de Costa Rica

Computer Engineering

Operating Systems Principles

Project 2: Robotic Finger

Professor

Diego Vargas

Students

Fabián Astorga Cerdas - 2014040808

Javier Sancho Marín - 2014159997

Oscar Josué Ulate Alpízar - 201229559

Date

May 28th, 2018

1 Introduction

One of the main reasons of an Operating System existence is to let the user to interact with devices. For that, the O.S. has to implement the so called drivers to understand and interact with those devices. Commonly, devices have the firmware layer that let the system operate with them [3].

The driver is essentially a piece of software very close to the hardware with the sole purpose of telling the Operating System on how to interact with some specific type of hardware. In the case of this project, an USB driver was implemented to control an Arduino.

Drivers can be divided in several categories, but the main categorization is probably the following:

1.1 Block Device Drivers

These drivers are made for devices that support a file system, therefore they receive blocks of data and memory positions from the Operating System to make the actions needed [1].

1.2 Character Device Drivers

This devices just receive a stream of data from the Operating System. These are simpler devices that can have a mapped memory for controlling them [1].

The driver made in this project fits well inside the *Character Device* category, because what is does is to send a stream of data to the Arduino to control the finger.

1.3 Program Description

A Robotic Finger has to me made to enter a PIN on an application that runs in a tablet with the Android Operating System. The application needs to have tree different keyboard sizes (1cm $\times$ 1cm, 2cm $\times$ 2cm and 4cm $\times$ 4cm). The finger should be controlled by an Arduino controller, which is controlled by the team-made driver mentioned before.

To enter the PIN to the software, there is an interpreter written in C with the *lex* and *yacc* libraries. Then the result is sent to the rflib.h team-made library that is in charge of opening the port and send the commands to the Arduino using the driver.

The PIN is randomly made by the application in Arduino. There is no communication between the application and the system in C, therefore is a manual job to input the PIN in the configuration file of the Finger.

2 Development environment

2.1 Arduino IDE

As mentioned in the introduction, the physical representation of the bridges will be made with Arduino. This is a development board for hardware prototyping. The main purpose of this board is to help the developers to implement multidisciplinary systems. The Arduino board has a micro controller in charge of all the logic needed to represent the bridges. The Arduino code is written in a Java variant specially made for the board.[1][2]

2.2 Atom IDE for the main system in C

Atom is a very good text editor that lets the developer code in almost any language they want. In this case, the chosen language is C. Other important reason to use Atom as the editor is because it has great interaction with GitHub. It lets to use Git with ease which is a really important subversion environment.

2.3 Android Studio for the application APK

Android Studio was the preferred IDE for the FingerMe application development because is the main editor used for Android development. It was elected over other IDEs because it has a lot of documentation, it lets the development team to create visual interfaces in an easy way and provides good emulators to test the results.

3 Continuous learning attribute analysis

3.1 Robotic Finger Device Hardware

The Device Hardware is a crucial part of the system. Without it, there is no way to test the driver functionality. There was some investigation about the

Servo functionality, as making theme move good was required for touching the screen of the tablet.

For making the finger, the team based the design in [2] and modified it so it met the requirements for our needs.

3.2 Driver

The driver is the core of the system. The team made the most research for it. To make the driver, a lot of Linux research was made and in the C language.

Some of the most interesting learnings was that there can be many kinds of drivers, with many capabilities and permissions from the Operating System to control certain things. There are kernel drivers and user drivers. Both with specific functionalities. The one made for this system fits into the kernel driver. This required a lot of research for the development, because a lot of knowledge is required of the Linux O.S.

3.3 Lex and Yacc

Lex and Yacc are two libraries specifically for creating interpreters and compilers such as the one needed for interpreting the input of the finger. They simplify the job incredibly and using them simplifies the whole work. Some research was needed, but the team had some background in that area. They are incredibly recommended libraries.

4 Program design

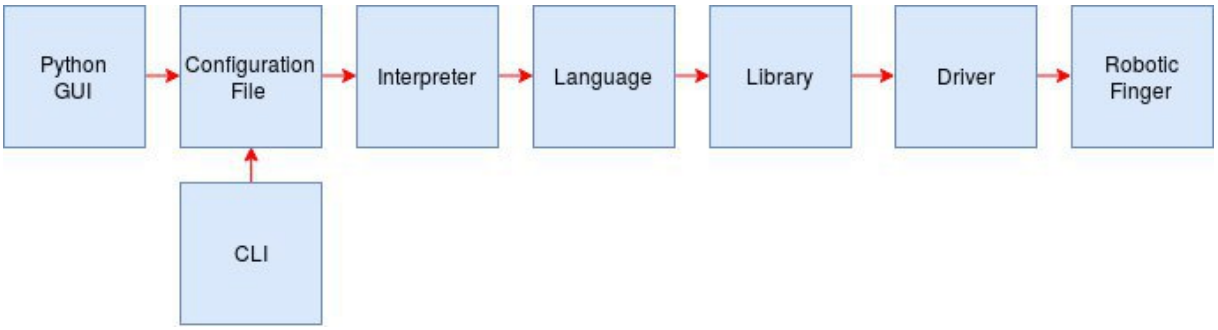

Figure 1: Workflow of the system.

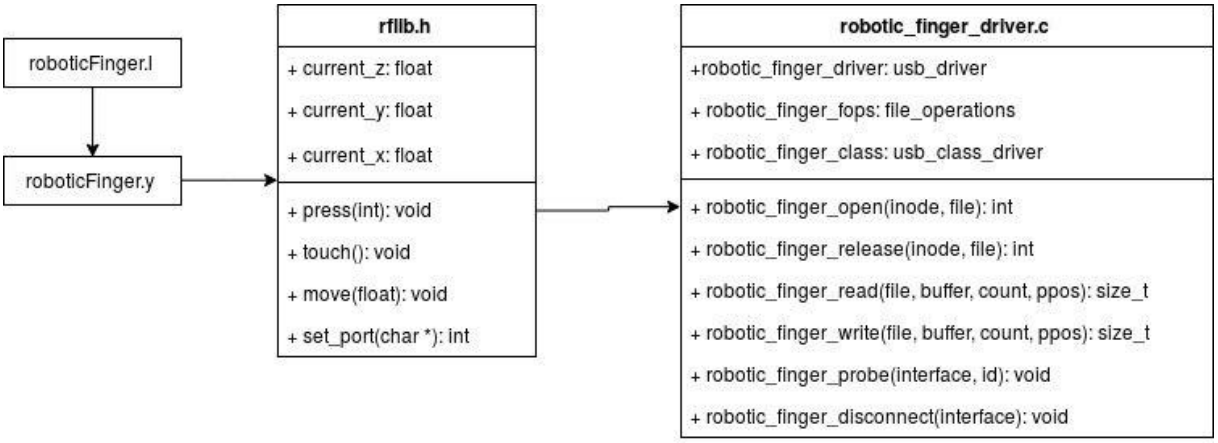

Figure 2: UML of the system.

4.1 Robotic Finger

The mechanical device that implements the Robotic Finger physical functions is based on an open source finger [2], but modified for our necessities. The mechanical components that allow three-axis movement were taken from the 3D model of the previously mentioned device and laser-cutted in order to adapt the parts to perform the required actions. The tip of the finger that touches the tablet is a home-made pointer. Instead, the device was designed with servo motors in mind, which are cheap and very easy to control, although not as precise as a stepper motor, but simpler to use. Nevertheless, the functional requirements state that the precision needed should be at least 0,5 cm (half the amount of the smallest keyboard button), thus using a servo motor based solution should have no negative impact on the performance of the finger. In addition to the mechanical device, an embedded device was required to control the actuators that move the finger itself. This role was fulfilled by an Arduino UNO, which was chosen because of its 5V supply (servo motors work a 5V tension supply), its integrated serial communication capability, and the rapid prototyping and development features of the Arduino environment. The driver should be also easy to develop since the Arduino UNO uses a simple Atmel ATmega8u2 as a USB/Serial controller for the ATmega328P on board microcontroller, which is also simple yet powerful.

4.2 Android Application

An Android application was made to use and test the Robotic Finger. It consists in a numerical pad, with square buttons of different sizes, depending of the choice of the user.

The tablet used consists in an ASUS 10-inch tablet. This tablet runs Android 4.1.2, so the application was made for Android 4, API 14. The application starts by default with the 1cm $\times$ 1cm screen resolution, but can be changed by a dropdown selector on the top of the screen. In the following figure, you can see a screenshot of the application.

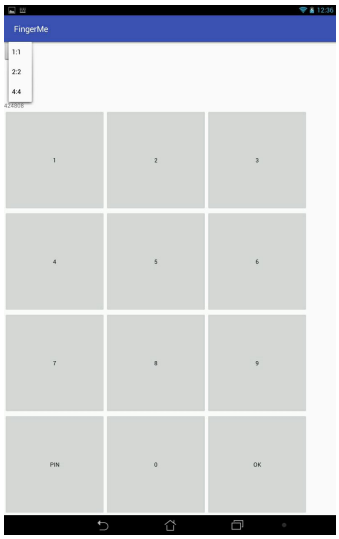

Figure 3: Screenshot of the Android application running on the ASUS tablet.

4.3 Language & Interpreter

The project objectives specify that the required language was very simple and it only has three types of instructions: *touch*, *press* and *move&press (map)*. As team we decided that the program need two more instructions *move* and *pin* to facilitate the run-time workflow, *move*: moves the finger to the specified number and *pin*: receives a PIN number and generates all the necessary movements to complete the inserted pin. So, the *move&press* instruction becomes a composite instruction *move* follow by *press*. The instruction reference is presented as follows:

Instruction	# args	Description
TOUCH	none	Taps the screen
PRESS t	1	Holds contact with the screen for t seconds
MAP n	1	Taps the screen and then moves finger to number position n
MOVE n	1	Moves finger to number position n
PIN p	1	Press the digits on the screen keyboard

To implement the interpreter a lexical analysis module was developed using flex that tokenized the specified keywords in the above table (both upper and lower case) and decimal numbers along with format characters such as whitespace and newline. The tokenized output of the lexical analyzer then becomes the input of a GNU/bison-generated parser that contains the allowed program structures and rules. If the current line is a valid instruction, corresponding actions are executed using the library. Program execution is dropped and an error reported otherwise. Note that the interpreter executes actions line-by-line, which means a line can only contain a single instruction.

4.4 Device Driver

The device driver is responsible for communicating with the embedded device that controls the robotic finger, using a USB connection. The device file will be called /dev/finger0. We developed a LKM (Linux Kernel Module) to interact with the device to send and receive information. The driver was developed at Kernel space. In Linux all drivers are treated as files, thus we implemented standard file operations for the driver file. The generation of the module is through a C file. This C file has the methods that describe the operations over the device file:

- **init**: Function that initializes the driver.
- **probe**: Function is executed when the device is connected and it is needed to test.
- **read**: Function called when reading the device.
- **write**: Function called when data is sent to the device.
- **open**: Function called to open the port.
- **release**: Function called when closing the device.

4.5 Robotic Finger Library

A library was made in order to communicate the interpreter with the device driver. This library contains the functions that are needed to write the PIN in the tablet application. These methods are simple file data transfer with the device file. In this part the methods requested by the specification are implemented:

- **TOUCH**: Function used when the system needs the robot to tap on the screen.
- **PRESS t**: Taps the screen for t seconds.
- **MAP n**: Taps on the tablet screen and then moves the finger to the number n.
- **MOVE n**: Moves the finger to a certain position.
- **PIN**: It is a compound function that taps 6 digits on the screen (6 is the length of the PIN). It uses a combination of the previously mentioned functions.

The chosen language for this library was the C programming language. It was chosen because its ease of use when creating the interpreter because of the background knowledge of the team using those libraries. Besides that, the C language let the programmer to use very close to the O.S. That is why the team uses C for the driver, that is why the rflib.h was written in the same language.

4.6 Graphical User Interface

The system's Graphical User Interface was developed in the Python Language with the Tkinter library for the ease of use to develop it. It is shown in the Fig. 4 and what it basically does is to write the data in the configuration file. When the user press the **Start Routine** button, on the background, what it does is to call the command line application.

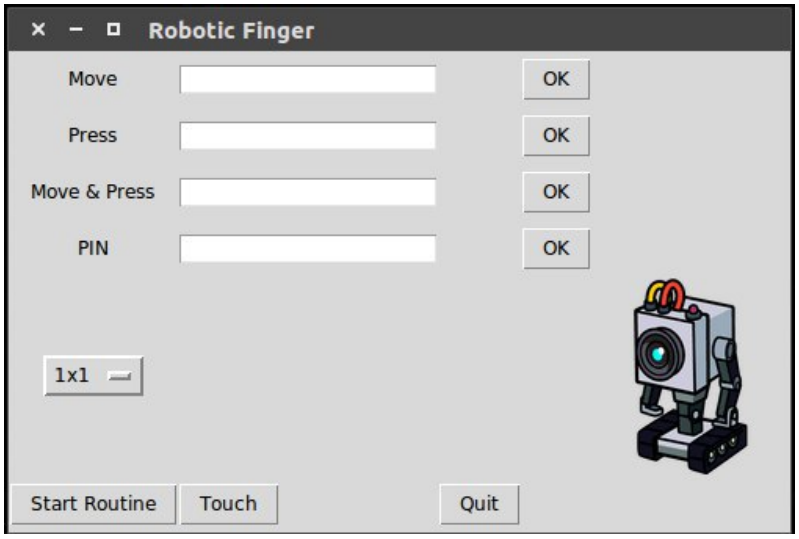

Figure 4: Graphical User Interface used to control the Robotic Finger.

5 How to use the program

To compile the executable files and prepare the program in order to control the robotic finger, use the configure script rf_conf in the root directory, with the following command:

```
$ ./rf_conf
```

After you execute the previous command, the driver and interpreter are ready to init the finger's execution. The program can be used by the user through the graphical user interface (see the figure 3). If you want to use the program manually, you have to make a configure file with the language proposed by the development team, go to interpreter directory and use the following command:

```
$ ./roboticFinger -c <config_file> -p <hardware_port>
[-s <keyboard_size>]
```

6 Activity log

Threads	Fabián Astorga	Javier Sancho	Oscar Ulate
Driver	4	7	5
Arduino	6	1	2
Robotic Finger	9	15	8
Scripts	0	5	4
Configuration file	0	1	5
Interpreter	8	5	0
Library	1	13	5
GUI	8	0	2
Android App	0	0	8
Log File	2	2	2
Communication	5	0	0
Documentation	2	1	4
Total hours	45	50	44

7 Final status

- Physical device:Complete movement in three coordinates to execute all the instructions created by the team see section 4.3.
- Device driver: A program developed in C programming language that interacts with the physical device, which is a USB type driver, has been completed. For more details see section 4.4.
- The device library: All the functions provided by the driver has been implemented in C programming language, that library interacts with the driver. See section 4.
- The language: 5 instructions have been created to manage correctly the implementation of the robotic finger, that instructions had been detailed in section 4.3.
- The interpreter: we used the tools Lex and Yacc to performance a good interpreter even with log file to see error messages. A short explanation

is in section 4.3.

- The graphical user interface: A friendly graphic interface was created using python programing language with all the options implemented in the library, this communicate with the interpreter to interact with the robotic finger. See section 4.6.

- The physical test program: Created in android environment, which has the facility to switch between the different screen resolution without previous configuration. See section 4.2

- Scripts: We build some files to make easier the preparation process, to clean, compile and make all the requirements actions to install the drive, including a set of correct permissions.

8 Conclusions

Software layers used for interactions with hardware are very important for computers as we know them. Without those layers, interacting with peripherals and external hardware devices could be very difficult or maybe even impossible. Providing the user with an interface that allows him or her to use an I/O device is a good way to limit what the user can do in order to reduce damages to the system.

The driver is works as a layer for separating the software and hardware layers of a system. Without it, every software would need to implement every driver they need to interact with every I/O device they would need, even the Disk, foe example.

9 Suggestions and recommendations

1. It is recommended to use Lex and Yacc libraries to create the interpreter. Those libraries facilitate incredibly the implementation of those pieces of code.

2. It is suggested to research about Linux Drivers and how it handle drivers. Trying to code the driver without a Linux background is close to impossible.

3. Divide the system in layers. Modularizing the system is great because no only creates a clean software, but also, let the driver to work for every other application that needs it. If the code was monolithic, it wold be impossible.

References

[1] Oracle, "Types of Device Drivers (Writing Device Drivers)", Docs.oracle.com, 2010. [Online]. Available: https://docs.oracle.com/cd/E19455-01/805-7378/6j6un034i/index.html. [Accessed: 28- May- 2018].

[2] T. Malkki, "4 AXIS ROBOT ARM DIY," 2016. Available: http://www.instructables.com/id/4-Axis-Robot-Arm-DIY/.

[3] "What is a Device Driver? - Definition from Techopedia", Techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/6824/device-driver. [Accessed: 28- May- 2018].