SISTEMA DE MONITOREO REMOTO DE CONTROL DE ACCESO POR CLAVE

Germán Alejandro, Tello
e-mail: german.tello.93@gmail.com
Alan Alexis, Velazquez
e-mail: alanvalexis@gmail.com

ÍNDICE

ABSTRACT: *Este trabajo presenta el desarrollo de un sistema de monitoreo remoto para el proyecto diseñado durante la primera etapa del año en la cátedra de Técnicas Digitales III. El proyecto consiste en un sistema de control de acceso mediante una clave de 4 dígitos, integrado con una Raspberry Pi 4B que ejecuta Raspberry Pi OS, un sistema operativo de propósito general. El sistema propuesto se divide en dos fases principales de desarrollo. La primera fase comprende un módulo en kernel space encargado de gestionar el hardware de la Raspberry Pi, específicamente el bus I2C, para facilitar la comunicación con el microcontrolador STM32F103 que opera el sistema de control de acceso. La segunda fase consiste en una aplicación en user space que proporciona una interfaz para que el usuario interactúe con el sistema, utilizando el módulo en kernel space para las comunicaciones con el hardware. Adicionalmente, el sistema ampliado incorpora capacidades de acceso remoto mediante SSH y Ethernet, lo que mejora el proyecto original al permitir el monitoreo y operación remotos del sistema de control de acceso.*

PALABRAS CLAVE: Linux, I2C, Raspberry Pi, monitoreo remoto vía SSH, módulo de Kernel, Kernel Space, User Space, Shell.

1 INTRODUCCIÓN

En la parte A se desarrolló un sistema capaz de permitir accesos a usuarios autorizados mediante una clave de 4 dígitos, el mismo cuenta con un procesador Cortex M3, montado en un microcontrolador STM32F103, en la placa de desarrollo Bluepill, además para el software se utiliza un FreeRTOS. Sin embargo, la problemática planteada en la parte B de la cátedra Técnicas Digitales III, se suma el desafío de poder remotizar nuestro sistema, y darle la posibilidad de operarlo y obtener lectura de parámetros de funcionamiento de manera remota.

Por lo tanto, para el desarrollo de la parte B de nuestro proyecto, vamos a implementar una solución utilizando una Raspberry Pi 4B, la cual tendrá embebido un sistema operativo de propósito general (Raspberry PI OS), y por medio del bus I2C, nos conectaremos al proyecto desarrollado en la parte A. Para esto, necesitamos desarrollar dos módulos programados en C, uno que estará corriendo en Kernel Space y será el encargado de administrar el hardware I2C de la Raspberry Pi y otro módulo que correrá en User Space y será el encargado de recibir las órdenes del usuario y trasmitirlas al módulo kernel por medio de file system de sistema operativo.

A través de este desarrollo, se busca optimizar el rendimiento del dispositivo y facilitar su integración en entornos más amplios, como sistemas de automatización, monitoreo o control industrial.

2 OBJETIVOS

Desarrollar un módulo driver de Linux y una interfaz de usuario para controlar y monitorear el dispositivo desarrollado en la Evaluación Globalizadora A, adquiriendo la habilidad de desarrollar sistemas en entornos híbridos con múltiples subsistemas gobernados por un sistema central que cuente con un sistema operativo de propósito general, dándonos la posibilidad de mayor potencia de procesamiento y comunicación remota.

3 ALCANCE

Nuestro propósito general es poder replicar la funcionalidad del sistema de control de acceso elaborado en la EGA pero, en esta ocasión, mediante la operación del mismo de manera remota desde una interfaz de usuario en el GPOS montado en la Raspberry Pi.

Las funcionalidades a replicar son:

1. Acceso por clave numérica de 4 dígitos como usuario general.
2. Acceso por clave numérico con el usuario maestro.
3. Borrado y asignación de usuarios y contraseñas.
4. Control mediando contraseña “Maestra”.
5. Estado actual de puerta y alarma.
6. Activar y desactivar alarma.

Además, se pretende monitorear el estado del acceso (estado de la puerta y la alarma) y generar un LOG de eventos importantes. Mediante dicho LOG de eventos, podremos saber el estado de la puerta y alarma.

4 HARDWARE

4.1 DESCRIPCIÓN DE HARDWARE

La base en la cual se desarrollará esta parte del proyecto es una Raspbrerry Pi (se ilustra en la figura 1), particularmente en este caso utilizaremos la versión 4 modelo B+.

Figura 1. Raspberry Pi

La Raspberry Pi 4 B cuenta con un procesador quad-core de 64 bits, soporte para doble pantalla con resoluciones de hasta 4K mediante dos puertos micro HDMI, decodificación de video por hardware con resolución hasta 4Kp60, hasta 8 GB de RAM, conexión inalámbrica Wi-Fi de doble banda (2.4/5.0 GHz), Bluetooth 5.0, Ethernet Gigabit, USB 3.0 y capacidad de alimentación por Ethernet (PoE) mediante un accesorio adicional PoE HAT. En síntesis, ofrece un rendimiento comparable al de una PC básica con procesador de arquitectura x86.

Además, posee 26 pines de propósito general (GPIO), en los cuales se pueden implementar protocolos de comunicación tales como I2C, UART, o SPI.

La raspbrerry implementa un sistema operativo de propósito general basado en Linux, específicamente de la distribución Debian, llamado Raspbrerry Pi OS (anteriormente llamado Raspbian). Particularmente en este caso utilizaremos un Raspberry Pi OS basado en Debian GNU/Linux 12, con una versión de kernel de Linux 6.6.51+rpt-rpi-v8.

Como se mencionó, la Raspberry posee un puerto de comunicación I2C que es el que utilizaremos en este proyecto, el mismo se encuentra en los GPIO 2 (pin 3) y GPIO3 (pin 5), tal como se ilustra en la figura 2.

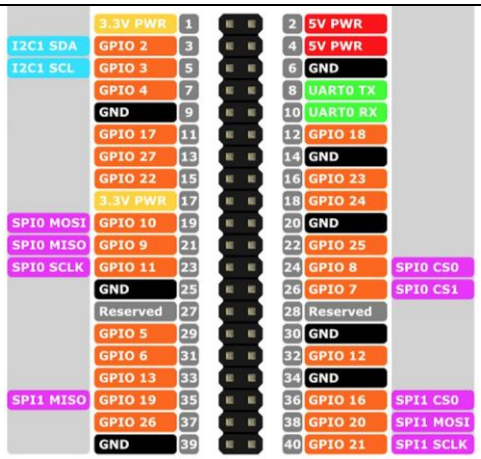


Figura 2. Pinout de la Raspberry Pi

4.2 DIAGRAMA DE INTERCONEXIÓN

El diagrama que se muestra en la figura 3, es el diagrama de interconexión con detalles de pines.

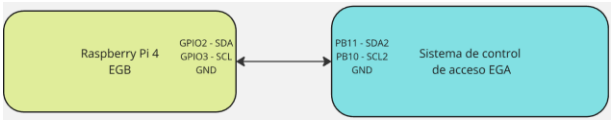


Figura 3. Detalles de pines.

5 MODIFICACIONES A LA PARTE A

Para poder realizar la correcta implementación de la comunicación entre el dispositivo desarrollado en la parte A y la raspbrerry pi, es necesario realizar modificaciones al código original.

5.1 CONEXIONADO CON LA RASPBERRY PI

En el desarrollo de la parte A se tomó como previsión dejar disponibles los pines de un puerto I2c. Para esto, se utilizará el puerto 2 del I2C, correspondiente a los GPIO PB10 (pin26) y PB11 (pin25). La conexión se aprecia el diagrama en bloques completo del anexo I y en la figura 3.

El mismo se configuró a una velocidad de transmisión de 100Kbits/s y una dirección de Slave de 0x15, además de utilizar las interrupciones NVIC para eventos y errores, a un nivel de prioridad de 8 para que no entre en conflicto con el kernel de RTOS.

5.2 MODIFICACIONE DE SOFTWARE

La comunicación I2C requiere que uno de los dispositivos opere en modo maestro, solicitando el envío o recepción de información, y el otro dispositivo en modo esclavo, enviando y recibiendo la información que el dispositivo maestro solicita. Para este caso, la Bluepill trabajará en modo esclavo, siendo la Raspberry Pi la que solicite la comunicación. En la Bluepill existirán dos buffers de comunicación, uno de RX de entrada, donde se almacena los datos provenientes de la Raspberry Pi, y uno de TX de salida, que la Raspberry Pi leerá al solicitar información.

Al código original se le agrega una tarea extra. Esta tarea se encargará de leer las colas mediante las cuales las otras tareas intercambian información, para recolectar dicha información en el buffer de salida de una manera apropiada y definida. Además, leerá el buffer RX de entrada, cargará la información en las colas pertinentes para poder ejecutar los comandos externos solicitados desde la Raspberry Pi.

En la figura 4, se aprecia el diagrama de tareas modificado con la nueva tarea controladora para la comunicación I2C. En el anexo II se puede apreciar este mismo diagrama con más detalles.

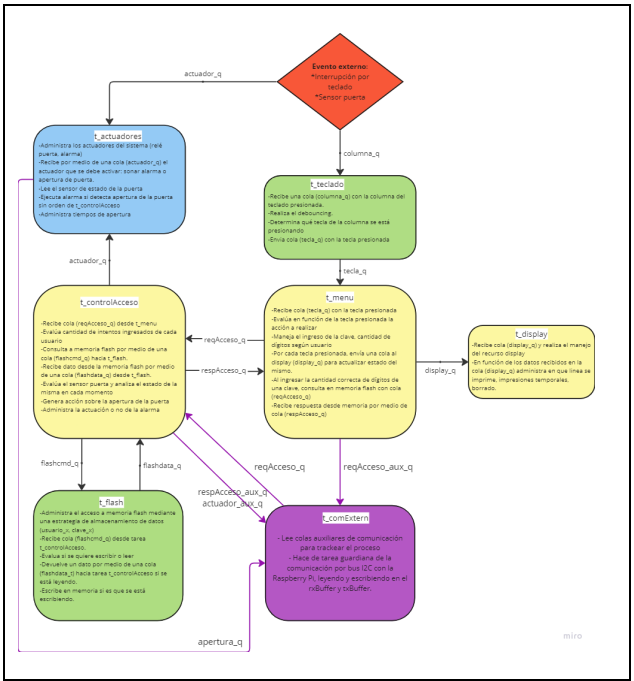


Figura 4. Diagrama de tareas

5.3 ESTRUCTURA DE TRAMA I2C

Los datos en el bus I2C se organizan de la siguiente manera (figura 5). Los mismos son transformados de estructuras internas de funcionamiento del programa a bytes seriados de manera tal que puedan ser transmitidos por el bus y luego son interpretados por el programa User en la Raspberry Pi, de la misma manera que las órdenes enviadas desde la Raspberry Pi son entramadas de manera serial para ser recibidas por la Bluepill, para luego ser reordenadas en estructuras. Para mayor detalle referirse al Anexo IV.

GLOBAL ID (00, 01, 02...)	INDICADOR (p, r)	USUARIO (0, 1, 2...)	PASSWORD (8 DIGITOS)	NRO-INTENTOS (1 DIGITO)	ACCION (2 DIGITOS)	TERMINADOR ($)
GLOBAL ID (00, 01, 02...)	INDICADOR (a)	TIPO DE ACTUADOR (0, 1, 2)	BITS RELLENO (8 DIGITOS)	BIT RELLENO (1 DIGITO)	ACCION (2 DIGITOS)	TERMINADOR ($)
GLOBAL ID (00)	INDICADOR (l)	BIT RELLENO (1 BIT)	BITS RELLENO (8 DIGITOS)	ACCION (SCANN / READ)	STATUS (2 DIGITOS) (CANTIDAD DE LOGS)	TERMINADOR ($)
GLOBAL ID (00, 01, 02...)	INDICADOR (d)	BIT RELLENO (1 BIT)	BITS RELLENO (8 DIGITOS)	ESTADO (CERRADA /ABIERT)	ACCION (2 DIGITOS) (ACC_OK, ACC_DENEG)	TERMINADOR ($)

6 SOFTWARE

6.1 CONCEPTOS BASICOS DE LINUX

Linux es un sistema operativo de código abierto basado en el kernel desarrollado por Linus Torvalds en 1991. Es conocido por su estabilidad, seguridad y flexibilidad, lo que lo hace ideal para una amplia variedad de dispositivos, desde servidores y PCs hasta sistemas embebidos.

El sistema Linux puede dividirse a grandes rasgos en dos partes, el Kernel o núcleo y el espacio de usuario.

El kernel es el encargado de gestionar los recursos del sistema, como la CPU, la memoria y los periféricos, permitiendo que estos sean utilizados de manera eficiente. Además, el kernel se encarga de coordinar la ejecución de procesos, administrar el acceso a la memoria, organizar los datos en los sistemas de archivos y garantizar la seguridad mediante el control de permisos y accesos. También es el encargado de actuar como intermediario entre el hardware y las aplicaciones mediante la implementación de controladores o *drivers*. Es de diseño modular, esto significa que pueden agregársele o modificársele funcionalidades conforme sean necesarias, lo que lo hace adaptable a una amplia variedad de dispositivos.

El espacio de usuario es la parte del sistema operativo donde se ejecutan las aplicaciones y procesos de los usuarios. Es independiente del kernel, lo que significa que las aplicaciones en este espacio no interactúan directamente con el hardware ni tienen acceso directo a los recursos del sistema, sino que se comunican con el kernel a través de llamadas al sistema (system calls). Este diseño garantiza la estabilidad y seguridad del sistema, ya que un error en una aplicación en el espacio de usuario no afecta directamente al kernel ni a otros procesos críticos del sistema.

En el diagrama de la figura 4 se muestra una representación de los bloques que conforman el SO

Linux y su interacción entre ellos para implementar las soluciones propuestas en el presente trabajo. El diagrama de la figura 3 puede apreciarse con mayor detalle en el anexo III.

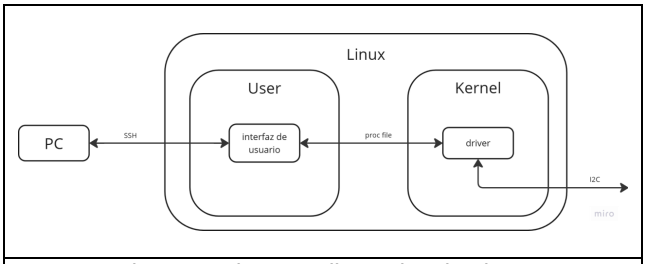


Figura 5. Diagrama ilustrativo de Linux

6.2 DRIVER

Un driver en Linux es un módulo de software que actúa como una interfaz entre el kernel y un dispositivo de hardware específico. Su función principal es traducir las instrucciones del sistema operativo en comandos que el hardware pueda entender, y viceversa, permitiendo que el dispositivo sea reconocido y utilizado por el sistema. Los drivers suelen ejecutarse en el kernel. Pueden integrarse directamente en el kernel o cargarse dinámicamente como módulos según sea necesario.

Desde el espacio de usuario, los drivers se ven como archivos, y es mediante la manipulación de estos archivos que las aplicaciones a nivel usuario logran interactuar con los distintos elementos del hardware

Para que un driver pueda funcionar apropiadamente, el hardware que este pretende manejar debe estar definido en el device tree de Linux. El device tree es una estructura de datos utilizada para describir el hardware de una determinada plataforma. Su propósito principal es proporcionar al kernel una forma estandarizada de identificar y configurar los dispositivos presentes en el sistema. Incluye información como direcciones de memoria, IRQs, buses (como I2C, SPI), relojes, GPIOs y otros recursos de hardware.

Dentro del driver desarrollado existen 3 funciones que resultan las más importantes. Primero i2c_prove que se ejecuta al momento de insertar el driver en el kernel de Linux y es la encargada de crear el archivo que utilizara el programa de usuario para comunicarse mediante I2C. Luego están las funciones i2c_read e i2c_write, que se ejecutan al leer y escribir el archivo del driver, mediante estas funciones se ejecuta la transferencia de datos desde el buffer I2C en el espacio de kernel hacia el espacio de usuario y viceversa.

6.3 INTERFAZ DE USUARIO

Se desarrolló un programa sencillo que oficia de interfaz de usuario para la operación y control del dispositivo. El mismo ofrece las mismas funcionalidades que se presentan en el dispositivo físico, tal como la opción de ingresar con una contraseña, o la opción de cargar nuevos usuarios o eliminar existentes. También se agregó la funcionalidad de visualizar todos los intentos de ingresos anteriores mediante un log de eventos.

La interfaz de usuario se implementa mediante un menú de opciones sencillo en la consola de comando de Linux. Para acceder inicialmente al programa es necesario ingresar un usuario y contraseña válidos. La validez de los mismos se constata siempre en el dispositivo de acceso, no en la Raspberry Pi.

7 PORCENTAJE DE ALCANCE

De los objetivos planteados en la sección 3, se han logrado casi en su totalidad. Los objetivos 1, 2 ,3 y 4 se cumplieron satisfactoriamente, el objetivo 6 se cumplió parcialmente, faltando culminar un detalle en la implementación para que la misma sea más directa, y el objetivo 5 quedó inconcluso.

8 CONCLUSIONES

Se han logrado implementar con éxito la mayoría de los objetivos planteados inicialmente. Este nuevo desarrollo amplía las capacidades del sistema original al proporcionar una interfaz para la gestión y monitoreo, aprovechando la flexibilidad del entorno Linux.

El programa permite realizar las tareas del sistema original como el ingreso con clave y la administración de usuarios, agregando la supervisión de eventos de acceso, lo que mejora la funcionalidad.

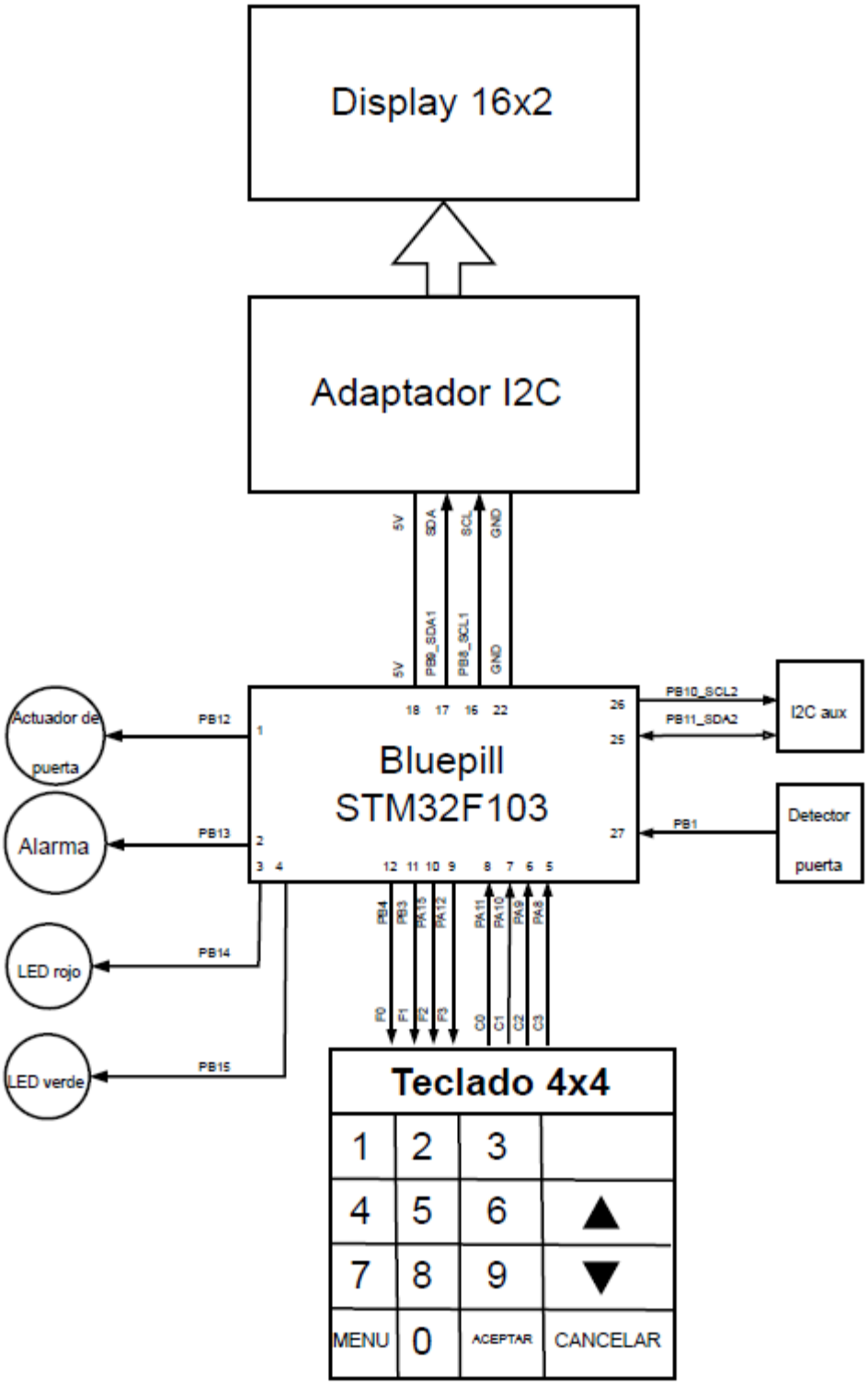
Este desarrollo permitirá la escalabilidad para aplicaciones más complejas en el futuro. Entre las posibles mejoras futuras se encuentran la implementación de interfaces gráficas más intuitivas o la integración con redes para monitoreo remoto.

9 REFERENCIAS

[1] Raspbery Pi https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

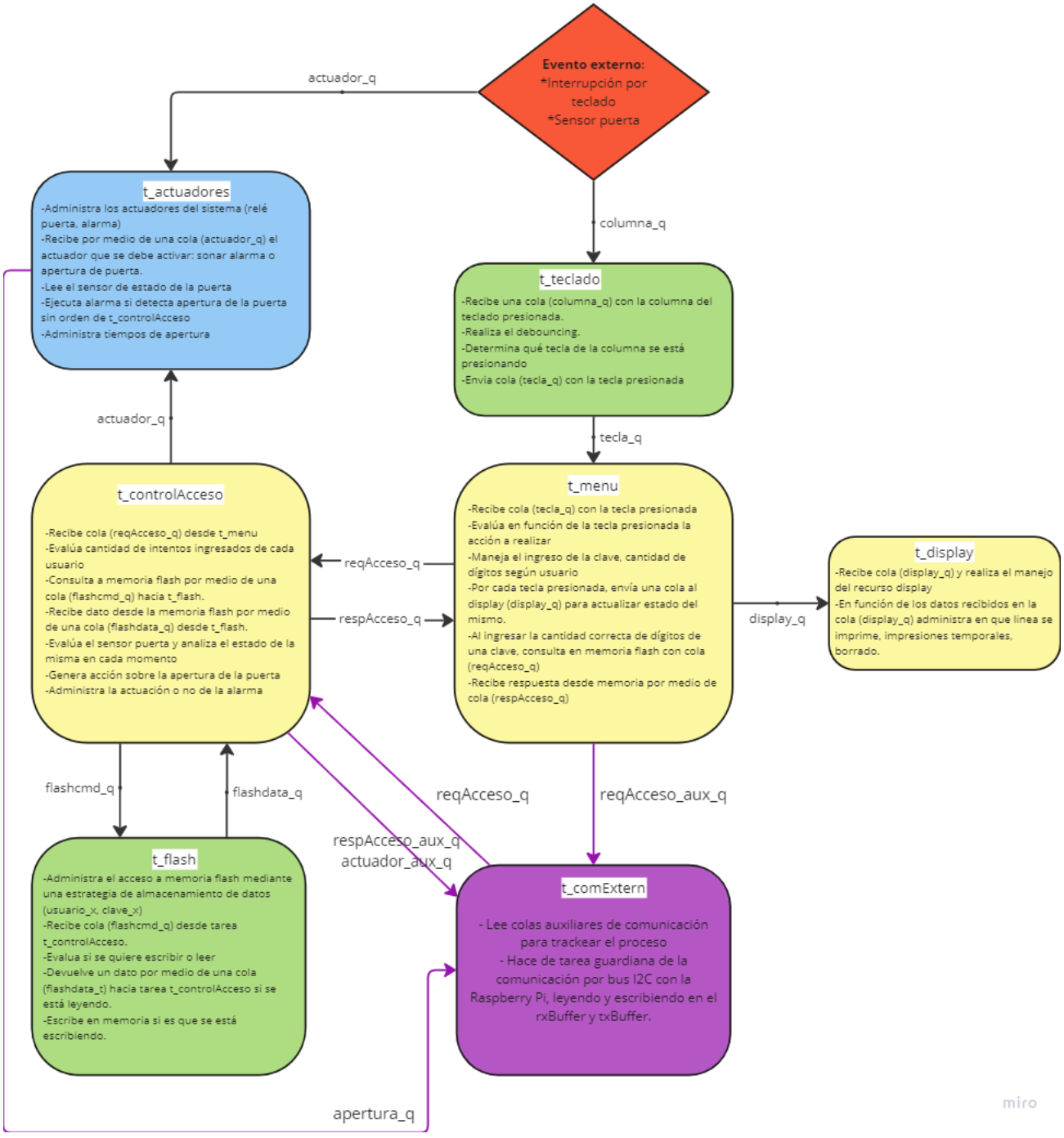
[2] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, *“LINUX Device Drivers”*, Third edition, 2005.

10 ANEXO I: DIAGRAMA EN BLOQUES

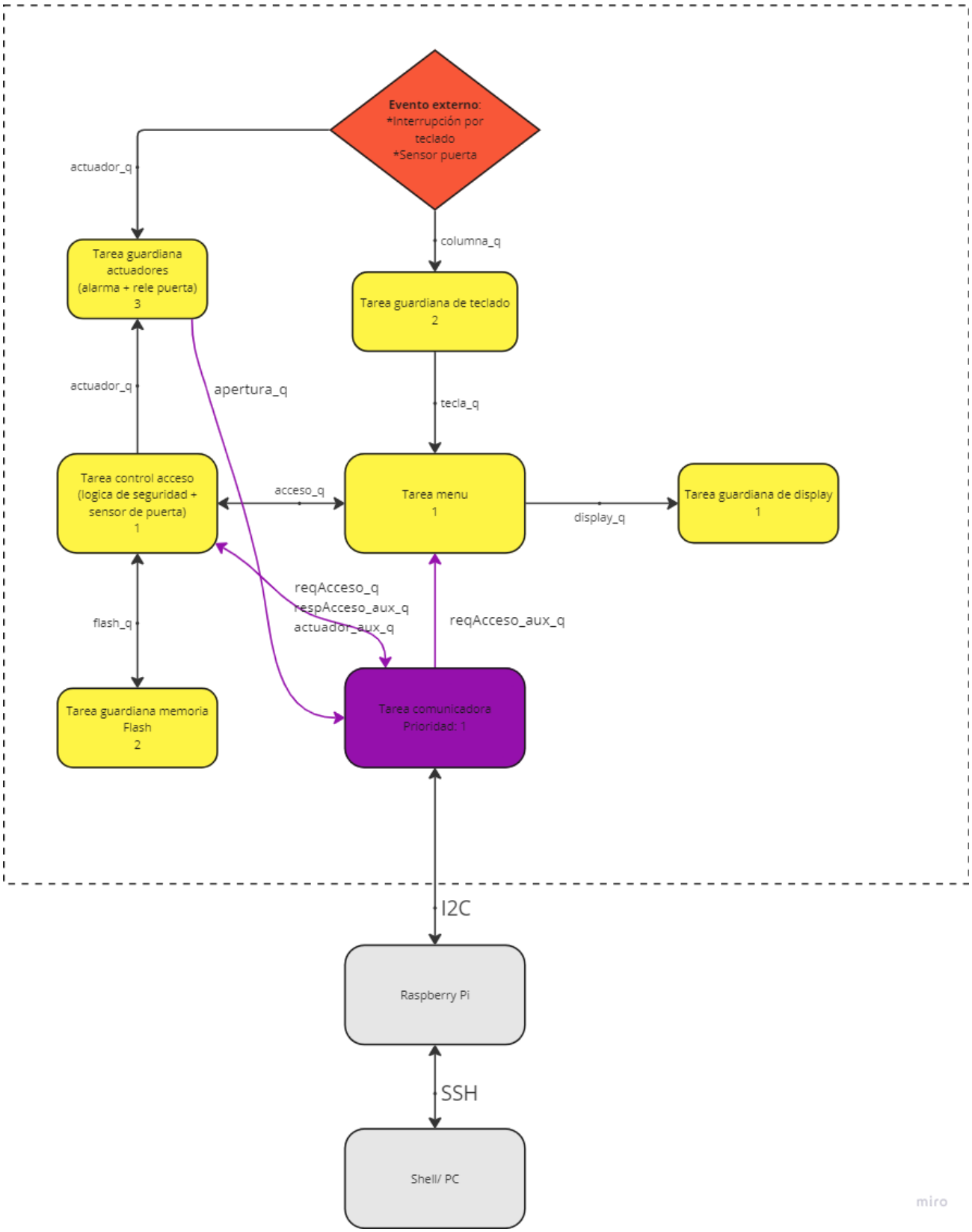


11 ANEXO II: DIAGRAMA DE TAREAS MODIFICADO

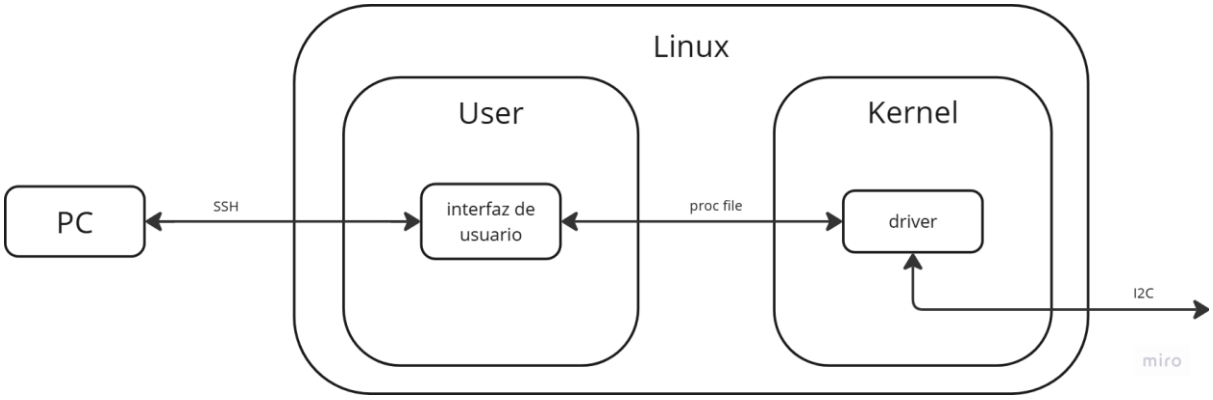
11.1 DIAGRAMA DE TAREAS EXTENDIDO



11.2 DIAGRAMA DE TAREAS SIMPLIFICADO



12 ANEXO III: DIAGRAMA EN BLOQUES DE SOFTWARE



13 ANEXO IV: ORGANIZACIÓN DE DATOS DE MANERA SERIE

GLOBAL ID (00, 01, 02...)	INDICADOR (p, r)	USUARIO (0, 1, 2...)	PASSWORD (8 DIGITOS)	NRO-INTENTOS (1 DIGITO)	ACCION (2 DIGITOS)	TERMINADOR ($)
GLOBAL ID (00, 01, 02...)	INDICADOR (a)	TIPO DE ACTUADOR (0, 1, 2)	BITS RELLENO (8 DIGITOS)	BIT RELLENO (1 DIGITO)	ACCION (2 DIGITOS)	TERMINADOR ($)
GLOBAL ID (00)	INDICADOR (l)	BIT RELLENO (1 BIT)	BITS RELLENO (8 DIGITOS)	ACCION (SCANN / READ)	STATUS (2 DIGITOS) (CANTIDAD DE LOGS)	TERMINADOR ($)
GLOBAL ID (00, 01, 02...)	INDICADOR (d)	BIT RELLENO (1 BIT)	BITS RELLENO (8 DIGITOS)	ESTADO (CERRADA /ABIERT)	ACCION (2 DIGITOS) (ACC_OK, ACC_DENEG)	TERMINADOR ($)