gianmaria.difronzo@studio.unibo.it
riccardo.polazzi@studio.unibo.it

Progetto Programmazione Di Reti

2024

Panoramica

Il progetto implementa un sistema di chat client-server in Python utilizzando la programmazione con i socket, permettendo la comunicazione simultanea tra più client attraverso una chatroom condivisa. Questo sistema è composto da due componenti principali: il server (Server.py) e il client (Client.py).

Descrizione Generale del Sistema

Il **server** ha il compito di gestire le connessioni dei client, ricevere e inoltrare i messaggi a tutti i partecipanti della chatroom. È progettato per supportare la comunicazione simultanea tra più client, mantenendo una lista delle connessioni attive e garantendo che ogni messaggio inviato da un client sia distribuito a tutti gli altri.

Il **client** consente agli utenti di connettersi al server, inviare messaggi e ricevere aggiornamenti in tempo reale da parte degli altri partecipanti. L'interfaccia utente del client è realizzata utilizzando il modulo tkinter, che offre una GUI semplice e intuitiva per interagire con la chatroom.

Requisiti:

Per avviare il codice del server di chat presentato, è necessario soddisfare i seguenti requisiti. Deve essere installato un sistema operativo come Windows, macOS, Linux, o qualsiasi altro sistema operativo che supporti Python. È necessario avere installato Python 3.x, preferibilmente una versione recente come Python 3.8 o superiore. Le librerie standard utilizzate nel codice sono socket, threading, sys, e time. Queste sono incluse nella libreria standard di Python, quindi non è necessario installarle separatamente. Inoltre, deve essere presente un file chiamato CustomExceptions.py nella stessa directory del server di chat, contenente le definizioni delle eccezioni personalizzate utilizzate nel codice

Durante l'avvio del server, saranno richiesti un indirizzo IP e una porta come input. L'indirizzo IP specifica l'interfaccia di rete su cui il server ascolterà le connessioni, mentre la porta specifica il punto di accesso per la comunicazione con i client. Questi parametri sono essenziali per inizializzare correttamente il server di chat e consentire ai client di connettersi.

Analogamente, durante l'avvio del client, sarà necessario fornire l'indirizzo IP e la porta del server di chat per connettersi correttamente ad esso. Questi parametri permettono al client di localizzare il server sulla rete e stabilire una connessione per partecipare alla chat.

Una volta che tutti i requisiti sono soddisfatti, puoi avviare il server eseguendo lo script principale del server di chat e fornendo l'indirizzo IP e la porta quando richiesto. Successivamente, i client dovranno utilizzare gli stessi parametri per connettersi al server e partecipare alla chat.

Come utilizzare il codice per il server e il client :

Server:

- Aprire un terminale e spostarsi nella directory contenente il file server.py.
- Eseguire il server con il comando: python ./server.py

Client:

- Aprire un altro terminale e spostarsi nella directory contenente il file client.py.
- Eseguire il client con il comando: python ./client.py

Comunicazione:

- Scrivere messaggi nella finestra del client per inviarli al server e a tutti gli altri client connessi.
- Per connettersi nuovamente al server, eseguire nuovamente il comando python ./client.py in nuovi terminale (per un massimo di 5).

Note:

- Il server deve rimanere in esecuzione finché almeno un client è connesso.
- Per chiudere il server correttamente è necessario terminare l'esecuzione dello script server.py con Ctrl+C, in questo modo i client ancora connessi verranno rimossi in modo automatico

Descrizione del Client per un Sistema di Chat

Introduzione

Il programma descritto è un client per un sistema di chat sviluppato in Python utilizzando il socket programming e il toolkit grafico Tkinter. Questo client permette agli utenti di connettersi a un server di chat, inviare messaggi a una chatroom condivisa e ricevere messaggi dagli altri partecipanti. La funzionalità chiave del programma include la gestione di connessioni multiple, l'interfaccia grafica user-friendly e la gestione degli errori di connessione.

Inizializzazione e Configurazione

All'avvio, il programma richiede all'utente di inserire l'indirizzo IP del server e la porta a cui connettersi. Una volta ottenuti questi dati, il programma crea un'istanza della classe ChatClient, la quale gestisce tutte le operazioni principali del client. Questa classe inizializza le variabili di connessione e crea una socket per la comunicazione con il server. La configurazione della connessione è gestita dal metodo setup_socket, che tenta di stabilire la connessione al server e gestisce eventuali errori di connessione con messaggi appropriati.

Interfaccia Grafica

L'interfaccia grafica del client è creata utilizzando Tkinter. La finestra principale contiene una lista di messaggi visualizzati, un campo di input per inserire nuovi messaggi e un pulsante per inviarli. La lista dei messaggi è accompagnata da una barra di scorrimento per facilitare la visualizzazione dei messaggi più recenti. Il campo di input è legato all'evento del tasto Invio, permettendo agli utenti di inviare messaggi premendo semplicemente Invio.

Ricezione e Invio dei Messaggi

Per gestire la ricezione dei messaggi dal server, il client avvia un thread separato che esegue il metodo receive_messages. Questo metodo ascolta continuamente i messaggi in arrivo attraverso il ciclo while, li decodifica e li visualizza nella lista dei messaggi dell'interfaccia grafica. Se il messaggio ricevuto è "{quit}", il client chiude la connessione e termina il programma attraverso la chiusura della socket, della gui, e l'uscita dal ciclo while. Per inviare messaggi, il metodo send_message invia il testo inserito dall'utente al server. In caso di errori durante l'invio, come la rottura della pipe, il client chiude la connessione e chiude l'interfaccia grafica.

Gestione degli Errori

Il programma è progettato per gestire vari tipi di errori di connessione e di sistema. Durante la configurazione della socket e la comunicazione, il client intercetta diverse eccezioni, come ConnectionRefusedError, ConnectionResetError, BrokenPipeError, e altri errori generici. Per ciascuno di questi errori, vengono visualizzati messaggi di errore specifici definiti nella classe CustomExceptions, e il programma esegue azioni appropriate come chiudere la connessione e terminare l'esecuzione.

Conclusione

In sintesi, il programma client per un sistema di chat è progettato per essere robusto e user-friendly. Permette agli utenti di partecipare a una chatroom condivisa con una semplice interfaccia grafica, gestisce efficacemente la ricezione e l'invio dei messaggi e si occupa di vari tipi di errori di connessione in modo elegante. Questo programma può essere utilizzato come base per applicazioni di chat più complesse, offrendo una solida infrastruttura per la comunicazione in tempo reale tra più utenti.

Descrizione del Server per un Sistema di Chat

Il codice rappresenta un server di chat scritto in Python che utilizza socket per le comunicazioni di rete e threading per gestire più client contemporaneamente. All'inizio, vengono importate diverse librerie necessarie per il funzionamento del server. La libreria socket è utilizzata per le comunicazioni di rete, threading per gestire l'esecuzione simultanea di più thread (e quindi di più client), CustomExceptions per gestire gli errori personalizzati, e sys e time per la gestione del sistema e dei ritardi temporali.

Configurazioni Iniziali

All'avvio del server, viene richiesto l'inserimento di due valori in input: serverHost e serverPort, che indicano rispettivamente l'indirizzo e la porta su cui il server ascolta le connessioni in entrata.

La classe ChatServer viene inizializzata attraverso il metodo __init__ al quale vengono passati come argomenti i due valori precedentemente inseriti . Questo metodo configura i parametri necessari per il funzionamento del server, e cioè buffer_size per la dimensione del buffer dei messaggi e address che combina host e port. Vengono anche inizializzate tre liste: clients per memorizzare le connessioni dei client sul server,names per memorizzare i loro nomi, e threads per tenere traccia dei thread di gestione. Vengono anche utilizza i flag handleThread_flag e receiveThread_flag per controllare l'esecuzione dei thread.

Infine viene creato un socket TCP con AF_INET e SOCK_STREAM, grazie al quale il server puotrà comunicare con i client.

Avvio Del Server

Il metodo start avvia il server, effettuando il binding del socket del server precedentemente creato all'indirizzo specificato e iniziando quindi ad ascoltare le connessioni in entrata(per un massimo di 5). Viene poi creato e avviato un thread detto accept_Thread che servirà esclusivamente per accettare le connessioni in entrata. Inoltre è presente un ciclo infinito che stampa un messaggio ogni 5 secondi per indicare che il server è in esecuzione, e serve per gestire l'interruzione da tastiera per spegnere il server in modo “pulito”. Quando grazie all’eccezzione KeyboardInterrupt viene rilevata la terminazione del processo del server tramite il comando CTRL + C , verrà richiamato il metodo shutdown_server , che si occupa della chiusura del server. Questo metodo notifica tutti i client connessi e chiude le loro connessioni iterando la lista dei client e inviando a ciascuno un messaggio che corrisponde a {quit} ,in questo modo tutti i client capiscono che devono disconnettersi in modo automatico. Successivamente svuota le liste dei client e dei nomi,eliminandone il loro contenuto, imposta i flag = False per interrompere i thread attivi, chiude il socket del server e attende la terminazione di tutti i thread. Se si verifica un errore durante la chiusura dei thread, viene gestito e stampato un messaggio di errore. Anche la chiusura del server Socket e dell’invio del messaggio ai clients sono gestiti dai relativi controlli di errore

Accettazione Delle Connesioni

Il metodo receive_connections gestisce le richieste di connessione in entrata tramite un thread dedicato. Questo entra in un ciclo che accetta nuove connessioni client finché receiveThread_flag è True. Per ogni nuova connessione, il server memorizza il socket del client all interno della sua lista, invia un messaggio di benvenuto e avvia un nuovo thread per gestire le

comunicazioni con il client. Viene gestita una varietà di errori , tra cui BrokenPipeError, OSError e una eccezione generica

Gestione delle connessioni

Il metodo handle_client gestisce le comunicazioni con un client specifico tramite un thread dedicato creato in precedenza. Riceve e memorizza il nome del client nelle sue liste, invia un messaggio di benvenuto al client e notifica a tutti gli altri utenti connessi che un nuovo utente si è unito alla chat.

Successivamente, entra in un ciclo che riceve messaggi dalla socket del client. Per ciascun messaggio ricevuto, verifica se è uguale a "{quit}". Se il messaggio non è "{quit}", viene inviato in broadcast a tutti gli altri utenti. Se invece è "{quit}", avvia la procedura di rimozione del client tramite il metodo appropriato, descritto nella sezione "Rimozione Dei Client" della relazione.

In questo metodo vengono gestiti errori come ConnectionResetError, ConnectionAbortedError e UnicodeDecodeError

L'errore ConnectionResetError in questo caso è gestito per affrontare la situazione in cui un processo client viene terminato in modo "sporco", e cioè quando viene disconnesso senza avvertire il server con il messaggio “quit” nella chat.

In modo simile ConnectionAbortedError è gestito per affrontare una chiusura improvvisa del client

Rimozione Dei Client

Il metodo delete_client si occupa della disconnessione di un client.

Ogni volta che un Server disconnette un Client, prima invia un messaggio di "quit" al Client che sta per essere disconnesso. Questo permette al Client di rilevare la richiesta di disconnessione e di disconnettersi autonomamente. Successivamente, il Server rimuove il Client e il suo nome dai suoi dizionari e invia un messaggio a tutti gli utenti rimanenti per notificare che il client ha lasciato la chat.Viene anche fatta una stampa dei nomi degli utenti ancora connessi al server

Inoltre è gestito l'errore che si verifica quando si tenta di inviare un messaggio su un socket già chiuso, stampando un semplice messaggio sul terminale per indicare che l'operazione non è riuscita.

Messaggi In Broadcast

Il metodo broadcast scorre la lista dei Client connessi al server e invia un messaggio a ciascuno ,concatenando un prefisso opzionale al messaggio. Se si verifica un errore durante l'invio del messaggio, viene gestito e stampato un messaggio di errore.

In questo caso il controllo di errore è gestito da OSError e da BrokenPipeError

CustomExceptions

La classe CustomExceptions è progettata per gestire una serie di errori specifici che possono verificarsi durante le operazioni di rete e di comunicazione tra un client e un server. Ogni messaggio di errore predefinito corrisponde a un tipo specifico di problema. Di seguito viene fornita una descrizione dettagliata per ciascun errore gestito.

CONNECTION_REFUSED_ERROR

Cosa è: Un'eccezione che potrebbe essere sollevata quando una connessione TCP viene rifiutata dal server.
Quando viene sollevata: Quando il client tenta di connettersi a un server che non è in ascolto sulla porta specificata o il server rifiuta esplicitamente la connessione.
A cosa serve: Permette di gestire situazioni in cui il server non è disponibile o non accetta connessioni, informando l'utente o l'applicazione che il tentativo di connessione non ha avuto successo.

OS_ERROR

Cosa è: Una classe generica di eccezioni per gli errori del sistema operativo.
Quando viene sollevata: Quando si verificano errori durante le operazioni di sistema, come accesso a file, creazione di socket, binding, ecc.
A cosa serve: Fornisce un modo per catturare e gestire errori generici del sistema operativo in modo che l'applicazione possa rispondere in modo appropriato, ad esempio loggando l'errore o tentando di ripristinare l'operazione.

GENERAL_CONNECTION_ERROR

Cosa è: Un'eccezione generica per errori di connessione.
Quando viene sollevata: Quando si verifica un errore non specificato durante una connessione, che non rientra in altre categorie più specifiche.

A cosa serve: Gestisce errori di connessione imprevisti o non categorizzati, permettendo di avere un fallback generico per problemi di rete.

CONNECTION_RESET_ERROR

Cosa è: Un'eccezione sollevata quando una connessione viene resettata da una delle parti.
Quando viene sollevata: Quando una delle estremità di una connessione chiude in modo anomalo, inviando un pacchetto TCP RST.
A cosa serve: Permette di rilevare e gestire situazioni in cui la connessione è stata interrotta improvvisamente dal peer, informando l'applicazione della necessità di ripristinare o chiudere correttamente la connessione.

RECEIVE_OS_ERROR

Cosa è: Un'eccezione per errori del sistema operativo specifici durante la ricezione di dati.
Quando viene sollevata: Quando si verificano errori del sistema operativo durante l'operazione di ricezione di dati su una connessione.
A cosa serve: Gestisce errori specifici del sistema operativo durante la ricezione, consentendo di eseguire azioni correttive come riprovare la ricezione o chiudere la connessione.

GENERAL_RECEIVE_ERROR

Cosa è: Un'eccezione generica per errori di ricezione.
Quando viene sollevata: Quando si verificano errori durante la ricezione di dati che non rientrano in altre categorie più specifiche.
A cosa serve: Fornisce una gestione generica degli errori di ricezione, permettendo di loggare o reagire a problemi di ricezione non specifici.

BROKEN_PIPE_ERROR

Cosa è: Un'eccezione sollevata quando si tenta di scrivere su una connessione chiusa.
Quando viene sollevata: Quando si tenta di inviare dati su un socket che è stato chiuso dall'altra estremità.
A cosa serve: Consente di gestire situazioni in cui l'invio di dati non è possibile perché il socket è chiuso, permettendo di pulire le risorse o notificare l'utente dell'interruzione della connessione.

GENERAL_SEND_ERROR

Cosa è: Un'eccezione generica per errori di invio.
Quando viene sollevata: Quando si verificano errori durante l'invio di dati che non rientrano in altre categorie più specifiche.
A cosa serve: Fornisce una gestione generica degli errori di invio, permettendo di loggare o reagire a problemi di invio non specifici.

CONNECTION_ABORTED_ERROR

Cosa è: Un'eccezione sollevata quando una connessione viene abortita.
Quando viene sollevata: Quando una connessione viene chiusa in modo imprevisto e involontario, spesso a causa di problemi di rete o interventi di dispositivi intermedi.
A cosa serve: Permette di rilevare e gestire situazioni in cui la connessione è stata chiusa inaspettatamente, informando l'applicazione della necessità di ripristinare o chiudere correttamente la connessione.

UNICODE_DECODE_ERROR

Cosa è: Un'eccezione sollevata quando si verifica un errore durante la decodifica di una stringa Unicode.

Quando viene sollevata: Quando i dati ricevuti non possono essere decodificati correttamente in una stringa Unicode a causa di una codifica non valida.
A cosa serve: Gestisce errori di decodifica Unicode, permettendo di loggare l'errore o informare l'utente di problemi con i dati ricevuti.