Sebastián Román

Memòria Puzzle2

Implementació d'Interfície d'Usuari per a la Lectura de Targetes RFID

1. Introducció

Aquest projecte implementa una interfície gràfica d'usuari (GUI) en Ruby per a la lectura de targetes RFID utilitzant les llibreries -**GTK3** - i -**Thread**-. La interfície permet que l'usuari apropi una targeta al lector RFID i visualitzi l'identificador únic (UID) en pantalla. Això es complementa amb un botó de "Clear" que permet esborrar la informació mostrada i reiniciar la lectura. A través de la llibreria externa -puzzle1-, l'aplicació interactua amb el lector RFID.

La implementació fa ús de **threads** per gestionar la lectura de l'UID de manera asíncrona, evitant bloquejar la GUI. La GUI ha de mantenir-se sempre activa i receptiva, fins i tot quan es llegeix el UID, un procés que podria ser lent o bloquejant en cas de no utilitzar fils.

En aquest projecte, la lectura del UID de la targeta RFID és una operació que es pot considerar de llarg termini, ja que pot trigar a completar-se. Aquest procés de lectura pot ser **bloquejant** en el sentit que, mentre el lector RFID espera per una targeta, l'execució del programa queda aturada en aquest punt fins que el UID sigui llegit. Si aquesta lectura s'executés directament en el fil principal, la GUI es congelaria durant aquest temps d'espera. Això és un problema crític en aplicacions gràfiques, ja que provoca que la interfície no pugui respondre als usuaris fins que la lectura s'hagi completat.

2. Estructura general del codi

El codi es divideix en les següents seccions:

1. Importació de llibreries: 'gtk3', 'thread' i 'puzzle1'.

2. Classe 'Window': Defineix la interfície gràfica i la funcionalitat de l'aplicació.

3. Mètodes principals de la classe 'Window': Inclouen el mètode 'initialize', el mètode per gestionar el botó "Clear" (on_clear_clicked) i el mètode 'rfid' per a la lectura del UID.

4. Inicialització de l'aplicació: Crea la finestra, inicia la lectura RFID i el bucle principal de GTK.

3. Codi i expilació dels diferents blocs:

Sebastián Román

-Importació de llibreries

```ruby
require "gtk3"
require "thread"
require_relative "puzzle1"
```

-La Classe Window : variables

La classe Window hereda de 'Gtk::Window' i representa la finestra principal. Aquí es defineixen les propietats de la finestra i els elements de la interfície.

```ruby
class Window < Gtk::Window
  def initialize
    super
    set_title 'rfid_gtk.rb' # Estableix el títol de la finestra
    set_border_width 10     # Defineix el marge de la finestra
    set_size_request 500, 200 # Tamany de la finestra
```

- Configuració de la finestra: Defineix el títol, el marge i el tamany, i incialitza les variables.

```ruby
    # Conecta la senyal "destroy" de la finestra per a tancar  la aplicació
    signal_connect("destroy") do
      Gtk.main_quit            # Surt del loop principal de GTK
      @thread.kill             # Atura el thread de lectura si está en execució
    end
```

- Aquí li diem a l'aplicació què ha de fer quan es tanqui la finestra del GUI . Es tanca l'aplicació amb Gtk.main_quit. S'atura qualsevol fil (o procés en segon pla) que estigui actiu amb @thread.kill

-Contenidor **Box**: Un contenidor vertical que organitza els elements dins la finestra.

Posar @ davant de les variables en ruby serveix per referenciar aquella com una variable de instància i per tant estarà disponible per altres mètodes dins de la instància Window.

```ruby
    # Crea un contenedor vertical per organizar els  widgets/funcionalitats dins de la finestra
    @hbox = Gtk::Box.new(:vertical, 6)
    add(@hbox)
```

-Elements de la interfície:

- Label : Mostra un missatge d'instruccions a l'usuari.

- Button : Permet esborrar el UID i reiniciar la lectura.

```ruby
    # Crea un Label que mostra el missatge de instruccions al usuari
    @label = Gtk::Label.new("Por favor, acerque su tarjeta al lector")
    @label.override_background_color(0, Gdk::RGBA.new(0, 0, 1, 1)) # Fons blau
    @label.override_color(0, Gdk::RGBA.new(1, 1, 1, 1))          # Text blanc
    @label.set_size_request(100, 200) # Tamany del label
    @hbox.pack_start(@label)          # Afegeix el Label al contenidor

    # Crea un botó de "Clear" para esborrar la informació del UID llegit
    @button = Gtk::Button.new(label: 'Clear')
    @button.signal_connect('clicked') { on_clear_clicked } # Asocia el event "clicked" amb el mètode on_clear_clicked
    @button.set_size_request(100, 50) # Tamany del botó
    @hbox.pack_start(@button)         # Afegeix el botó al contenidor
  end
```

3.3 Mètodes de la Classe ‘Window’

Els mètodes més rellevants són:

```ruby
  # Mètode per gestionar l'event depreionar el botó "Clear"
  def on_clear_clicked
    # Restaura el missatge inicial en el Label
    @label.set_markup("Por favor, acerque su tarjeta al lector")
    @label.override_background_color(0, Gdk::RGBA.new(0, 0, 1, 1)) # Canvia el fons a blau
    @thread.kill                # Atura el thread de lectura si està executant
    rfid                        # Reinicia la lectura del UID
  end
```

- ‘on_clear_clicked’: Restaura el missatge del Label i mata qualsevol thread de lectura actiu abans de reiniciar el procés de lectura del UID.

```ruby
  # Mètode para iniciar el procés de lectura de UID en un thread separat
  def rfid
    @rfid = Rfid.new              # Instancia l' objecte Rfid
    # Crea un nou thread  per a  realitzar la lectura del UID
    @thread = Thread.new do
      uid = @rfid.read_uid        # Llegeix l' UID desde el lector RFID
      # Actualitza la interface en el thread principal per evitar bloqueigs
      GLib::Idle.add do
        @label.set_markup("uid: " + uid) # Mostra l'UID llegit en el Label
        @label.override_background_color(0, Gdk::RGBA.new(1, 0, 0, 1)) # Canvia el fons a vermell
        false # Retorna false para asegurar que la actualització s'executi tan  sols una cop
      end
    end
  end
end
```

- rfid: Inicia un fil per a la lectura del UID. Aquí es fa servir ‘GLib::Idle.add’ per assegurar que les actualitzacions de la GUI es fan en el fil principal, evitant bloquejos i errors de concurrència.

Si intentem modificar la interfície des d'un fil secundari, podem provocar bloquejos.

Per aquesta raó, utilitzem GLib::Idle.add. Aquesta funció ens permet programar l'execució d'un bloc de codi en el fil principal**,** fins i tot si el cridem des d'un fil secundari. Així, quan llegim l’UID en un fil secundari, utilitzem **GLib::Idle.add** per assegurar que qualsevol canvi a la GUI (com mostrar el UID o canviar el color del text) es faci de forma segura en el fil principal.

```
62  # Crea y mostra una instància de la finestra principal
63  win = Window.new
64  win.show_all
65  win.rfid # Inicia el mètode de lectura RFID
66  Gtk.main # Inicia el loop principal de la interfazinteface gràfica
```

-Execució dels mètodes per posar en marcha l'aplicació.

4. Problemes amb la biblioteca ffi i errors en **runtime** amb GTK

La biblioteca ffi (Foreign Function Interface) és una llibreria que permet a Ruby fer crides a codi natiu, com ara biblioteques escrites en C. Algunes biblioteques de lectors RFID que utilitzen ffi poden causar problemes quan treballen amb **GTK**. Aquests errors poden aparèixer quan executem l'aplicació , i poden ser deguts a problemes de compatibilitat entre les versions de ffi i altres biblioteques. Per això reinstalarem la biblioteca ffi a la RPI:

Amb el comandament: gem install ffi

5. Conclusions

Aquest projecte il·lustra com implementar una GUI que interactua amb un dispositiu extern en Ruby, fent ús de fils per evitar bloquejos i assegurant que les actualitzacions de la interfície gràfica es facin en el fil principal. A més, utilitza 'GLib::Idle.add' per evitar problemes amb operacions gràfiques multithread, una pràctica necessària per treballar amb llibreries no thread-safe com GTK.

Els problemes potencials com el bloqueig de la GUI i la gestió de múltiples fils es resolen de manera efectiva, i el codi final ofereix una estructura modular i escalable per afegir funcionalitats o integrar nous dispositius.