PBE

PUZZLE 1

1. Conexiones Raspberry - RC522:

MFRC522 Pin	Raspberry Pi Pin	Descripción
SDA (SS)	GPIO 8 (Pin 24)	Pin de selección de esclavo (Slave Select). Indica al MFRC522 que el maestro (Raspberry Pi) desea comunicarse.
SCK	GPIO 11 (Pin 23)	Reloj de la comunicación SPI. Sincroniza la transferencia de datos.
MOSI	GPIO 10 (Pin 19)	"Master Out Slave In". Este pin lleva datos desde la Raspberry Pi al MFRC522.
MISO	GPIO 9 (Pin 21)	"Master In Slave Out". Este pin lleva datos desde el MFRC522 a la Raspberry Pi.
IRQ	No conectado	Interrupción. Este pin se utiliza para notificar al maestro sobre eventos específicos. En muchos casos, no es necesario conectarlo.
GND	GND (Pin 6)	Tierra. Conecta el pin GND del MFRC522 a la tierra de la Raspberry Pi para un circuito común.
RST	GPIO 25 (Pin 22)	Reset. Este pin se utiliza para reiniciar el MFRC522. Puede ser conectado a un GPIO para controlarlo si es necesario.
VCC	3.3V (Pin 1)	Alimentación. Conecta el pin VCC del MFRC522 a la salida de 3.3V de la Raspberry Pi para alimentarlo.

- **SDA (SS)**: este pin se activa (se lleva a nivel bajo), el MFRC522 se prepara para recibir datos
-
- **SCK:** Cada vez que hay un pulso en este pin, se puede enviar o recibir un bit de datos.
-
- **MOSI**: Cuando se envían datos, la Raspberry Pi coloca los bits en este pin en el momento apropiado según la señal del pin SCK.

- **MISO:** Cuando el MFRC522 tiene datos listos para enviar, los coloca en este pin mientras la Raspberry Pi controla el reloj mediante SCK.

- **IRQ:** Este pin es opcional y se utiliza para notificar al maestro que ha ocurrido un evento (como la lectura de una tarjeta).

- **RST:** Este pin se utiliza para reiniciar el MFRC522.

2. Conexiones Raspberry-PC

Al principio hubo complicaciones con este tema, al no poder conectar la Raspberry a un monitor, teclado y ratón independientes para operar directamente en ella.

La primera opción fue optar por **conexión vía SSH** y aunque también surgió algún que otro problema, finalmente fue posible conectarla.

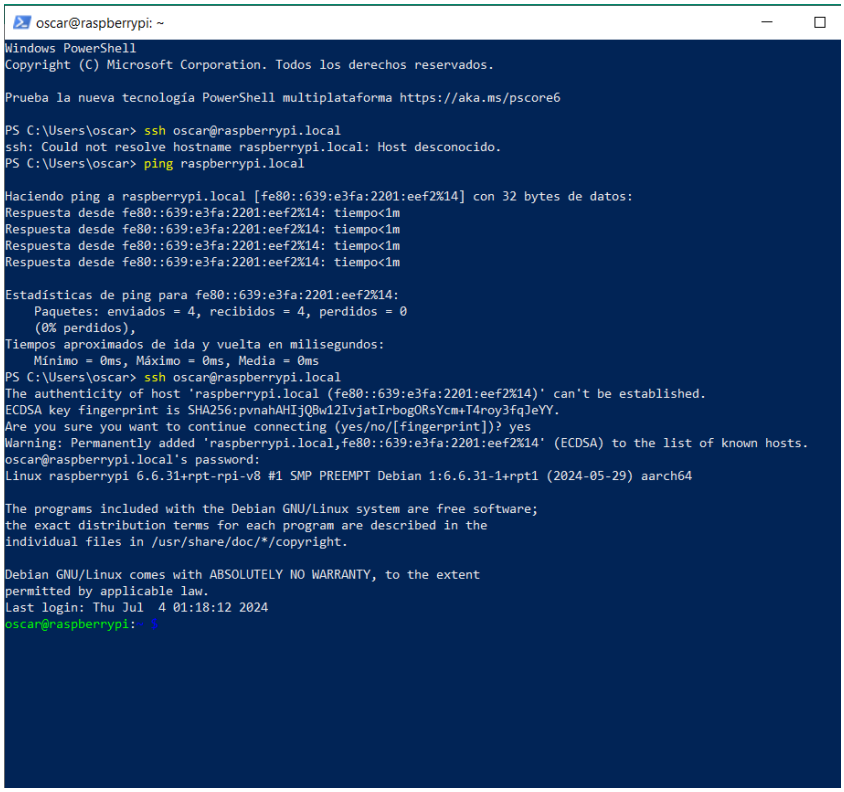
Al no disponer de la dirección IP de la Raspberry, decidí usar el hostname de la Raspberry: **raspberry.local,** que también nos permití la conexión.

Probé primero en mi máquina virtual de Ubuntu, sin embargo, al parecer era imposible activar la conexión. Nunca encontraba al host, aun estando las configuraciones correctas. Probando en otra máquina virtual, con la misma configuración, de algún compañero sí se conseguía exitosamente la conexión.

Ante este problema, decidí acceder directamente desde Windows usando PowerShell y finalmente pude realizar la conexión.

Nota: En cualquiera de estas opciones, la Raspberry tiene que estar alimentada y conectada al dispositivo a través de un cable Ethernet para darle internet.

Aquí observamos la ejecución de la conexión:



Explicación de los comandos

- La línea de comando **ssh oscar@raspberrypi.local** se utiliza para establecer una conexión SSH con un dispositivo que tiene la dirección raspberrypi.local, donde oscar es el nombre de usuario al que nos conectamos

  El hostname raspberrypi.local es una forma común de referirse a la Raspberry Pi en la red local cuando el dispositivo tiene habilitado el servicio mDNS (Multicast DNS). Esto permite que los dispositivos de la red se identifiquen por nombre en lugar de tener que recordar direcciones IP.

Si la Raspberry Pi está en la misma red local y está configurada para aceptar conexiones SSH, deberías poder acceder a su terminal de línea de comandos.

Finalmente, pide la contraseña asociada con el usuario oscar en la Raspberry Pi. Una vez ingresada correctamente, tenemos acceso a la terminal de la Raspberry Pi, lo que te permitirá ejecutar comandos, administrar archivos y realizar configuraciones.

Nota: Como podemos observar al principio no encuentra el hostname. Es normal porque, a veces falla, por lo que ejecutamos el siguiente comando como vemos en la imagen.

- El comando ping raspberrypi.local verificar la conectividad de red y asegura que la Raspberry Pi esté accesible desde tu computadora. Si recibimos respuesta, significa que la Raspberry Pi está encendida y conectada a la red, lo que te permitirá continuar con otras interacciones, como establecer una conexión SSH

  Comprobamos que sí que está conectada, repetimos el primer comando y observamos que nos hemos conectado correctamente

3. Instalación Ruby y bibliotecas probadas RFID RC522:

Primero, actualizamos nuestra RaspberryPi: **sudo apt-get update**

Para poder comunicar la Raspberry con el RFID-RC522 necesitamos activar el puerto SPI usando: **sudo rasp-config**

Iremos a “Interfacing Options” y seleccionaremos SPI para habilitar la interfaz SPI. Hay que reiniciar la RaspberryPi: **sudo reboot**

Instalamos Ruby y algunas librerías que te permitirán interactuar con el RC522:

- Instalar Ruby: **sudo apt-get install ruby-full**

Para trabajar con un lector RFID MFRC522 en una Raspberry Pi utilizando Ruby, necesitamos instalar varias librerías:

- **rpi_gpio:** Esta librería permite controlar los pines GPIO de la Raspberry Pi desde Ruby. Necesaria para gestionar la comunicación con el lector RFID.

- **colorize**: Se utiliza para agregar color al texto en la terminal, lo que mejora la legibilidad de las salidas del script.

- **mfrc522:** Esta librería proporciona una interfaz para interactuar con el lector RFID MFRC522. Es fundamental para leer el UID de las tarjetas RFID.

Nota: Finalmente se ve como no se utilizan todas, solo la última

Motivo de estas librerías

No he tenido que investigar mucho puesto que son las que mas recomiendan para trabajar con este periférico. Las dos tres librerías son gemas de Ruby, que son paquetes de software que se pueden instalar fácilmente desde el repositorio oficial de gemas de Ruby, conocido como RubyGems: **sudo gem install pi_piper, sudo gem install** y **sudo gem install colorize**

```
oscar@raspberrypi:~ $ sudo gem install rpi_gpio
Fetching rpi_gpio-0.5.0.gem
Fetching epoll-0.3.0.gem
Building native extensions. This could take a while...
Successfully installed epoll-0.3.0
Building native extensions. This could take a while...
Successfully installed rpi_gpio-0.5.0
Parsing documentation for epoll-0.3.0
Installing ri documentation for epoll-0.3.0
Parsing documentation for rpi_gpio-0.5.0
Installing ri documentation for rpi_gpio-0.5.0
Done installing documentation for epoll, rpi_gpio after 0 seconds
2 gems installed
```

```
oscar@raspberrypi:~ $ sudo gem install colorize
Fetching colorize-1.1.0.gem
Successfully installed colorize-1.1.0
Parsing documentation for colorize-1.1.0
Installing ri documentation for colorize-1.1.0
Done installing documentation for colorize after 0 seconds
1 gem installed
```

```
oscar@raspberrypi:~ $ sudo gem install mfrc522
Fetching mfrc522-3.0.0.gem
Fetching fubuki-1.0.1.gem
Building native extensions. This could take a while...
Successfully installed fubuki-1.0.1
Successfully installed mfrc522-3.0.0
Parsing documentation for fubuki-1.0.1
Installing ri documentation for fubuki-1.0.1
Parsing documentation for mfrc522-3.0.0
Installing ri documentation for mfrc522-3.0.0
Done installing documentation for fubuki, mfrc522 after 1 seconds
2 gems installed
```

4. Script Ruby para implementación Puzzle1:

Para ejecutar el código, clone el repositorio de GitHub con el código. Para ello tuve que instalar Github en la Raspberry con el comando: **sudo apt install git -y**

Clonamos el repositorio: git clone https://github.com/oscarparada08/PBE.git

Ahora tmb lo tenemos en nuestra Raspberry. Navegamos en el con el comando cd PBE

Y ejecutamos el código: ruby PUZZLE1.rb

```ruby
require 'bundler/setup'
require 'mfrc522'

# Inicializa el lector
r = MFRC522.new

begin
  # Solicita la tarjeta
  r.picc_request(MFRC522::PICC_REQA)

  # Selecciona la tarjeta y obtiene el UID
  uid, sak = r.picc_select

  # Imprime el UID
  puts "UID de la tarjeta: #{uid}"
rescue CommunicationError => e
  # Manejo de errores en la comunicación
  abort "Error comunicando con la tarjeta: #{e.message}"
end
```

Al comienzo de mi código, incluyo las librerías necesarias para que funcione correctamente. Utilizo require 'bundler/setup' para asegurarme de que todas las gemas que necesito estén cargadas adecuadamente, lo que es esencial para que el entorno trabajo esté listo.

Luego, importo la biblioteca mfrc522 con require 'mfrc522', que es fundamental para interactuar con mi lector MFRC522.

Después, inicializo el lector con r = MFRC522.new. Con esta línea, creo una instancia del lector, lo que me permite comenzar a interactuar con el hardware y prepararlo para la comunicación con tarjetas RFID.

A continuación, entro en un bloque begin, que me permite manejar cualquier error que pueda ocurrir durante la ejecución del código.

Dentro de este bloque, le pido al lector que busque tarjetas RFID en su proximidad con r.picc_request(MFRC522::PICC_REQA).

Esta línea envía una solicitud específica para que el lector detecte tarjetas. Si no hay tarjetas presentes, puedo encontrarme con un error que se manejará más adelante.

Una vez que el lector ha detectado una tarjeta, selecciono la tarjeta que he encontrado y obtengo su UID (Identificador Único) con uid, sak = r.picc_select.

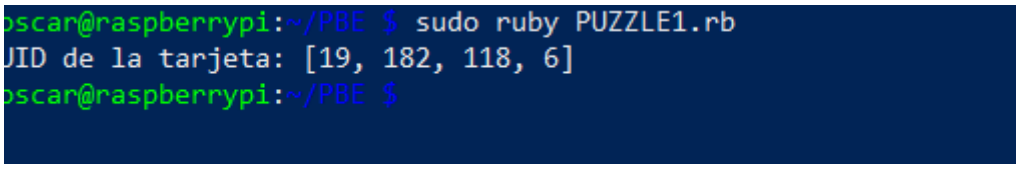
Aquí, selecciono la tarjeta que ha respondido a mi solicitud y obtengo su UID, que es un identificador único para cada tarjeta.

Luego, imprimo el UID en la consola utilizando puts "UID de la tarjeta: #{uid}".

Esta línea es crucial porque me permite ver el resultado de mi interacción con la tarjeta RFID.

Finalmente, manejo cualquier posible error de comunicación que pueda ocurrir. Si algo sale mal en el bloque begin, me muevo al bloque rescue, donde capturo errores relacionados con la comunicación con rescue CommunicationError => e. Si se produce un error, imprimo un mensaje de error en la consola y finalizo el programa con abort "Error comunicando con la tarjeta: #{e.message}". Esto es importante para saber qué salió mal.

En resumen, este código se encarga de interactuar con un lector RFID. Primero, busca tarjetas en la proximidad, luego selecciona una de ellas y obtiene su UID, que imprime en la consola. Si hay algún problema durante la comunicación, se maneja el error de forma adecuada.



Al final se consiguió el objetivo de que fuera capaz de leer la tarjeta y sacar por consola su valor. Sin embargo, aun no conseguí que fuera de forma cíclica. Además, cuando ejecutas el código la tarjeta tiene que estar en contacto ya con el RFID y me gustaría poder poner un tiempo de espera para si se llega un poco tarde aun pueda escanear la tarjeta sin volver a ejecutar el código

Problemas encontrados

Aun conectando la Raspberry por SSH, no recibía conexión a internet. Este problema fue difícil de solucionar, puesto que a priori, se estaban realizando los comandos correctos. Probé con otros tipos de conexión y persistía el error.

Al final el error se encontraba en el propio portátil. La configuración de serie no permitía compartir datos de conexión y tenía inhabilitado el uso compartido en el puerto Ethernet. Una vez visto el problema fue fácil ponerle solución

Al principio quise usar la librería rpi_gpio.so para controlar los pines, pero no pude ya que está compilado para una arquitectura de 32 bits (ELFCLASS32), mientras que el sistema es de 64 bits. Así que pase a usar la librería pi_piper

En el proceso hubo que hacer un montón de configuraciones, descargas de archivos, actualizaciones. Sobre todo, a la hora de descargar rpi_gpio y mfrc522. El bunder también dio muchos problemas. Tuve que volver a descargarlo usando un comando diferente al que usaba, más específico. Fue necesario para el pi_piper (en los problemas se hablara de esta librería) descargar una librería llamada eventmatcher y justamente tenía que ser la versión 1.0.9.

Al final, la librería pi_piper comenzó a dar problema por temas de inicialización y me di cuenta que no era necesaria. Con la librería mrfc522 era suficiente para desarrollar el puzzle1 (lectura tarjeta).