Documentazione Progetto Programmazione di Reti

Alessio Bifulco

June 10, 2024

Traccia del Progetto

Progetto di Programmazione di Reti: Implementare un sistema di chat client-server in Python utilizzando socket programming. Il server deve essere in grado di gestire più client contemporaneamente e deve consentire agli utenti di inviare e ricevere messaggi in una chatroom condivisa. Il client deve consentire agli utenti di connettersi al server, inviare messaggi alla chatroom e ricevere messaggi dagli altri utenti.

1 Descrizione del Sistema

Il sistema di chat client-server implementato consente a più client di connettersi a un server centralizzato e di scambiare messaggi all'interno di una chatroom condivisa. Ogni client può inviare e ricevere messaggi in tempo reale, rendendo possibile la comunicazione tra tutti i partecipanti connessi.

2 Funzionamento del Sistema

2.1 Server

Il server è progettato per accettare connessioni da più client contemporaneamente e per gestire la trasmissione dei messaggi all'interno della chatroom. Quando un messaggio viene inviato da un client, il server lo riceve e lo inoltra a tutti gli altri client connessi, escluso il mittente.

Il server utilizza i seguenti componenti principali:

- **Socket**: Utilizzato per la comunicazione di rete.
- **Threading**: Utilizzato per gestire le connessioni multiple contemporaneamente.
- **Lista dei Client**: Mantiene traccia dei client connessi.

2.2 Client

Il client permette agli utenti di connettersi al server, di inviare messaggi e di ricevere i messaggi inviati dagli altri utenti nella chatroom. Il client dispone di una semplice interfaccia grafica (GUI) che facilita l'interazione con il sistema.

Il client utilizza i seguenti componenti principali:

- **Socket**: Utilizzato per la comunicazione di rete.
- **Threading**: Utilizzato per ricevere messaggi dal server in modo asincrono.
- **Tkinter**: Libreria utilizzata per creare l'interfaccia grafica.

3 Requisiti per Eseguire il Codice

Per eseguire il sistema di chat client-server, è necessario disporre di:

- Python 3.x installato sul sistema.
- La libreria Tkinter, che è generalmente inclusa con Python.
- Connessione di rete stabile per la comunicazione tra server e client.

4 Implementazione

4.1 Codice del Server

```
import socket
import threading

# Lista per tenere traccia dei client connessi
clients = []

# Funzione per gestire la trasmissione dei messaggi
def broadcast(message, sender_client):
    i = 0
    while i < len(clients):
        client = clients[i]
        if client != sender_client:
            try:
                client.sendall(message)
            except Exception as e:
                print(f"Errore durante l'invio del messaggio: {e}")
                clients.remove(client)
        i += 1
```

```
# Funzione per gestire i client connessi
def handle_client(client, address):
    print(f"Connessione stabilita con {address}")
    connected = True
    while connected:
        try:
            message = client.recv(1024)
            if not message:
                connected = False
            broadcast(message, client)
        except Exception as e:
            print(f"Errore nella connessione con {address}: {e}")
            connected = False

    print(f"Connessione chiusa con {address}")
    clients.remove(client)
    client.close()

# Funzione principale per avviare il server
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 54000))
    server.listen(5)
    print("Server avviato su porta 54000")

    while True:
        client, address = server.accept()
        clients.append(client)
        thread = threading.Thread(target=handle_client, args=(client, address))
        thread.start()

if __name__ == "__main__":
    start_server()
```

4.2 Codice del Client

```
import socket
import threading
import tkinter as tk
from tkinter import simpledialog, scrolledtext

# Funzione per ricevere messaggi dal server
def handle_incoming_messages(client_socket, display_area):
    connected = True
    while connected:
        try:
```

```
            msg = client_socket.recv(1024).decode('utf-8')
            if not msg:
                connected = False
            display_area.config(state=tk.NORMAL)
            display_area.insert(tk.END, msg + '\n')
            display_area.config(state=tk.DISABLED)
            display_area.yview(tk.END)
        except Exception as e:
            print(f"Errore nella ricezione del messaggio: {e}")
            connected = False

# Funzione per inviare messaggi al server
def submit_message(client_socket, input_field, display_area, username):
    msg = input_field.get()
    if msg:
        complete_msg = f"{username}: {msg}"
        try:
            client_socket.sendall(complete_msg.encode('utf-8'))
        except Exception as e:
            print(f"Errore nell'invio del messaggio: {e}")
        input_field.delete(0, tk.END)
        display_area.config(state=tk.NORMAL)
        display_area.insert(tk.END, complete_msg + '\n')
        display_area.config(state=tk.DISABLED)
        display_area.yview(tk.END)

# Configurazione della GUI del client
def initialize_client_gui():
    # Finestra principale per chiedere l'username
    root = tk.Tk()
    root.withdraw()  # Nasconde la finestra principale temporaneamente

    username = simpledialog.askstring("Username", "Inserisci il tuo username:", parent=root)
    if not username:
        return  # Esci se l'utente non fornisce un username

    root.deiconify()  # Mostra la finestra principale
    root.title(username)
    root.geometry("450x500")

    # Connessione al server
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect(('127.0.0.1', 54000))
    except ConnectionRefusedError as e:
        print(f"Errore di connessione: {e}")
```

```
        return

    # Configurazione dell'area di testo per visualizzare i messaggi
    display_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, font=("Arial", 12), bg="lig
    display_area.config(state=tk.DISABLED)
    display_area.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    # Configurazione del frame di input per l'invio dei messaggi
    input_frame = tk.Frame(root, bg="light blue")
    input_frame.pack(padx=10, pady=5, fill=tk.X)

    input_field = tk.Entry(input_frame, font=("Arial", 12))
    input_field.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=10, pady=10)

    send_button = tk.Button(input_frame, text="Invia", font=("Arial", 12), bg="blue", fg="wh
    send_button.pack(side=tk.RIGHT, padx=10, pady=10)

    # Thread per gestire la ricezione dei messaggi dal server
    thread = threading.Thread(target=handle_incoming_messages, args=(client_socket, display_
    thread.start()

    # Avvio della GUI
    root.mainloop()

if __name__ == "__main__":
    initialize_client_gui()
```

5 Gestione degli Errori e delle Eccezioni

Il codice è progettato per gestire vari tipi di errori ed eccezioni:

- **Errore durante l'invio dei messaggi**: Se un client non è più disponibile, il server lo rimuove dalla lista dei client.
- **Errore di connessione**: Se un client non riesce a connettersi al server, viene visualizzato un messaggio di errore.
- **Errore nella ricezione dei messaggi**: Se un client non riesce a ricevere messaggi, il thread di ricezione termina e viene visualizzato un messaggio di errore.

6 Considerazioni Aggiuntive

- **Scalabilità**: Il sistema può essere scalato per gestire un numero maggiore di client, ma potrebbe essere necessario implementare ulteriori ottimizzazioni per garantire prestazioni ottimali.

- **Sicurezza**: La sicurezza non è stata considerata in questa implementazione di base. In un'applicazione reale, sarebbe necessario implementare meccanismi di autenticazione e crittografia per proteggere i dati trasmessi.

7 Conclusione

Il sistema di chat client-server implementato soddisfa i requisiti di base per la comunicazione in una chatroom condivisa. La gestione delle connessioni multiple e la trasmissione dei messaggi avvengono correttamente, e il client fornisce un'interfaccia grafica semplice per l'interazione con il sistema. La gestione degli errori è implementata per garantire che eventuali problemi vengano gestiti in modo appropriato.