Francesco Barzanti, matricola 0001078545

CHAT CLIENT-SERVER

1. INTRODUZIONE

Il sistema di chat Client-Server consiste per appunto in un server che gestisce le comunicazioni e due client che si connettono ad esso per scambiare messaggi. Questo sistema permette ai client di comunicare tra loro attraverso il server.

DEFINIZIONE DI ELEMENTI NECESSARI

- Python

- Modulo `socket` di Python (incluso nella libreria standard)

-Utilizzo di socket TCP (potremmo usare anche i socket UDP ma, TCP si adatta meglio in questo caso).

- Modulo `threading` di Python (incluso nella libreria standard)

- Due dizionari per la registrazione dei client in ingresso e dei relativi nomi associati agli indirizzi

-La porta e l'interfaccia del server

-Modulo `tkinter` di Python (GUI)

-Il bind

2. LATO SERVER (CHAT_SERVER.PY)

Inizializzazione: Il server crea un socket e lo associa ad un Indirizzo IP e alla porta `53000`.

A. ASCOLTO DELLE CONNESSIONI

Il server si mette in ascolto delle connessioni in entrata.

Viene definita una funzione "**accetta_connessioni_in_entrata()"** che gestisce le connessioni in entrata, quando viene invocata si mette in ascolto sul socket.

B. GESTIONE DELLE CONNESSIONI

Appena arriva una richiesta di connessione da un client, il server crea un nuovo thread per gestire la comunicazione con quel client restituendo un messaggio di saluto accompagnato da alcune indicazioni sull'utilizzo della chat.

Per utilizzare un thread si crea un'istanza con una funzione target e si invoca "**start()"**per inizializzarlo.
Per attendere fino a quando un thread ha completato il suo lavoro, si utilizza il metodo join.

La stessa funzione "**accetta_connessioni_in_entrata()**" inoltre pende il dizionario «indirizzi» e lo aggiorna con il dato relativo al nuovo client.
Infine, attiva il Thread verso questo client andando ad invocare la funzione "**gestice_client(client)".**
Quest'ultima funzione si occuperà di gestire il Thread per ciascun client che entra nella Chat.

C. BROADCAST DEI MESSAGGI

I messaggi ricevuti da un client vengono trasmessi a tutti gli altri client connessi. All'interno della funzione "**gestice_client(client)**" verrà chiamata la funzione broadcast per inviare i messaggi a tutti gli utenti.

La funzione "**broadcast(msg, prefisso="")**" «cicla» all'interno del dizionario che contiene «i clients» e invia sul relativo socket il contenuto del messaggio inviato dal client mittente accompagnato dal prefisso. Il prefisso è fornito in ingresso al momento della chiamata della funzione e corrisponde al nome dell'utente che ha inviato il messaggio.

3. LATO CLIENT (CHAT_CLIENT.PY CHAT_CLIENT2.PY)

A. CONNESSIONE AL SERVER

Ogni client crea un socket e si connette al server.

Una volta ottenuto l'indirizzo e la porta da parte dell'utente e creato un socket per connettersi al server, si da inizio al thread per la ricezione dei messaggi e quindi al ciclo principale (la GUI) con l'utilizzo di **Tkinter**, uno strumento incluso in Python.

B. INIVIO E RICEZIONE DEI MESSAGGI

Il processo di invio e ricezione dei messaggi in un sistema di chat Client-Server avviene all'interno di un ciclo infinito. Questo ciclo consente di ricevere messaggi quando si è connessi e inviarli quando si vuole.

1. RICEZIONE MESSAGGI

La ricezione dei messaggi avviene in un thread separato per consentire la comunicazione in tempo reale.

La ricezione dei messaggi è gestita dalla funzione "**recv()**", che è bloccante. Questo significa che l'esecuzione del programma si interrompe fino a quando non viene ricevuto un messaggio.
Una volta ricevuto, il messaggio viene aggiunto a msg_list, un elemento di Tkinter utilizzato per visualizzare l'elenco dei messaggi sullo schermo.
END (o "end") rappresenta la posizione del cursore subito dopo l'ultimo carattere nel buffer, assicurando che i nuovi messaggi siano aggiunti alla fine della lista.

2. INVIO MESSAGGI

   L'invio dei messaggi avviene quando l'utente interagisce con l'interfaccia della GUI, ad esempio premendo il pulsante di invio. Tkinter passa implicitamente l'evento alla funzione associata, che estrae il messaggio dal campo di input my_msg con “my_msg.get()”. Dopo aver ottenuto il messaggio, il campo di input viene cancellato utilizzando my_msg.set(). Il messaggio viene quindi inviato al server. Se il messaggio è "{quit}", il client chiude la connessione socket e termina l'applicazione GUI.

C. CHIUSURA DELLA CONNESSIONE

   Per gestire la chiusura della finestra della GUI, si definisce una funzione di pulizia che chiude la connessione socket prima di chiudere l'applicazione GUI. Se si vuole semplicemente uscire dal programma, è possibile usare destroy(). Se si vuole eseguire un ciclo infinito senza distruggere la finestra Tkinter, si utilizza quit().

   Ho utilizzato entrambe le funzioni per problema con l’interfaccia in chiusura. Andava in buffering la finestra, era possibile chiuderla solo con l’uscita forzata.

4. PROCEDURA PER L’AVVIO DELLA CHAT

A. AVVIO ANACONDA NAVIGATOR

B. AVVIO SPYDER

C. CARICAMENTO FILE

D. ESECUZIONE SERVER

   Con il file `chat_server.py` aperto in Spyder, cliccare sul pulsante di esecuzione (l'icona del triangolo verde) per avviare il server.

E. ESECUZIONE CLIENT

   - Aprire una nuova istanza di Spyder o un nuovo file nel pannello già aperto.

   - Caricare `chat_client.py` e cliccare sul pulsante di esecuzione per avviare il primo client.

   Ripetere questo passaggio con `chat_client2.py` per avviare il secondo client. Per ogni client eseguito verrà richiesto di inserire l’indirizzo IP e la porta per la connessione al server.

F. INTERAZIONE CLIENT

   - I client possono ora inviare e ricevere messaggi tramite il server.

   - Digitare '{quit}' per chiudere la connessione.

5. CONSIDERAZIONI AGGIUNTIVE

A. GESTIONE DEGLI ERRORI

Il codice include una gestione degli errori. Sono stati fatti miglioramenti includendo un trattamento più sofisticato delle eccezioni e una gestione delle disconnessioni inattese.

B. SCALABILITA’

Questo esempio è pensato per essere semplice e didattico. In un ambiente di produzione, si potrebbero considerare soluzioni più avanzate per la scalabilità e la sicurezza, come l'uso di framework specifici per le applicazioni di rete.