Documentazione del sistema di chat client-server

Marco Magnani, MATRICOLA: 0001097698

Descrizione del sistema

Il sistema è diviso in 3 codici sorgente uno per il client uno per il server ed uno per il file di configurazioni, esso implementa una chat client-server che permette a più utenti di comunicare tra loro tramite una connessione TCP. Il server gestisce le connessioni in entrata dai client e instrada i messaggi tra di essi, mentre i client possono inviare e ricevere messaggi da altri utenti connessi al server.
Per garantire una configurazione e una gestione agevoli del server, è stato implementato un file di configurazione dedicato. Questo file contiene una serie di informazioni utili per la corretta comunicazione e configurazione del server. In particolare sono presenti: Indirizzo IP del server, porta del server, dimensione dei messaggi, codifica dei messaggi

Server

Il server accetta le connessioni in entrata dai client e gestisce la comunicazione broadcast tra di essi. Quando un nuovo client si connette, il server verifica il nome utente univoco e gestisce l'invio e la ricezione dei messaggi. Importante caratteristica del server è la gestione delle eccezioni, server è in grado di gestire diverse eccezioni che possono verificarsi durante la comunicazione, possono essere eccezioni specifiche, come un errore di connessione con un client, ma anche eccezioni più generiche, quindi errori non gestiti nel server.

Le parti più importanti del codice del server includono:

Configurazione del socket

Il server utilizza un socket TCP per consentire la comunicazione con i client. Viene creato utilizzando la funzione *socket.socket()*, specificando *socket.AF_INET* per l'indirizzamento IPv4 e *socket.SOCK_STREAM* per il tipo di socket TCP. Successivamente, il server viene associato a un indirizzo e a una porta utilizzando il metodo *bind()*. Come indirizzo ho usato il localhost mentre come porta ne ho presa una libera ovvero la 54523

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind((SERVER_IP, PORT))
```

Accettazione delle connessioni

Il server ascolta le connessioni in arrivo utilizzando il metodo *listen()* e accetta le connessioni dei client con il metodo *accept()*. Quando un client si connette, il server crea un nuovo thread per gestire la comunicazione con quel client.

```
server.listen()
client_socket, addr = server.accept()
```

Gestione dei client connessi

Ogni client è gestito da un thread separato per consentire la gestione di client multipli del server. Quando il client si connette come prima cosa ne prende il nome utente e controlla se è già presente un altro utente con quel nome, in tal caso annulla la connessione, il server mantiene una lista dei client connessi in modo tale da inviare i messaggi ricevuti da un client a tutti gli altri client connessi. Quando invia il messaggio specifica anche lo username del client che lo ha inviato.

```
client_handler =
threading.Thread(target=handle_client, args=(client_socket, addr, clients))

client_handler.start()
```

Client

Il client è responsabile di connettersi al server e di inviare e ricevere messaggi dalla chatroom condivisa. Anche qui come nel server c'è una gestione delle eccezioni e degli errori. Le parti più importanti del codice del client includono:

Connessione al server

Il client utilizza un socket TCP per connettersi al server utilizzando l'indirizzo IP e la porta. Questi dettagli vengono passati come argomenti da riga di comando quando si esegue il programma.

client.connect((server_ip, server_port))

Ricezione dei messaggi

Il client riceve i messaggi dalla chatroom utilizzando un thread separato. Quando arriva un messaggio, lo stampa a schermo.

receive_thread = threading.Thread(target=receive_messages, args=(client,))

receive_thread.start()

Invio dei messaggi

Il client invia i messaggi alla chatroom digitandoli da tastiera e utilizzando il metodo *send()* del socket per inviarli al server.

message = input()

client_socket.send(message.encode('utf-8'))

Istruzioni per l'esecuzione del codice

Per eseguire il codice del server e del client, seguire i seguenti passaggi:

1. Aprire un terminale o prompt dei comandi.
2. Navigare nella directory contenente il file Python del server (server.py).
3. Eseguire il comando *python server.py* per avviare il server.
4. Aprire un altro terminale o prompt dei comandi.
5. Navigare nella directory contenente il file Python del client (client.py).
6. Eseguire il comando python client.py indirizzo_ip_del_server porta_server username per avviare il client.

Considerazioni aggiuntive

Affinché il client funzioni correttamente, il server deve essere attivo. L'assegnazione di un nome utente univoco è cruciale per distinguere gli utenti nella chat room ma anche a migliorare la comunicazione broadcast, infatti è stato necessario il suo utilizzo anche per evitare che il client riceva i propri messaggi.