Sistema di Chat Client-Server

Indice

Introduzione

Si vuole realizzare l'implementazione di un sistema di chat client-server in Python utilizzando socket programming.
Lo scopo è quello di creare un sistema in cui il server può gestire connessioni multiple di client in maniera sincrona, permettendo la comunicazione in tempo reale tra più utenti.

Architettura e funzionamento del Sistema

La chat viene realizzata con due programmi di rete: un programma server e un programma client.
- **server.py**: gestisce la connessione dei client e la distribuzione dei messaggi.
- **client.py**: permette ai singoli utenti di connettersi al server, di inviare e ricevere messaggi.

La persona che esegue il programma server resta in attesa che l'altra persona si colleghi usando il programma client. Quando la connessione è stata attivata, la chat può avere inizio.

Requisiti

Il sistema richiede l'installazione di Python 3.x e delle librerie standard 'socket', 'threading' e 'tkinter' per l'interfaccia grafica.

Progettazione e implementazione

Il socket è l'oggetto attraverso il quale è possibile instaurare la connessione e che consente lo scambio dei dati. Per questo scopo useremo i socket TCP, data la necessità di comunicazioni affidabili, ordinate e semplici da implementare.

Lato Client

Il client si connette al server tramite socket TCP, inserendo l'indirizzo del server e il numero di porta per stabilire la connessione, infine invia il proprio nome utente.

È dotato di un'interfaccia grafica Tkinter che consente agli utenti di inserire messaggi da inviare al server tramite socket. Un thread separato è dedicato alla ricezione dei messaggi dal server, che vengono poi visualizzati nella GUI.

Funzioni chiave:

- **send():** Gestisce l'invio dei messaggi al server.
- **receive():** Gestisce la ricezione dei messaggi dal server in un thread separato.
- **gen_view():** Crea la finestra della chat e gli elementi della GUI.
- **end():** Gestisce la chiusura della connessione e dell'applicazione.

Lato Server

Il server viene avviato e si mette in ascolto su un indirizzo IP e una porta specifici.
Il server accetta le connessioni dei client e avvia un thread separato per ciascun client connesso. Gestisce l'invio dei messaggi ricevuti da un client in broadcast a tutti i client attivi.
Inoltre, gestisce la disconnessione dei client rimuovendoli dalla lista dei client attivi e inviando un messaggio agli altri client. È dotato di un'interfaccia Tkinter per visualizzare i messaggi in arrivo e monitorare lo stato del server.

Funzioni chiave:

- **broadcast(msg):** Invia un messaggio a tutti i client connessi. La funzione cicla all'interno del dizionario che contine i cliet e invia sul relativo socket il contenuto del messaggio inviato dal client mittente.
- **delete(client):** Rimuove un client dalla lista e invia un messaggio ai client ancora attivi della sua disconnessione.
- **client_man(client):** Gestisce la comunicazione con un singolo client. All'interno della funzione viene chiamata la funzione broadcast per inviare messaggi a tutti i client.
- **client_accept():** Accetta nuove connessioni dai client in un thread separato.
- **gen_view():** Crea la finestra del server e gli elementi della GUI.
- **fine():** Gestisce la chiusura del server e delle connessioni.

Esecuzione

All'avvio del dello script server.py viene creata la finestra server, in cui viene specificato l'indirizzo IP e la porta (di default 5001) su cui il server è in ascolto. Il client potrà connettersi alla chat specificando il proprio nome utente, l'indirizzo IP e la porta. Una volta connesso potrà scrivere messaggi (inviandoli premendo il tasto "Invio"), che verranno visualizzati sia sulla finestra client sia sulla finestra server.
Il client può lasciare la chat scrivendo "{quit}" o chiudendo la propria finestra:

il server comunicherà a tutti gli utenti rimasti connessi (e trascriverà sulla propria finestra) il messaggio che segnala l'uscita del client.
Chiudendo la finestra del server, quest'ultimo provocherà la chiusura della finestra degli utenti connessi, avvisandoli della disconnessione tramite console.
Quando un client si disconnette, o viene disconnesso, da un server, avrà la possibilità di tentare una nuova connessione.

Avvio del server

Il server deve essere avviato prima di qualsiasi client.
Avviare un terminale e spostarsi nella locazione di memoria nel quale si trova il file server.py ed eseguirlo con il seguente comando:
`python server.py`

Avvio del client

Prima di tutto bisogna assicurarsi di aver avviato il server, successivamente avviare un terminale, diverso rispetto a quello di avvio del server, e spostarsi nella locazione di memoria nel quale si trova il file client.py ed eseguirlo con il seguente comando:
`python client.py`