Sistema di chat client-server

Zanchini Margherita

1 Introduzione

Il progetto si concentra sul sistema di chat client-server. Dopo che il server è stato avviato, ad esso potranno connettersi uno o più client, questi client potranno comunicare tra di loro attraverso un'interfaccia.
Ciascun client invia i messaggi al server, il quale si occupa di inoltrarli a tutti gli altri client, compreso il mittente del messaggio.
L'applicazione è stata sviluppata utilizzando il linguaggio di programmazione Python e sfrutta le socket TCP per gestire le connessioni.

2 Funzionamento del server

Per prima cosa si configura il server, indicando l'indirizzo completo su cui sarà in ascolto. Nel programma, la variabile che contiene questa informazione è chiamata 'ADDR', in essa sono combinate l'interfaccia di rete e la porta. Viene poi creato il socket TCP per il server e legato all'indirizzo 'ADDR'.

Il programma usa poi la funzione '**accetta_connessioni**' per la gestione dei nuovi client che vogliono connettersi al server. Per ogni nuovo client connesso il server crea un thread in modo da poter gestire più client in contemporanea.
La funzione 'accetta_connessioni' utilizza l'istruzione bloccante 'SERVER_SOCKET.listen()' che blocca l'esecuzione del programma fino a quando non arriva una richiesta di connessione da un client.
Dopo aver mandato un messaggio di benvenuto, l'indirizzo del nuovo client connesso viene aggiunto all'interno di un dizionario chiamato 'indirizzi'. Infine, viene attivato un nuovo thread per quel client, così che il server possa gestire il client e allo stesso tempo accettare nuove connessioni. Il thread utilizza la funzione 'gestione_client'

La funzione '**gestione_client**' si occupa di gestire la comunicazione tra il server e il client. Il nome scelto dal client viene ricevuto e aggiunto su un dizionario chiamato 'clients'. Viene poi inviato un messaggio per informare tutti i client (con una funzione chiamata 'broadcast' che sarà spiegata successivamente) che un nuovo client si è unito alla chat.
Il server si mette in attesa di nuovi messaggi provenienti dal client. Se il messaggio ricevuto è uguale a '{quit}', allora la connessione con quel client viene chiusa, altrimenti viene condiviso il messaggio a tutti gli altri.

Infine c'è la funzione '**broadcast**', che si occupa di inviare il messaggio passato come parametro a tutti i client connessi, compreso il mittente. Il messagio viene combinato con il nome del mittente, il messaggio è poi codificato e inviato a ogni client presente del dizionario 'clients'.

3 Funzionamento del client

Il client si connette di default all'indirizzo '127.0.0.1' e alla porta '60000', ma questi valori possono essere cambiati modificando il contenuto delle variabili 'HOST' e 'PORT'. Combino queste due informazioni nell'indirizzo completo chiamato 'ADDR' e con esso mi connetto al server.

Il client utilizza una GUI creata con la librearia tkinter.

L'interfaccia grafica comprende un'area in cui sono visualizzati tutti i messaggi inviati con la possibilità di scorrere con la scrollbar per visualizzare i messaggi più vecchi. Comprende poi un'area in cui è possibile digitare il proprio messaggio e inviarlo attraverso il pulsante 'Invio'. Un ulteriore bottone, 'Esci', consente di disconnettersi dal server e chiudere la finestra.

Il client crea un thread per la ricezione dei messaggi, così che possa ricevere messaggi senza bloccare l'interfaccia utente. Il thread utilizza la funzione '**receive**'.
La funzione istanzia un ciclo infinito in cui ogni messaggio ricevuto viene prima decodificato e poi visualizzato nell'area dedicata sull'interfaccia grafica.

La funzione '**send**' viene richiamata ogni volta che l'utente preme il tasto 'Invio' sulla GUI.
Per prima cosa viene preso il messaggio dal campo di input dell'interfaccia grafica, e immediatamente dopo il campo viene svuotato. Il messaggio viene poi codificato e inviato al server. Se invece il messaggio corrisponde a '{quit}' viene richiamata la funzione 'close_connection'.

La funzione '**close_connection**' può essere richiamata in tre diverse situazioni e serve per terminare la connessione tra server e client.
Il primo caso si verifica quando l'utente preme il pulsante 'Esci' sulla GUI.
Il secondo caso si verifica quando l'utente digita sul campo di input '{quit}' e invia il messaggio.
Il terzo caso si verifica quando viene premuta la x rossa nell'angolo in alto a destra della GUI.
Questa funzione invia al server il messaggio '{quit}' garantendo così che in qualsiasi dei casi menzionati sopra, anche il server riceverà l'istruzione di chiudere la connessione con il client. Successivamente il client chiude la connessione e la finestra dell'interfaccia grafica.

4 Esecuzione

- La prima cosa da fare è avviare il server, da terminale si esegua il comando 'python server.py'. In questo modo il server si metterà in ascolto.
- Successivamente avviare il primo client, da terminale si esegua il comando 'python client.py'. Il client si connetterà automaticamente al server all'indirizzo 127.0.0.1 e porta 60000. Si può cambiare la configurazione all'interno del codice del client.
- Ripetere il punto precedente per il numero di client che si vuole connettere al server.
- Ora il server e i vari client saranno connessi e si può testare la chat.