Laboratorio de MAC-IPP
Profesor: Luís Vera Quiroga
Alumno ayudante: Patricio Villalon
Sergio Villena

Practico Nº2 Comunicación Serial y Socket

Este código es un cliente de chat en Python usando la interfaz de usuario Tkinter. Establece una conexión con un servidor utilizando el protocolo TCP, envía el nombre de usuario al servidor y luego inicia un hilo de recepción de mensajes para recibir los mensajes del servidor. El cliente también incluye una interfaz de usuario para que el usuario ingrese un mensaje para enviar al servidor.

Aclaración: es importante desactivar el firewall y conocer la ip del servidor para poder conectarlo

Desarrollo

1 Agregue las siguientes bibliotecas:

```
import socket
import tkinter as tk
from threading import Thread
import tkinter.messagebox as messagebox
import time
```

2 Crearemos las variables necesarias para poder conectarnos como servidor

```
HOST = 'localhost'  # Servidor al que se va a conectar
PORT = 8888  # Puerto del servidor al que se va a conectar
```

3 __init__: Esta es la función de inicialización de la clase. Establece los componentes de la interfaz de usuario, establece el nombre del usuario y luego llama a la función

```
class ClientGUI:
    def __init__(self, master):
        self.master = master
        master.title('Cliente')

        self.name = tk.StringVar()
        self.name.set("USUARIO1")

        self.received_messages_text = tk.Text(master, height=10, width=50)
        self.received_messages_text.pack()

        self.message_entry = tk.Entry(master, width=50)
        self.message_entry.pack()

        self.name_entry = tk.Entry(master, width=50, textvariable=self.name)
        self.name_entry.pack()

        self.send_button = tk.Button(master, text='Enviar', command=self.send_message)
        self.send_button.pack()

        self.connected = False
        self.connect_to_server()
```

4 connect_to_server: Esta función intenta establecer una conexión con el servidor. Si se conecta con éxito, enviará el nombre del usuario al servidor y luego iniciará el hilo de recepción de mensajes.

```
def connect_to_server(self):
    while True:
        try:
            # Crear un objeto de socket TCP
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            # Conectarse al servidor
            self.socket.connect((HOST, PORT))

            # Configurar la variable de conexión
            self.connected = True

            # Enviar el nombre al servidor
            self.socket.sendall(self.name.get().encode())

            # Iniciar el hilo de recepción de mensajes
            self.receive_thread = Thread(target=self.receive_messages)
            self.receive_thread.start()

            break  # Si se conectó, salir del bucle

        except Exception as e:
            print(e)

            # Configurar la variable de conexión
            self.connected = False

            # Esperar 5 segundos antes de intentar reconectar
            time.sleep(5)
```

5. receive_messages: Esta función maneja la recepción de mensajes del servidor. Recibirá los mensajes del servidor y los agregará a la caja de texto de la interfaz de usuario.

```
def receive_messages(self):
    while self.connected:
        try:
            # Recibir datos del servidor
            data = self.socket.recv(1024)

            # Si no hay datos, el servidor cerró la conexión
            if not data:
                break

            # Agregar el mensaje recibido a la caja de texto
            self.received_messages_text.insert(tk.END, data.decode() + '\n')
            self.received_messages_text.see(tk.END)

        except Exception as e:
            print(e)
            self.connected = False
            self.socket.close()
            break
```

6. send_message: Esta función maneja el envío de mensajes al servidor. Obtiene el mensaje ingresado por el usuario en la interfaz de usuario, verifica si está conectado al servidor y luego envía el mensaje al servidor.

```
def send_message(self):
    # Obtener el mensaje ingresado en el cuadro de texto
    message = self.message_entry.get()

    # Verificar si estamos conectados al servidor
    if not self.connected:
        messagebox.showerror('Error', 'No se pudo enviar el mensaje: no hay conexión con el servidor.')
        return

    # Verificar si el socket aún está abierto
    if not self.socket._closed:
        # Enviar el mensaje al servidor
        self.socket.sendall(message.encode())

        # Borrar el cuadro de texto de mensaje
        self.message_entry.delete(0, tk.END)
    else:
        # Mostrar un mensaje de error si el socket está cerrado
        messagebox.showerror('Error', 'No se pudo enviar el mensaje: la conexión con el servidor se ha perdido.')
```

7.Este último fragmento de código crea una instancia de la clase ClientGUI y luego ejecuta el bucle principal de Tkinter, que es responsable de manejar todos los eventos de la interfaz de usuario.

```
if __name__ == "__main__":
    root = tk.Tk()
    client_gui = ClientGUI(root)
    root.mainloop()
```