Progetto Client-Server in Python

Il presente documento fornisce una panoramica del progetto inerente al corso Programmazione di Reti dell'Università Alma Mater Studiorum - Bologna.

Sommario

Introduzione

Lo scopo del progetto è quello di implementare un sistema chat client-server in Python che simuli una chatroom condivisa. L'utente deve essere in grado di connettersi ad un server per inviare e ricevere messaggi. Il server ha il requisito di supportare più utenti (connessioni) simultaneamente. Il progetto prevede la possibilità di gestione di eventuali errori ed eccezioni.

Funzionamento del Sistema

Il sistema si suddivide in due componenti principali: client e server. Il primo è responsabile di interagire con l'utente tramite una finestra di dialogo per l'inserimento del nome utente (fig. 1). Successivamente una schermata principale permette la visualizzazione e l'invio di messaggi (fig. 2).

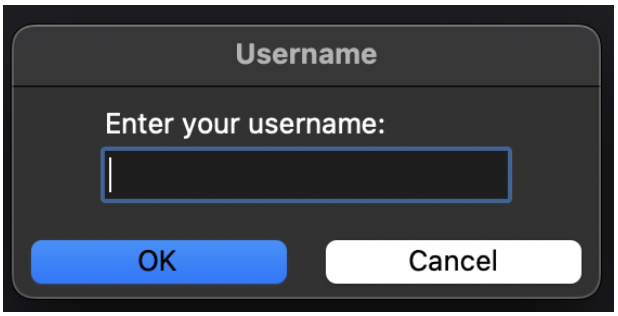

figura 1

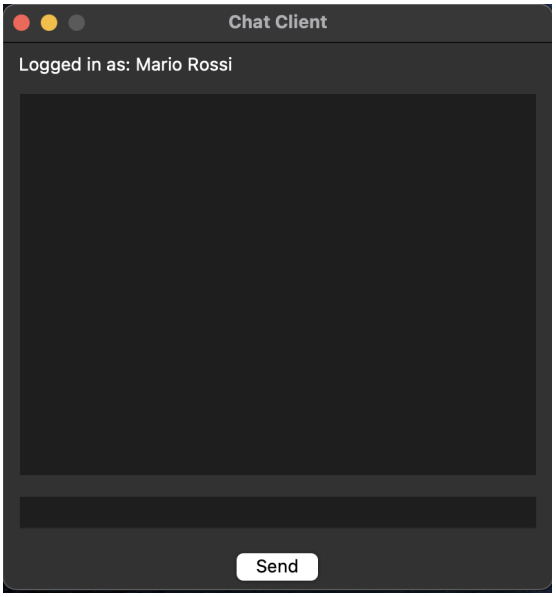

figura 2

Componenti del Client

Il client quindi si presenta con:

- una label che mostra l'utente attualmente connesso
- una text area che mostra i tutti gli eventuali messaggi
- una entry box per l'inserimento del messaggio
- un bottone per l'invio del messaggio (in alternativa si può usare il tasto Enter sulla tastiera)

Componenti del Server

Il server agisce senza una sua parte grafica. La sua funzione è quella di stampare una stringa sul terminale quando un nuovo utente si connette al sistema.
es: Connection established with ('127.0.0.1', 56616)

Funzionamento nel dettaglio del Client

Il client è responsabile di gestire la comunicazione con il server utilizzando socket TCP/IP per inviare e ricevere messaggi. Si compone di diversi metodi come:

- **main():** inizialmente nasconde la finestra principale di tkinter (root.withdraw()) e chiede all'utente di inserire un nome utente valido utilizzando una finestra di dialogo (simpledialog.askstring). Se l'utente non inserisce un nome utente, viene mostrato un messaggio di errore. Successivamente il client crea un socket TCP (socket.socket(socket.AF_INET, socket.SOCK_STREAM)) e tenta di connettersi al server specificando l'indirizzo IP '127.0.0.1' e la porta 5555. Se la connessione al server fallisce per un errore di connessione (ConnectionError), viene mostrata una finestra di errore corrispondente.
  Dopo aver stabilito la connessione con successo, la finestra principale di tkinter viene mostrata (root.deiconify()). Viene istanziato l'oggetto ChatClient, passando come parametri la finestra principale di tkinter (root), il socket del client (client_socket) e il nome utente (username). Infine, il client inizia a eseguire il ciclo principale (root.mainloop()).

- **Classe ChatClient(self, root, client_socket, username):** dopo aver inizializzato l'oggetto viene avviato un thread (self.receive_thread) che chiama il metodo receive_messages() per gestire la ricezione continua dei messaggi dal server.

- **receive_messages():** dopo aver preso l'input dall'utente e formattato aggiungendone il nome, invia il messaggio al server utilizzando self.client_socket.send(). Cancella l'input dalla entry box e aggiunge il messaggio alla text area (self.display_message()). Se ci sono errori il client chiude la socket.

Funzionamento nel dettaglio del Server

Il server è pensato per gestire connessioni multiple tramite thread, utilizzando socket TCP/IP

- **main():** è la funzione che inizializza la socket (server.socket(socket.AF_INET, socket.SOCK_STREAM)) e la associa al corrispondente indirizzo ip e porta utilizzando la server.bind(). Il server inizia così un loop infinito (while: True) dove per ogni connessione stabilita con successo avvia un thread (threading.Thread(*target*=handle_client, *args*=(client_socket,))

- **handle_client(client_socket):** riceve i messaggi inviati dal client (client_socket.recv(1024).decode('utf-8')), li decodifica in formato UTF-8 e lo inoltra a tutti i client connessi utilizzando la funzione broadcast() se il messaggio non è vuoto.

- **broadcast(message, client_socket):** invia un messaggio a tutti i client tranne quello che lo ha inviato (client_socket). Itera attraverso la lista dei client connessi (clients) e utilizza client.send() per inviare il messaggio a ognuno dei client.

Modalità di Utilizzo

1. Recarsi nella directory “client-server-merighi” dove risiedono i due file “client.py” e “server.py”

   ```
   cd .../client-server-merighi
   ```

2. Assicurarsi di avere installato python3 e le due librerie necessarie: tkinter e socket
3. Eseguire prima il server (altrimenti comparirà un messaggio di errore)

   ```
   python3 server.py
   ```

4. Eseguire il client

   ```
   python3 client.py
   ```