PROGETTO PROGRAMMAZIONE DI RETI

NERI ELENA

MATRICOLA: 0001098044

TRACCIA 1: SISTEMA DI CHAT CLIENT-SERVER

Implementare un sistema di chat Client-Server in Python utilizzando socket programming. Il server deve essere in grado di gestire più client contemporaneamente e deve consentire agli utenti di inviare e ricevere messaggi in una chatroom condivisa. Il client deve consentire agli utenti di connettersi al server, inviare messaggi alla chatroom e ricevere messaggi dagli altri utenti.

INTRODUZIONE

Il codice in linguaggio Python fa in modo di consentire a più client di connettersi anche simultaneamente allo stesso server, di inviare e ricevere messaggi in una stessa chat room e infine uscire in qualsiasi momento gli stessi vogliano.

SERVER E CLIENT

L'intero programma è costituito da due componenti principali:

1. **SERVER**: è ciò che si rende disponibile al client; In particolar modo, in questo caso, si occupa di:
    - accettare le connessioni in entrata
    - gestire i loro comportamenti
    - inviare messaggi ai client connessi
    - controllare quanti client sono presenti
    - terminare le connessioni
2. **CLIENT**: è ciò che utilizza i servizi messi a disposizione dal server; In questo programma quest'ultimocsi occupa di:
    - Connettersi al server e ricevere messaggi da esso
    - Inviare messaggi dal client al server
    - Realizzare l'interfaccia grafica della chatroom
    - Terminare la connessione del client e dell'interfaccia grafica

ESECUZIONE:

Al fine di eseguire correttamente i file è necessario seguire il seguente ordine di avvio da terminale:

1. Selezionare la corretta directory con il comando cd
2. Eseguire il comando ***python Server.py***
3. Eseguire il comando ***python Client.py***

(al fine di permettere connessioni da più client è possibile eseguire da più terminali lo stesso file Client.py oppure lanciare i file duplicati Client2.py e Client3.py)

ANALISI SERVER:

Import os : dichiarazione Python per accedere alle funzionalità che interagiscono con il sistema operativo. È utilizzato per chiamare la funzione ***os._exit(0)*** per chiudere il server in maniera rapida una volta che tutti i client si sono disconnessi

import sys: utilizzato anch'esso per terminare il programma, ma assicurandosi che termini correttamente

import signal: utilizzato per catturare il segnale di interruzione per interrompere il processo in esecuzione.

Inbound_connection()

È la funzione che serve per gestire l'accettazione delle connessioni dei client in entrata che tentano di connettersi al server della chat. Nello specifico, viene eseguito un ciclo infinito, il quale continua ad accettare connessioni da parte dei client. All'interno del ciclo, la funzione utilizza il metodo ***accept*** del socket del server per accettare nuove e crea una tupla contenente socket e indirizzo IP del client. Una volta accettata la nuova connessione, la funzione stampa a terminale un messaggio che indica l'indirizzo IP e la porta del client che si è appena connesso, invia al client un messaggio di benvenuto e viene avviato un nuovo thread per gestire la comunicazione con il Client. La funzione gestisce gli eventuali errori che possono verificarsi durante l'accettazione delle connessioni: in caso di errore viene stampato un messaggio di errore e il ciclo viene interrotto.

manageClient()

Funzione responsabile di gestire singolarmente ogni client connesso al server: il server riceve, tramite il socket, il nome utente del client al quale invia un messaggio che indica che è stato connesso con successo alla chat room e viene aggiunto alla lista clients (con ***chiave=socket, valore=nome utente***). La funzione entra anch'essa in un ciclo infinito dove riceve tutti i messaggi dei client e nel momento in cui viene cliccato il ***bottone QUIT***, esso viene rimosso dalla lista dei clients connessi e viene inviato un messaggio di notifica a tutti gli altri client presenti nella chat. Questa funzione gestisce eventuali errori che potrebbero verificarsi durante la ricezione o l'invio dei messaggi e in caso di errore, il client viene rimosso dalla lista dei client connessi venendo così chiusa la connessione con lo stesso; nel caso tutti i client avessero lasciato il server, esso si chiude.

broadcast()

la funzione è utilizzata per inviare un messaggio a tutti i client connessi alla chat room. Essa itera su tutti i client connessi e per ognuno il messaggio (passato alla funzione come parametro) viene inviato utilizzando il metodo ***send()*** del socket del client.

Check_empty_clients()

Verifica se non ci sono più client connessi al server, se clients è vuoto significa che tutti i client hanno lasciato la chat; quindi, il server stampa a video un messaggio e chiude il processo del server.

Sigint_handler()

Gestisce il segnale ***SIGINT*** al fine di garantire una chiusura corretta del server nel momento in cui questo segnale viene ricevuto. Nello specifico la funzione stampa dapprima un messaggio di chiusura, successivamente chiude il socket del server con il metodo ***close()*** e infine termina il processo tramite ***sys.exit(0)***

ANALISI CLIENT:

import tkniter: è una libreria standard del linguaggio Python utilizzata per la creazione di interfacce grafiche.

Receive()

Funzione che gestisce la ricezione dei messaggi dal server e visualizzarli nell'interfaccia grafica della chat. Entra in un ciclo infinito all'interno del quale tenta di ricevere un messaggio; nel caso venisse ricevuto il messaggio (la stringa non è vuota), esso viene inserito in coda nella ***message_list***. Se si verifica un'eccezione di tipo ***OSError*** il ciclo si ferma.

Stop()

È la funzione responsabile di chiudere il socket del client e distruggere l'interfaccia nel momento in cui l'utente client vuole terminare la connessione; questo potrebbe generare eccezioni, che però vengono comunicate tramite un messaggio su terminale "errore durante la chiusura del socket". Una volta chiuso il socket si procede a chiudere anche la finestra dell'interfaccia grafica: anche questo può generare un'eccezione di tipo ***tkt.TclError***, se l'errore stampa un'eccezione di tipo 'can't invoke' allora la finestra è stata distrutta e non è necessario fare più nulla.

Send()

Invia messaggi scritti dall'utente al server: è chiamata ogni volta che l'utente preme il ***bottone SEND*** oppure che preme il ***tasto Enter***. La funzione, una volta ricevuta la stringa di messaggio, se non vuota, viene inviata al server tramite il socket del client. Se si verifica un'eccezione di tipo ***ConnectionResetError*** viene chiamata la funzione ***stop()*** al fine di chiudere il socket e distruggere la finestra dell'interfaccia grafica.

Check_window_count()

Gestisce il conteggio delle finestre aperte e chiude di conseguenza il programma se tutte le finestre sono state chiuse tramite la variabile globale ***open_windows***. Quando questa è uguale a 0, tutte le finestre sono state chiuse e viene chiamata la funzione ***sys.exit(0)***

Start()

È il punto di ingresso per avviare il client della chat. Host di default viene inizializzato come ***localhost*** e la porta associata è ***53000***. Viene poi creato un socket del client che tenta di connettersi al server: se la connessione fallisce, si stampa un messaggio di errore e la funzione ritorna senza continuare. Successivamente viene creata la finestra principale della GUI e tutti gli oggetti che vanno a creare l'interfaccia (frame, scrollbar, listbox, send button, quit button). Viene creato un thread per eseguire la funzione ***receive()*** che ascolta i messaggi in arrivo del server e li visualizza nella GUI. Infine è eseguito il ciclo principale di Tkinter ***tkt.mainloop()***, responsabile della gestione dell'interfaccia e degli eventi utente.