Documentazione progetto
Programmazione di reti

Nome e cognome: Federico Mantoni, Matricola:0001091295

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.

Funzionamento del sistema

Il sistema è composto da due componenti principali:

- Il server: Gestisce le connessioni alla chatroom di più client, la ricezione e l'invio dei messaggi ai client collegati, la comunicazione ai client di entrata e uscita degli altri client dalla chat
- Il client: Permette la connessione e disconnessione dal server e fornisce una GUI per l'invio e la visualizzazione dei messaggi.

Come utilizzare

Requisiti necessari:

- Connessione ad internet
- Python 3.x e librerie socket, threading e tkinter installate nel sistema

Esecuzione:

1. Eseguire chat_server.py: Il server si metterà in ascolto per la connessione da parte dei client
2. Eseguire più istanze di chat_client.py su terminali diversi: specificare l'indirizzo del dispositivo dove è stato avviato il server (127.0.0.1 se viene eseguito tutto nello stesso dispositivo) e la porta in ascolto (default 53000, non è necessario specificarla)
3. Inserire il nome e provare ad inviare un messaggio: scrivere un messaggio nella casella di testo sottostante la lista dei messaggi e premere invio (o nella tastiera o il bottone nella GUI)
4. Se tutto funziona correttamente, tutti i client riusciranno a visualizzare i messaggi inviati dagli altri client
5. Si può terminare la connessioni in 2 modi:

- Inviare il messaggio “!quit” in chat
- Premere il bottone X della finestra

Funzionamento approfondito

In entrambi i moduli, all’inizio vengono inizializzate le variabili necessarie per la corretta connessione e comunicazione

- HOST: l'indirizzo IP su cui il server è in ascolto.
- PORT: la porta su cui il server è in ascolto.
- BUFSIZ: la dimensione del buffer per la ricezione dei messaggi.
- QUIT: Il messaggio da inviare per terminare la connessione
- ENCODING: La codifica di caratteri utilizzata

Funzionamento del server

Il server inizialmente si mette in ascolto di connessioni da parte di client.
accetta_connessione_client():
Il server accetta la connessione da parte del client, memorizza l’indirizzo e porta del client connesso e chiede l’inserimento di un nome per essere mostrato successivamente in chat. Dopodiché esegue il thread per la gestione del client

gestione_client(client):
Il server memorizza il nome scelto dal client, invia un messaggio di benvenuto al client e comunica alla chatroom della sua entrata.
Rimane in ascolto per ricevere i messaggi di un client e trasmetterli a tutti gli altri. Gestisce anche la fine della connessione nel caso il messaggio inviato corrisponda a QUIT

broadcast(messaggio, nome=""):
La funzione con la quale effettivamente il server va ad inviare un messaggio a tutti i client, iterando tra la lista dei client connessi

Funzionamento del client

All’inizio vado a settare l’interfaccia grafica del client e chiedo in input indirizzo e porta del server host, se la connessione va a buon fine, mostro la GUI

ricevi_messaggio():
Si occupa della ricezione e della corretta visualizzazione dei messaggi, inclusi quelli mandati dal client stesso.

invia_messaggio(event=None):
Ascolta l'evento della pressione del tasto invio. Si occupa di leggere il testo scritto nella casella di testo ed inviare il messaggio al server, nel caso il messaggio inviato corrisponda a QUIT, la connessione ed il processo verranno terminati

on_closure(event=None):
Ascolta l'evento della chiusura della finestra. Invia il messaggio QUIT al server.

Considerazioni aggiuntive:

- Sono state implementate gestione di errori ed eccezioni dove necessario sia lato client che lato server
- il sistema utilizza socket TCP (tipo SOCK_STREAM) per garantire una comunicazione affidabile tra client e server
- Viene fatto uso dei thread:
    - Lato client per permettere l'uso reattivo della GUI in concorrentemente con la ricezione dei messaggi
    - Lato server per gestire concorrentemente l'invio di messaggi e l'accettazione di nuove connessioni