ITESO, Universidad Jesuita de Guadalajara

# Laboratory 1 | ALSA
# IoT for Embedded Linux

Team Members:
Rodrigo Ramos Romero IE715082
Diego Fernando Pérez González IE 736696
Ana Isabel Guillen González IE 739422
Daniel Jiram Rodríguez Padilla IE 703331

Professor:
Zamora Davalos Ricardo

26/10/2025

# ALSA

## Contenido

## Introduction

For this laboratory we had the opportunity of working directly with the different ALSA drivers and were able to satisfactorily see the results of a mailing-audio system implementation. Alongside the main objective of this Lab, which was getting more familiar with ALSA, we also put into practice more basic (but not less important) concepts and skills like file handling at a terminal level, TCP/IP connection through and Ethernet Network, C language programming, cross compilation, etc.

This document englobes the essential parts of the Laboratory, starting with what the original objectives are, a little bit of basic theoretical background to be able to grasp more context of the laboratory itself (since we made use of /implemented TCP sockets, multithreading, the ALSA drivers, etc.), the development of the Lab, the obtained results and finally some conclusions and references.

## Objectives

The objective of the practice is for the student to use elements covered in class such as processes, threads, inter-process communication mechanisms like sockets, FIFOs, pipes, shared memory, semaphores, and file handling.

Requirements:

- Using two boards, implement a solution that allows two users to communicate through audio.
- The connection between both boards must use sockets (TCP or UDP).
- The communication must emulate a send/receive audio file service. It must be initiated from the console by specifying the IP address of the recipient.
- The interface must include buttons for:
    - Increase Volume
    - Decrease Volume
    - Start Audio Recording
    - Stop and Send Audio
    - Play Received Audio
- The audio file must not be transmitted until the recording is finished, as indicated by the corresponding button.
- The solution must be **symmetric**, meaning either board can send a file.

## Theoretical framework

### ALSA

The Advanced Linux Sound Architecture (ALSA) is a software framework and component of the Linux kernel that provides audio functionality and sound card device drivers. Is the standard audio system in modern Linux distributions. Informally considered as the sound of Linux, it handles the part of the audio hardware in an API software, dealing also with the sound of the system [1].

## Sockets

Considered a software endpoint for network communication between two programs. It is the essential part that handles the mechanism that enables data to be sent or received over a network. Informally could be seen as a "virtual plug" that an application uses to connect to a network, since a socket is the bound to a specific network address [2].

## TCP

A socket binds an IP address and a port number to a **transport protocol**, in this case we used TCP, to facilitate data exchange. The Transmission Control Protocol (TCP) provides reliable, connection-oriented communication. Before any data is exchanged, TCP uses a three-step process to establish a secure and reliable connection between client and server:

1. Synchronize: Client sends a packet with the SYN flag.
2. Synchronize-Acknowledge: The server receives the SYN packet and then sends a SYN-ACK packet in response.
3. Acknowledge: The client receives the SYN-ACK packet (at this point the connection is established [3].

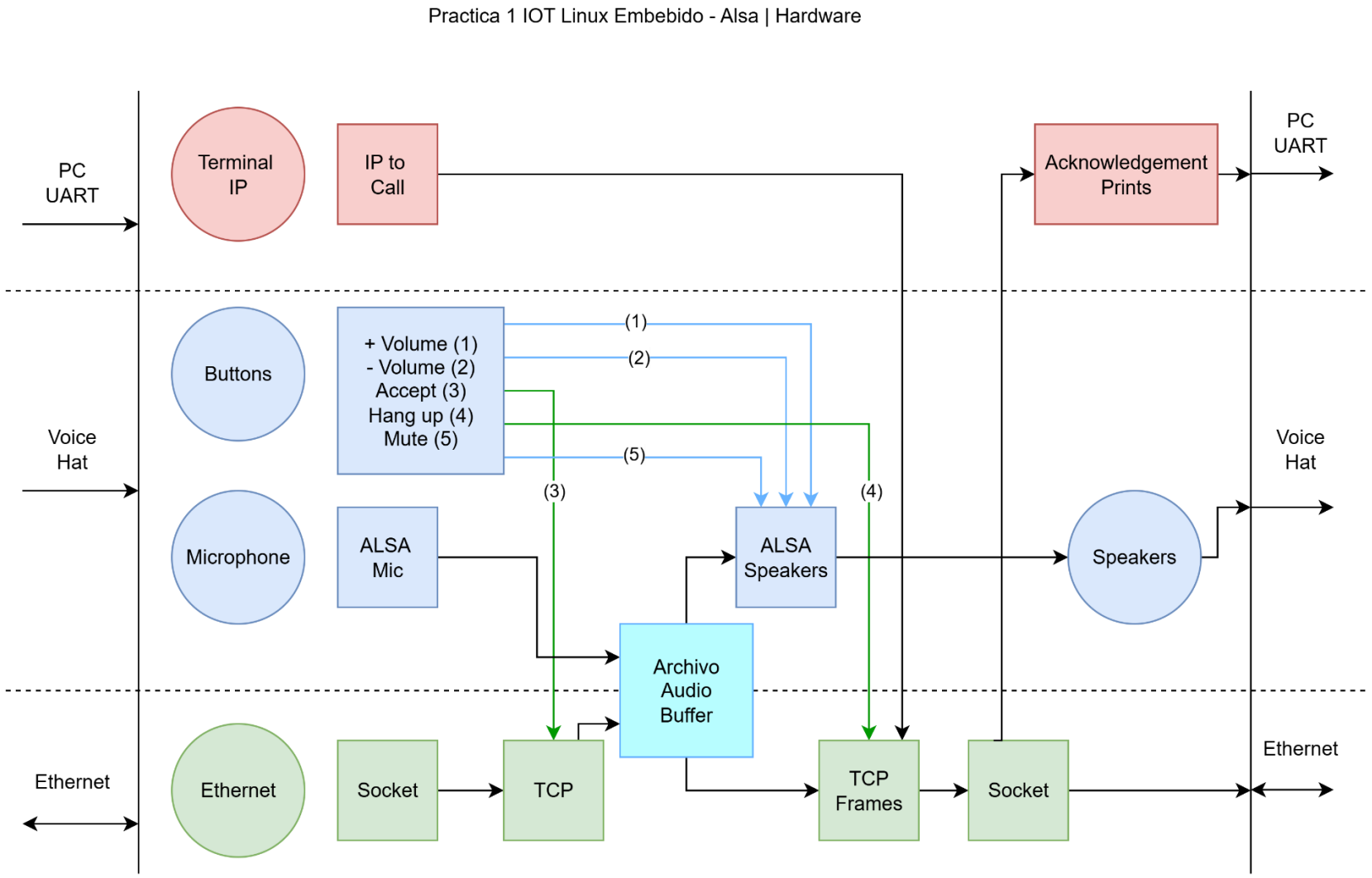
## Threads and Multithreading

A thread is the smallest sequence of programmed instructions that can be managed independently by a scheduler. For instance, a single program or process can consist of multiple threads, making it possible to perform multiple tasks concurrently. In contrast with a process, which has its own memory space and resources, threads exist within a process and share its resources [4].

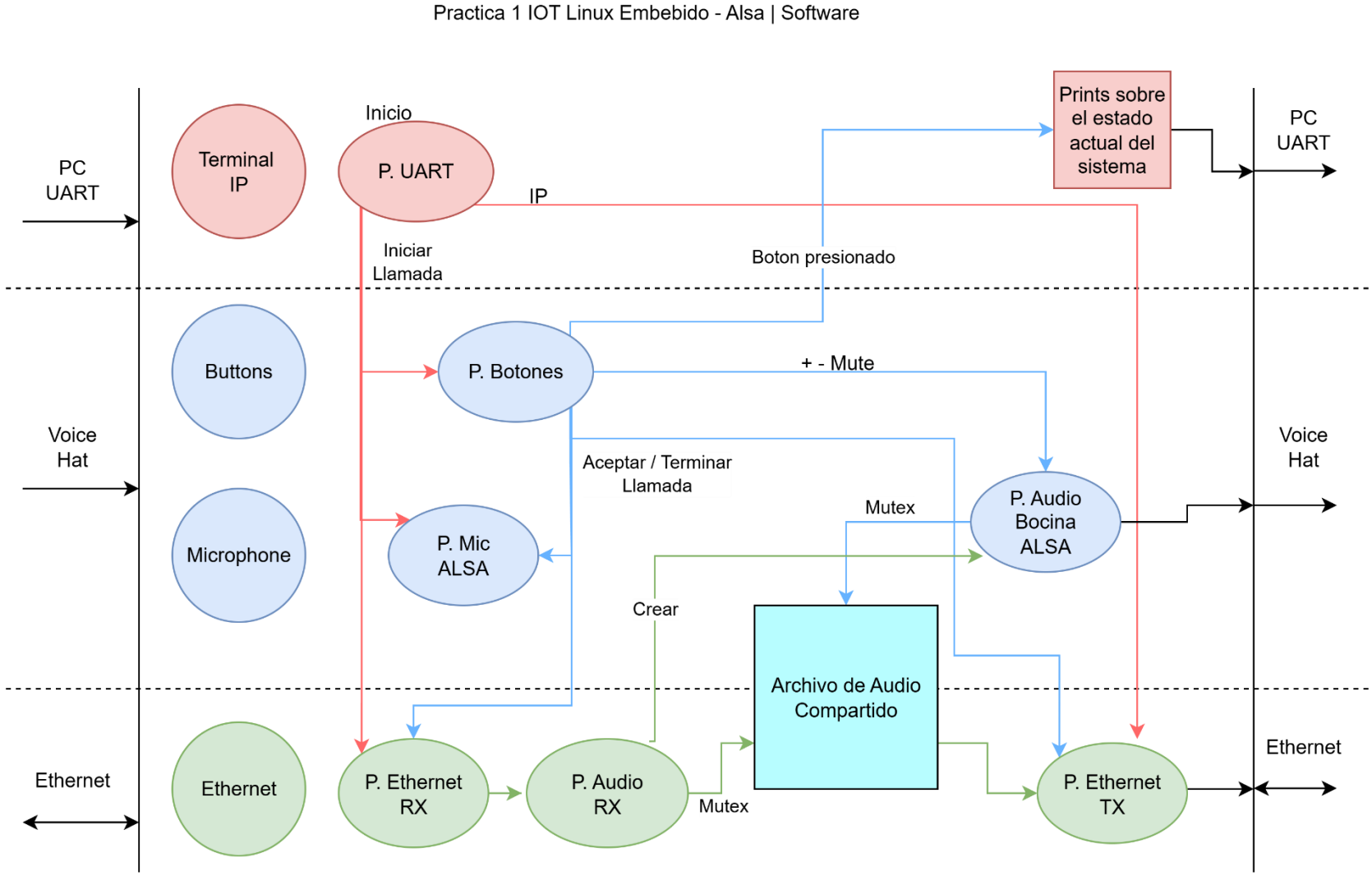
Some of the advantages of implementing multithreading are:

- **Improved responsiveness** (threads can be the responsible of handling long-running or blocking tasks in the background, keeping the main user interface free to respond, for example)
- **Better resource utilization** (in multicore for example, multithreading allows tasks to run in parallel, maximizing the hardware's capabilities)
- **Simplified communication** (the ability for thread to share memory simplifies the exchange of data between tasks) [5]

## Hardware Diagrams



## Diagrama de Software



# Development

## Creation and testing of network sockets

We started off by creating TCP sockets to first see and make sure that we were able to communicate between the two boards. We decided TCP over UDP because of the data integrity of the audio packets, so that what was recorded and sent to the other board, could be listened as exactly the same on the other board.

A simple ping kind-of-test was executed to prove the functionality of the sockets:

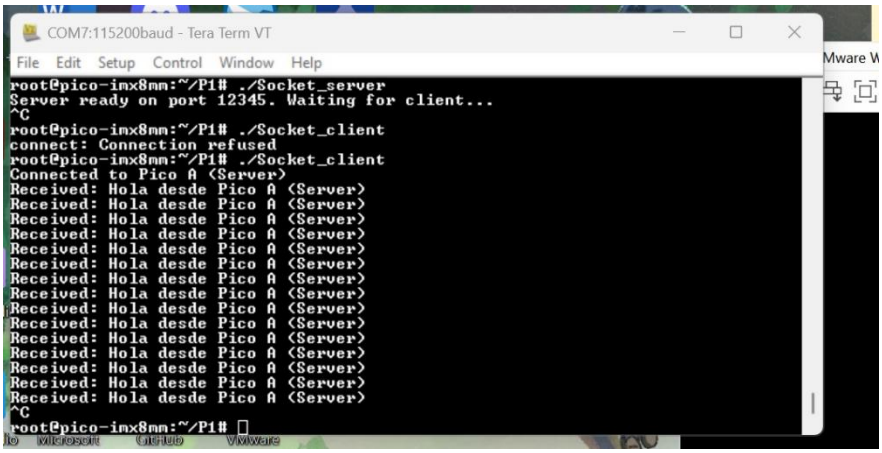



*Illustration 1. Client listening to Server*

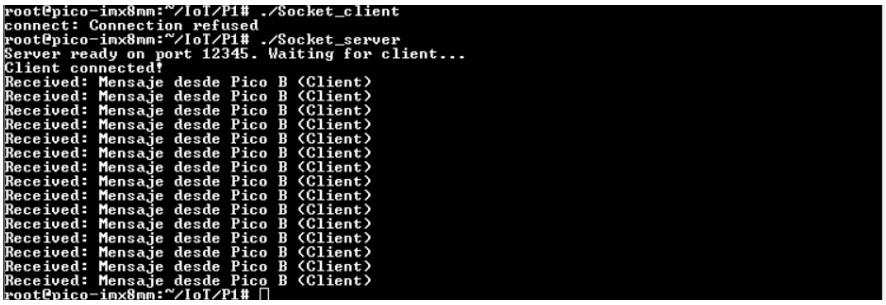



*Illustration 2. Server listening to Client*

## SW module that handles push-button events

For our system, we implemented a buttons driver, that handles all the events derived from the pressing of the different pushbuttons that the PICO-PI-IMX8M has. We basically listen continuously for press events from a Linux input device: "/dev/input/event0", which is the pushbuttons used for our system.

The three main things that this module does is:

- Reading hardware button inputs from "/dev/input/eventX"
- Triggers corresponding shell commands or scripts using "system()" (shown in next section)
- Sends feedback messages through a socket to the other board

## Process orchestration of several shell scripts

Inside the buttons driver, we also have the process orchestration of several shell scripts to execute some of the core partes of the system, Illustrations 3 to 6 show how this is implemented in C code.

```
case 207:
    printf("PLAY presionado (record start)\n");
    system("./Grabar_mensaje.sh &");
```

*Illustration 3. Execution of shell script to record a message*

```
case 407:
    printf("NEXT presionado (playback)\n");
    system("./Reproducir_mensaje.sh");
```

*Illustration 4. Execution of shell script to play an audio message*

```
case 103:
    printf("VOL+ presionado\n");
    snprintf(msg, sizeof(msg), "VOL+ presionado\n");
    system("./vol_up.sh");
```

*Illustration 5. Execution of shell script to turn up the volume*

```
case 108:
    printf("VOL- presionado\n");
    snprintf(msg, sizeof(msg), "VOL- presionado\n");
    system("./vol_down.sh");
```

*Illustration 6. Execution of shell script to turn down the volume*

## Application and Multithreading

We have both a client-side and a server-side, although they do basically the same, with the exception of the network connection (one actively connect to the server, and the server-side with for the client to connect) application that has a general logic for handling button events and messages.

Multithread is implemented for this application, using basically two threads:

- A **main thread** (basically what happen in the main loop) that handles network communication: prints or process incoming messages (either a .wav file or just string messages)

- A **button thread** that runs the function “Button_called()” in charge of detecting button presses and send the corresponding messages.

```
// --- Start button thread ---
pthread_t button_tid;
if (pthread_create(&button_tid, NULL, button_thread, &sock) != 0) {
    perror("pthread_create");
    close(sock);
    close(socket_connection);
    return -1;
}
```

*Illustration 7. Creation of button monitoring thread.*

```
void *button_thread(void *arg) {
    int sock = *(int *)arg;
    Button_called(sock);
    return NULL;
}
```

*Ilustración 8. Definition of the function that runs in the buton monitoring thread.*

# Results

## Screenshots on terminal for both devices

Fully symmetrical, both the server and the client could either record, send, receive and play audio:

```
root@pico-imx8mm:~# ./Messaging_System_Server
Server ready on port 12345. Waiting for client...
Client connected!
Escuchando botones: PREV, PLAY, NEXT, VOL+, VOL-, MUTE, PAIR...
PLAY presionado (record start)
[  265.499378] imx_sph0645_startup():
[  265.504615] imx_sph0645_trigger():
MUTE presionado (stop record)
[  271.312057] imx_sph0645_trigger():
[  271.315686] imx_sph0645_shutdown():
NEXT presionado (playback)
[  281.330449]
[  281.334158]
[  287.126449]
[  287.132235]
PREV presionado
mensaje.wav                                   100% 2172KB   2.1MB/s   00:01
[Client] Playing received voice note
[Client] Recording voice note...
[Client] Recording stopped
[Client] Playing received voice note
[Client] Voice note sent
NEXT presionado (playback)
[  328.124517]
[  328.128160]
[  331.504442]
[  331.510256]
VOL+ presionado
Volumen: 80% -> 90%
VOL+ presionado
Volumen: 90% -> 100%
VOL- presionado
Volumen: 100% -> 90%
VOL- presionado
Volumen: 90% -> 80%
VOL- presionado
Volumen: 80% -> 70%
^C
```

*Illustration 9. Server’s full functionality (all actions executed)*

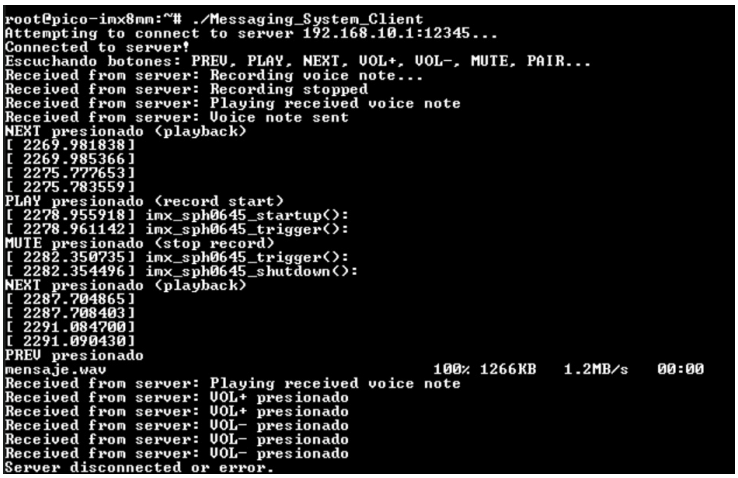



*Illustration 10. Client's full functionality (all actions executed)*

### Link to video of final version, fully functional:

*Lab1_ALSA_FunctionalityShowCase.mp4*

## Conclusions

This laboratory definitely helped us to get more familiar not only with ALSA, but also with the embedded board (the PICO-PI-IMX8M), the process orchestration of shell scripts, multithreading, and general programming on the terminal as well. We could say that the first part of the Lab development consisted of creating or gathering parts of submodules that would be useful to us, while the second part (once we had a base application) was more of a trial-and-error methodology to make the system fully functional.

Something that got us to keep going (or to keep on working on the lab) despite not having the necessary scripts to correctly capture and play audio yet, was to still implement the multithread and try just to type in some .txt and read from it as well. Things like that made us learn or realize that there is always a way to advance in the development of a project or an application, despite having one or several blocker of any sort.

## References

[1] Spiros, I. (n.d.). *What is Advanced Linux Sound Architecture – MVPS.net Blog*. https://www.mvps.net/docs/what-is-advanced-linux-sound-architecture/#:~:text=ALSA%20manages%20audio%20applications%20and,distributions%20due%20to%20its%20flexibility.

[2] JumpCloud. (2025, May 21). *What is a Socket? - JumpCloud*. https://jumpcloud.com/it-index/what-is-a-socket#:~:text=A%20socket%20serves%20as%20an,reliable%2C%20connection%2Doriented%20communication.

[3] *TCP: How the Transmission Control Protocol works*. (2020, March 2). IONOS Digital Guide. https://www.ionos.com/digitalguide/server/know-how/introduction-to-tcp/#:~:text=and%20network%20services.-,How%20exactly%20do%20TCP%20connections%20work?,socket%22)%20for%20each%20endpoint.

[4] *ByteByteGo | Process vs Thread: Key Differences*. (n.d.). ByteByteGo. https://bytebytego.com/guides/what-is-the-difference-between-process-and-thread/#:~:text=The%20program%20contains%20a%20set,communication%20is%20faster%20for%20threads.

[5] *Operating systems: threads*. (n.d.). https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html#:~:text=References:,of%20the%20file%20being%20edited.