Sistema di Chat Client-Server in Python

Nome Autore

July 15, 2024

Contents

1 Introduzione

Questo documento descrive l'implementazione di un sistema di chat client-server in Python utilizzando la programmazione con socket. Il sistema consente a più client di connettersi a un server e scambiare messaggi in una chatroom condivisa.

2 Requisiti

- Libreria `threading` (inclusa nella libreria standard di Python)
- Libreria `tkinter` (inclusa nella libreria standard di Python)

3 Funzionamento del Sistema

Il sistema di chat client-server funziona come segue:

- Il server accetta connessioni da più client e utilizza il multithreading per gestire le comunicazioni in parallelo.
- Quando un client invia un messaggio, il server lo trasmette a tutti gli altri client connessi.

4 Codice del Server

```
1
2  import socket
3  import threading
4
5  clients = []
6
7  def broadcast(message, client_socket):
8      for client in clients:
9          if client != client_socket:
10             try:
11                 client.send(message)
12             except:
13                 client.close()
14                 clients.remove(client)
15
16 def handle_client(client_socket):
17     while True:
18         try:
19             message = client_socket.recv(1024)
20             if message:
21                 print(f"Received message: {message.decode('utf-8')}")
22                 broadcast(message, client_socket)
23             else:
24                 client_socket.close()
25                 clients.remove(client_socket)
26                 break
27         except:
28             client_socket.close()
29             clients.remove(client_socket)
30             break
31
32 def main():
33     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
34     server.bind(('0.0.0.0', 5555))
```

```
    server.listen(5)
    print("Server is listening on port 5555...")

    while True:
        client_socket, addr = server.accept()
        print(f"Connection from {addr} has been established.")
        clients.append(client_socket)
        client_thread = threading.Thread(target=handle_client, args=(
            client_socket,))
        client_thread.start()

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

5 Codice del Client con GUI

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

class ChatClient:
    def __init__(self, master):
        self.master = master
        self.master.title("Chat Client")

        self.master.configure(bg='black')

        self.chat_display = scrolledtext.ScrolledText(self.master, wrap
            =tk.WORD, state='disabled', bg='black', fg='white',
            insertbackground='white', selectbackground='gray')
        self.chat_display.pack(padx=20, pady=5)

        self.message_entry = tk.Entry(self.master, width=50, bg='black'
            , fg='white', insertbackground='white')
        self.message_entry.pack(padx=20, pady=5, side=tk.LEFT)
        self.message_entry.bind("<Return>", self.send_message)

        self.send_button = tk.Button(self.master, text="Send", command=
            self.send_message, bg='gray', fg='black')
        self.send_button.pack(padx=5, pady=5, side=tk.LEFT)

        self.client_socket = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
        self.server_address = ('127.0.0.1', 5555)
        self.connect_to_server()

    def connect_to_server(self):
        try:
            self.client_socket.connect(self.server_address)
            threading.Thread(target=self.receive_messages, daemon=True)
                .start()
        except Exception as e:
            self.display_message(f"Unable to connect to server: {e}")

    def receive_messages(self):
```

```
        while True:
            try:
                message = self.client_socket.recv(1024).decode('utf-8')
                if message:
                    self.display_message(message)
                else:
                    self.client_socket.close()
                    break
            except Exception as e:
                self.display_message(f"Error receiving message: {e}")
                self.client_socket.close()
                break

    def send_message(self, event=None):
        message = self.message_entry.get()
        if message:
            try:
                self.client_socket.send(message.encode('utf-8'))
                self.display_message(f"You: {message}")
                self.message_entry.delete(0, tk.END)
            except Exception as e:
                self.display_message(f"Error sending message: {e}")
                self.client_socket.close()

    def display_message(self, message):
        self.chat_display.configure(state='normal')
        self.chat_display.insert(tk.END, message + '\n')
        self.chat_display.configure(state='disabled')
        self.chat_display.yview(tk.END)

def main():
    root = tk.Tk()
    client = ChatClient(root)
    root.mainloop()

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

6 Gestione degli Errori

Il codice include la gestione delle eccezioni per garantire che le connessioni chiuse o i messaggi non validi non causino il blocco del server o dei client. Ad esempio, i blocchi `try-except` sono utilizzati per gestire errori di connessione e invio/ricezione dei messaggi.

7 Ottimizzazioni

1. **Threading**:
   - *Descrizione*: Utilizzare il threading sia sul client che sul server permette di gestire l'invio e la ricezione dei messaggi in parallelo.
   - *Benefici*: Evita il blocco dell'interfaccia utente e consente al server di gestire più client contemporaneamente.
2. **Buffering**:
   - *Descrizione*: L'uso del metodo `recv` con un buffer di 1024 byte.

- *Benefici*: Riduce il numero di chiamate di rete necessarie, migliorando l'efficienza della trasmissione dei dati.

3. **Gestione delle eccezioni**:
   - *Descrizione*: Miglioramento della gestione delle eccezioni per affrontare disconnessioni e altri errori.
   - *Benefici*: Assicura che il sistema continui a funzionare anche in presenza di problemi di connessione.
4. **Architettura scalabile**:
   - *Descrizione*: Progettazione del server per gestire più client contemporaneamente e trasmettere i messaggi a tutti i client connessi.
   - *Benefici*: Migliora la scalabilità del sistema, consentendo di supportare un numero maggiore di utenti.

8 Conclusioni

Il progetto ha dimostrato come creare un sistema di chat room client-server utilizzando Python e socket, con l'implementazione di threading per gestire più connessioni simultanee e altre ottimizzazioni per migliorare la velocità e l'efficienza della comunicazione.