Óscar Parada Fernández

PBE

PUZZLE 2

1. Actualizaciones Puzzle 1

Se han realizado las modificaciones solicitadas en la revisión para que el código fuera mucho mejor:

- clase Rfid
- Constructor
- bucle de polling al ser una biblioteca no bloqueante

A su vez se reestructuró de forma un poco diferente y se realizaron cambios.

```
require 'mfrc522'
class Rfid
        @@r = MFRC522.new
        def read_uid
                quedat = 1;
                puts "Si us plau, introdueixi la seva targeta sobre el lector"
                $stdout.flush
                while(quedat==1)
                        begin
                                @@r.picc_request(MFRC522::PICC_REQA) #Establiment de perifèric
                                uid_dec, que = @@r.picc_select          #Intent de lectura (pot llençar CommunicationError degut a timeout)
                        rescue CommunicationError => e            #Capturem raising (si entra aquí s'està assolint timeout)...
                        else                                      #Ha capturat uid. Sortim
                                quedat = 0
                        end
                end
                uid = Array.new                                   #Vector hexadecimal
                uid_dec.length.times do |i|
                        uid[i]=uid_dec[i].to_s(16)
                end

                return uid.join().upcase                          #Retornem uid capturat en forma de string concatenat, en majúscules
        end
end

if __FILE__ == $0
        rf = Rfid.new()
        uid = rf.read_uid
        puts uid
end
```

- La **instancia del lector MFRC522** se inicializa como una variable de tipo clase (@@) y no depende de una instancia particular, no hay una necesidad estricta de un constructor explícito para cada objeto de Rfid. Esto crea una instancia compartida de MFRC522, lo que significa que todas las - instancias de **Rfid** usarán el mismo lector, evitando conflictos de acceso en los pines GPIO.

- Creamos un bucle **while** controlado por la variable **quedat**, que permanece en 1 hasta que la lectura sea exitosa. Dentro del bucle, se utilizan las mismas líneas que antes. Cuando la lectura es exitosa, el bloque `else` del manejo de excepciones establece **quedat** en 0, permitiendo salir del bucle.

- Una vez fuera del bucle, el UID obtenido (en formato decimal) **se convierte a hexadecimal** y se guarda en un array uid.

  Con **id_dec.length.times** recorre cada elemento del arreglo uid_dec tantas veces como elementos tenga uid_dec. Se pasa a hexadecimal y se transforma en una cadena concatenada de caracteres en mayúsculas usando **uid.join().upcase**. Finalmente, el método **read_uid** retorna el UID en formato de cadena.

- En la parte final del código, fuera de la clase Rfid, en el bloque **if __FILE__ == $0**, se crea una instancia de la clase Rfid y se llama al método puts uid, mostrando el UID leído en la consola.
  Este bloque final garantiza que el código dentro de él solo se ejecute cuando el archivo se ejecute directamente. Cuando usemos este código como una biblioteca, esta parte no se ejecutaría.

2. Script PUZZLE 2

Primero instalamos la librería gtk3 en nuestro sistema con comando: **sudo apt-get install libgtk-3-dev** y la gema con comando: **sudo gem install gtk3**

Se ha optado por dividir el código en dos archivos:

PUZZLE2.rb

En este archivo se realiza el lanzamiento del proceso gráfico y de lectura del dispositivo. Las únicas acciones que llevamos a cabo con los objetos gráficos son interrelacionarlos.

```
require "gtk3"
require_relative 'PUZZLE1.rb'
require_relative 'widget_options.rb'

rf = Rfid.new                                                    #Objecte únic RFID

def scan_tag(rf, info_label)
  thr = Thread.new {                                             #Fil auxiliar bloquejant per llegir RFID
    uid =  rf.read_uid                                           #Lectura
    puts "Tag detected: " + uid                                  #Mostrem UID per command-prompt.
    info_label.set_markup("uid: " + uid)                         #Mostrem UID per pantalla.
    info_label.override_background_color(0, Gdk::RGBA::new(1,0,0,1))  #Modificació a color vermell de l'etiqueta.
  }
end

#S'ha utilitzat el fitxer widget_options.rb, on tenim, de manera separada,
#les variables de configuració per una manipulació més senzilla.

window = get_window            #Finestra. Objecte gràfic que encapsula tots els objectes gràfics
grid = get_grid                #Graella. Utilitat per organitzar objectes a la finestra
info_label = get_label         #Etiqueta. Canvia de color i text segons estat de l'aplicació
clear_button = get_button      #Botó 'clear' (per tornar a llegir RFID)
```

```
23
24    #Afegir objectes a graella, determinant la seva posició.
25    grid.attach(info_label,0,0,1,1)
26    grid.attach(clear_button,0,1,1,1)
27    window.set_window_position(:center) #Pantalla al centre
28
29
30    clear_button.signal_connect("clicked") do #Actuació en cas de accionar botó
31      reset_label(info_label)               #Reestabliment blau i missatge
32      scan_tag(rf, info_label)
33    end
34
35    window.signal_connect('destroy') { Gtk.main_quit } #Botó de tancament pantalla gràfica, finalitza aplicació
36
37
38    # Run Application
39    window.add(grid) #Afegim graella amb etiqueta i botó a la finestra visible
40    window.show_all  #Mostrem elements
41    scan_tag(rf, info_label) #Invoquem mètode d'escanejament per primer cop
42
43    Gtk.main                 #Bucle gràfic
```

Creamos una interfaz muestra una ventana con una etiqueta de texto que cambia de color y muestra el UID (identificador único) de una etiqueta RFID al detectarla. También incluye un botón para restablecer la interfaz y escanear nuevamente.

- **Bibliotecas:** El código carga la biblioteca **GTK3** para manejar la interfaz gráfica. También se importa el **PUZZLE1.rb** que incluye la clase Rfid y **widget_options.rb** que define métodos y configuraciones para la interfaz.

- **Inicialización del Lector:** Se crea una instancia de la clase Rfid con rf = Rfid.new

- **scan_tag**: ejecuta la lectura del UID de una etiqueta RFID en un hilo separado (Thread.new) para evitar que la interfaz gráfica se bloquee. Dentro de este hilo:

    - **rf.read_uid** lee el UID de la etiqueta detectada. El UID se muestra en la consola con puts. Finalmente, El texto de la **info_label** (etiqueta en la ventana) se actualiza para mostrar el UID, y su color de fondo cambia a rojo para señalar que se ha detectado una etiqueta.

- **Creación de los Componentes de la Interfaz:**

    - **window**, la ventana principal, configurada mediante **get_window** y personalizada con título, tamaño y márgenes.
    - **grid,** una cuadrícula que organiza los elementos dentro de la ventana, creada con **get_grid**.
    - **info_label**, una etiqueta de texto que muestra mensajes y el UID de la etiqueta, configurada con **get_label**.
    - **clear_button**, un botón con la etiqueta "Clear", creado con **get_button.** Este botón permite reiniciar la interfaz.

- **Estructuración de la Interfaz en la Cuadrícula:** Los elementos info_label y clear_button se añaden a la cuadrícula **(grid.attach),** especificando sus posiciones. La window se centra en pantalla **(window.set_window_position(:center),** y luego se añade la cuadrícula a la ventana visible.

- **Interacción:**

    - Al hacer clic en clear_button, se llama a **reset_label**, que reinicia el mensaje y el color de la info_label para permitir un nuevo escaneo. Luego, se llama a **scan_tag** para iniciar la lectura.
    - La ventana tiene un evento de cierre **(window.signal_connect('destroy')** que termina la aplicación.

- **Ejecución de la Aplicación**:

    - Finalmente, se muestran todos los elementos de la ventana **(window.show_all).**

    - **scan_tag(rf, info_label)** se ejecuta por primera vez para que el lector de RFID es té activo desde el inicio.
    - **Gtk.main** inicia el bucle principal de GTK para mantener la interfaz abierta y receptiva a las interacciones del usuario.
    -

En conjunto, este código configura una aplicación gráfica para lectura de etiquetas RFID que responde al escaneo de una tarjeta, muestra el UID y permite reiniciar el proceso con un botón clear

widget_options.rb

En este archivo invocamos los métodos de creación de los objetos gráficos de la biblioteca GTK y realizamos las modificaciones pertinentes. También organizamos y describimos de forma más estructurada los diversos atributos configurables de los elementos gráficos.

```ruby
require "gtk3"

#Paràmetres de configuració de finestra:
@titol_finestra = "Lector RFID"
@res_ample = 400
@res_altura = 125
@marge = 20

#Paràmetres de configuració de 'label' (etiqueta de color)
@missatge = "Please, login with your university card"

#Paràmetres de configuració del botó
@mis_boto = "Clear"

def get_window #Retorna objecte finestre
        window = Gtk::Window.new(@titol_finestra)
        window.border_width = @marge
        window.title = @titol_finestra
        window.set_default_size @res_ample, @res_altura
        return window
end


def get_grid #Retorna objecte graella
        grid = Gtk::Grid.new
        return grid
end

def get_label #Retorna objecte etiqueta
        label = Gtk::Label.new(@missatge)
        label.set_size_request(@res_ample - @marge, @res_altura - @marge)
        label.override_background_color(0, Gdk::RGBA::new(0,0,1,1)) #Blue
        label.override_color(0 , Gdk::RGBA::new(1.0, 1.0, 1.0, 1.0))#Blanc
        return label
end

def reset_label (label) #Mètode executat quan hi ha ID a pantalla i es vol tornar a llegir
        label.set_markup(@missatge) #Reestabliment missatge
        label.override_background_color(0, Gdk::RGBA::new(0,0,1,1)) #Tornem a color blau
end


def get_button #Retorna objecte botó
        button = Gtk::Button.new(:label => @mis_boto)
        return button
end

```

Este código configura una interfaz gráfica en Ruby utilizando GTK3 para crear una aplicación sencilla que incluye una ventana, una etiqueta (label), y un botón. Los elementos tienen parámetros personalizados de tamaño, color, y texto, y cada uno de ellos se genera a través de métodos específico.

- **Parámetros de Configuración Inicial:** Se define el título de la ventana, ancho y la altura de la ventana. Establece el margen entre los elementos de la interfaz y los bordes de la ventana. El mensaje que aparecerá inicialmente en la etiqueta y el texto que se mostrará en el botón, en este caso, “Clear”.

- **get_window** : Crea la ventana principal de la aplicación usando Gtk::Window. La ventana se configura con el título, ancho, altura y margen especificados anteriormente.

- **get_grid**: Crea una cuadrícula (Gtk::Grid), que funciona como un contenedor en el cual los elementos de la interfaz pueden organizarse fácilmente en filas y columnas.

- **get_label**: Crea una etiqueta que muestra el mensaje definido en @missatge) Se dimensiona con el ancho y la altura definidos en @res_ample y @res_altura, restando el margen para dejar espacio en los bordes.

- Las funciones **override_background_color y override_color**, configuran el color de fondo de la etiqueta en azul y el color del texto en blanco para que se vea claramente

- **reset_label**: Permite restablecer la etiqueta a su estado inicial. Cambia el mensaje de la etiqueta de vuelta al texto original de @missatge y restablece el color de fondo a azul.

- **get_button**: Crea un botón con el texto especificado en @mis_boto ("Clear"). Este botón se usa en la interfaz gráfica que posteriormente en el PUZZLE2 se usara cuando limpiemos.

Este script nos permite poder separar la creación de la interfaz gráfica del código principal, en el cual solo las entrelazamos. De esta forma cualquier cambio a nivel grafico pude ser realizado en este script puede ser llamado con el método sin tener que volver a escribir la configuración

<u>Problemas encontrados</u>

- Ocupación permanente del puerto SPI al intentar escanear una tarjeta RFID múltiples veces. Tras un escaneo inicial exitoso, los intentos posteriores generan un error que impide realizar más escaneos y bloquea la aplicación
  Para liberar el puerto SPI y los recursos GPIO después de cada lectura, la solución fue: **Uso de variable class type**
  La variable que representa el lector RFID se ha convertido en una variable de clase, es decir, una variable que pertenece a toda la clase y no solo a una instancia individual.
  Las variables de clase son compartidas por todas las instancias de esa clase.

  Al utilizar una variable de clase para el lector, el código asegura que solo haya una instancia del lector RFID que controle el acceso a los puertos GPIO. Esto previene conflictos de acceso, ya que todas las instancias de la clase usarán la misma variable de lector y no intentarán acceder al puerto GPIO de forma independiente.

- Como bien dice en el documento, volver a instalar la librería ffi y todas las relacionadas con ella ya que da problemas con la librería mrfc522 de ruby