RELAZIONE

Documentazione chiara e ben strutturata in un PDF che spiega il funzionamento del sistema, i requisiti per eseguire il codice e eventuali considerazioni aggiuntive.

Traccia:

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:

Questo documento fornisce una guida dettagliata per l'implementazione di un sistema di chat client-server utilizzando socket programming in Python. Il sistema consente a più client scelti inizialmente all'avvio del server di connettersi contemporaneamente al server, inviare messaggi nella chatroom condivisa e ricevere messaggi dagli altri utenti ed uscire.

Architettura del sistema:

Il sistema è costituito da due componenti principali:

- server: gestisce le connessioni dei client, la trasmissione dei messaggi e il coordinamento della chatroom.
- client: fornisce un'interfaccia grafica per gli utenti, consentendo loro di connettersi al server, inviare messaggi e visualizzare i messaggi ricevuti dagli altri utenti.

Per eseguire il codice del sistema di chat client-server, sono necessari i seguenti requisiti:

- python 3.x installato sul sistema.
- accesso a una rete per consentire la comunicazione.

Istruzioni per l'esecuzione:

Server:

- eseguire il file "server.py" utilizzando il comando su terminale "python server.py".
- specificare il numero massimo di connessioni simultanee richieste.

Client:

- eseguire il file "client.py" utlizzando il comando "python client.py".
- inserire i requisiti richiesti come host e porta oppure spingere "invio" per inserire di "default".
- iniziare a inviare e ricevere messaggi nella chatroom.

Funzionamento codice:

Lato Server

Funzione accept_incoming_connections()

Questa funzione gestisce l'accettazione delle connessioni in arrivo dai client. In particolare:

- Inizializzazione Variabili: Inizializza una variabile connections_count per monitorare il numero di connessioni attuali.
- Accettazione Connessioni: Entra in un ciclo while che continua finché il numero di connessioni accettate è inferiore al numero massimo consentito (MAX_CONNECTIONS). Durante ogni iterazione del ciclo:
    - Accetta una nuova connessione utilizzando SERVER.accept() e memorizza il client e il suo indirizzo.
    - Invia un messaggio di benvenuto al client.
    - Avvia un nuovo thread per gestire il client tramite la funzione handle_client().
    - Incrementa il contatore delle connessioni.

- Gestione Errori: Gestisce gli errori e le eccezioni che possono verificarsi durante l'accettazione delle connessioni.

Funzione handle_client(client):
Questa funzione gestisce la comunicazione con un singolo client. In particolare:
- Ricezione Nome del Client: Riceve il nome del client inviato dal client stesso.
- Messaggio di Benvenuto: Invia un messaggio di benvenuto al client.
- Aggiunta Client alla Lista: Aggiunge il client alla lista dei client connessi.
- Ciclo di Gestione dei Messaggi: Entra in un ciclo while che continua finché il client non invia un messaggio di uscita ({quit}). Durante ogni iterazione del ciclo:
    - Riceve i messaggi inviati dal client.
    - Trasmette il messaggio a tutti gli altri client connessi utilizzando la funzione broadcast() se il messaggio non è {quit}.
    - Chiude la connessione con il client, lo rimuove dalla lista dei client connessi e informa gli altri client della sua uscita dalla chat se il messaggio è {quit}.
- Gestione Errori: Gestisce gli errori e le eccezioni che possono verificarsi durante la gestione del client.

Funzione broadcast(msg, prefix="")
Questa funzione invia un messaggio a tutti i client connessi. In particolare:
- Iterazione sui Client Connessi: Itera attraverso tutti i client connessi nella lista clients.
- Invio del Messaggio: Invia il messaggio a ciascun client utilizzando sock.send().
- Gestione Errori: Gestisce gli errori e le eccezioni che possono verificarsi durante l'invio dei messaggi.

Variabili Globali e Inizializzazione del Server
Le variabili globali vengono inizializzate all'inizio dello script:
- 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.
- ADDR: Una tupla contenente l'indirizzo IP e la porta del server.

Lato Client
Funzione receive():
Questa funzione gestisce la ricezione dei messaggi dal server e li visualizza nella finestra della chat. In particolare:
- Loop di Ricezione Messaggi: Entra in un loop while che continua finché il client è connesso al server. Durante ogni iterazione del ciclo:
    - Riceve i messaggi dal server utilizzando client_socket.recv(BUFSIZ).
    - Decodifica i messaggi ricevuti in formato UTF-8.
    - Inserisce i messaggi nella finestra della chat utilizzando msg_list.insert(tkt.END, msg).
    - Scorre la finestra della chat per mostrare sempre l'ultimo messaggio ricevuto utilizzando msg_list.see(tkt.END).
- Gestione Errori: Gestisce le eccezioni di tipo OSError, interrompendo il loop se si verifica un errore.

Funzione send(event=None)
Questa funzione gestisce l'invio dei messaggi al server quando l'utente preme il pulsante di invio o il tasto "Invio" sulla tastiera. In particolare:
- Ottenimento del Messaggio: Ottiene il testo inserito dall'utente dall'oggetto my_msg.
- Pulizia del Campo di Input: Pulisce l'oggetto my_msg impostandolo su una stringa vuota.
- Invio del Messaggio: Invia il messaggio al server utilizzando client_socket.send(bytes(msg, "utf8")).
- Gestione Errori: Gestisce le eccezioni di tipo OSError, inserendo un messaggio di errore nella finestra della chat se si verifica un errore durante l'invio del messaggio.

Funzione on_closing(event=None):
Questa funzione gestisce la chiusura del client quando l'utente chiude la finestra della chat. In particolare:

- Invio del Messaggio di Uscita: Imposta il messaggio da inviare al server su {quit} utilizzando my_msg.set("{quit}") e chiama la funzione send() per inviare il messaggio al server.
- Gestione Errori: Gestisce le eccezioni di tipo OSError.

Funzione start_chat():
Questa funzione avvia la chat e gestisce la connessione al server. In particolare:

- Connessione al Server: Ottiene l'indirizzo IP e la porta del server dall'utente e connette il client al server utilizzando client_socket.connect(ADDR).
- Interfaccia Grafica della Chat: Crea l'interfaccia grafica della chat utilizzando Tkinter.
- Avvio Thread Ricezione Messaggi: Avvia un thread separato per gestire la ricezione dei messaggi dal server.
- Loop Principale di Tkinter: Avvia il loop principale di Tkinter per mantenere attiva l'interfaccia grafica della chat.

La variabile __name__ viene utilizzata per verificare se lo script è eseguito come programma principale, e se sì, viene chiamata la funzione start_chat() per avviare la chat

Considerazioni aggiuntive e conclusioni:

- il sistema utilizza socket TCP per garantire una comunicazione affidabile tra client e server.
- la gestione delle connessioni e dei messaggi avviene in modo asincrono utilizzando thread.
- vengono gestite eccezioni come errori di connessione e invio dei messaggi.
- le connessioni perse vengono rimosse correttamente dai dizionari.
- la gestione degli errori è progettata per gestire errori e eccezioni in modo appropriato, garantendo un'esperienza utente fluida.
- l'utilizzo di thread separati per la ricezione dei messaggi consente al client di continuare a funzionare in modo reattivo anche durante la ricezione dei messaggi.
- l'utente Tkinter offre un'esperienza intuitiva e semplice per gli utenti del client.

Nome: Nikolai.
Cognome: Zanni.
Matricola: 0002069041.