Sistema di Chat Client-Server

Lorenzo Bergami

Maggio 2024

1 Introduzione

Questo elaborato descrive la realizzazione di una chat room basata sul modello client-server, in cui più client possono connettersi a un server per scambiare messaggi in tempo reale. L'obiettivo è quello di esplorare e applicare i concetti di rete, come il modello client-server e i socket, per creare un sistema di comunicazione semplice ed efficace.

1.1 Modello Client-Server

Il modello client-server è un'architettura di rete in cui il server fornisce risorse o servizi e i client li richiedono. In questa implementazione, il server gestisce le connessioni dei client e smista i messaggi inviati da un client a tutti gli altri client connessi.

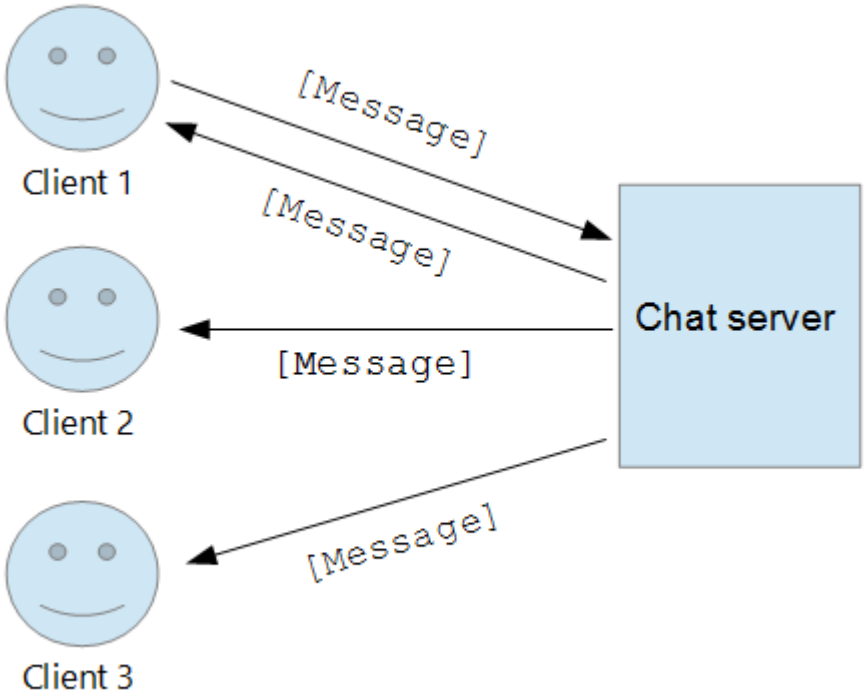

Figure 1: Client-Server Architecture

1.2 Socket

Un socket è un endpoint per la comunicazione tra due macchine. Utilizziamo i socket per consentire la comunicazione tra client e server sulla rete. Nel nostro caso, abbiamo usato i socket di tipo TCP della famiglia `AF_INET`, che garantiscono una trasmissione affidabile e ordinata dei dati.

1.3 Indirizzo e Porta

Un indirizzo IP identifica un dispositivo sulla rete, mentre una porta identifica un'applicazione specifica in esecuzione sul dispositivo. Nel nostro esempio, il server è in ascolto sull'indirizzo `localhost` (127.0.0.1) e sulla porta `8080`.

2 Funzionamento del Sistema

2.1 Server

Apertura del Server Il server viene avviato eseguendo lo script `chat__server.py`. All'avvio, il server crea un socket, lo associa a un indirizzo IP e a una porta specifica (`8080`), e inizia ad ascoltare le connessioni in arrivo.

Utilizzo del Server Quando un client si connette, il server accetta la connessione e crea un nuovo thread per gestire la comunicazione con quel client. Questo permette al server di gestire più connessioni contemporaneamente.

La funzione `handle_client` riceve i messaggi dal client e li smista agli altri client connessi tramite la funzione `broadcast`.

Chiusura del Server Il server rimane in esecuzione finché non viene terminato manualmente. Tutti i client connessi verranno disconnessi quando il server si chiude.

2.2 Client

Apertura del Client Il client viene avviato eseguendo lo script `chat__client.py`. All'avvio, il client crea un socket e si connette al server specificando l'indirizzo IP da terminale; la porta del server è impostata di default a 8080.

Utilizzo del Client Una volta connesso, il client avvia un thread per ricevere i messaggi dal server e contemporaneamente permette all'utente di inviare messaggi. La funzione `receive_messages` gestisce la ricezione dei messaggi, mentre la funzione `send_messages` legge l'input dell'utente e invia i messaggi al server. L'utente interagisce col programma tramite una GUI sviluppata con la libreria Tkinter: consiste molto semplicemente di un'area di visualizzazione dei messaggi, navigabile tramite scrollbar, e un box per inserimento di nuovi messagi, da inviare con l'apposito bottone `SEND`.

Chiusura del Client Il client rimane in esecuzione finché l'utente non invia un messaggio con scritto `{quit}` oppure termina manualmente l'esecuzione del programma (tramite la X rossa in alto oppure chiudendo il terminale). Quando la connessione viene chiusa, il thread di ricezione termina.

3 Requisiti per Eseguire il Codice

Per eseguire il codice del server e del client, sono necessari i seguenti requisiti:

1. Python 3: Assicurarsi di avere Python 3 installato sul proprio sistema.

2. Librerie Standard: Le librerie utilizzate (`socket`, `threading`) sono parte della libreria standard di Python, quindi non è necessario installare pacchetti aggiuntivi.

3.1 Considerazioni Aggiuntive

3.2 Threading

L'uso del threading permette la gestione simultanea di più connessioni e la comunicazione bidirezionale tra client e server. Questo approccio migliora l'efficienza e l'usabilità del sistema, permettendo agli utenti di inviare e ricevere messaggi contemporaneamente.

3.3 Possibili Estensioni

Questo progetto fornisce una base solida per ulteriori estensioni, come l'aggiunta di funzionalità di autenticazione degli utenti o crittografia dei messaggi per migliorare la sicurezza. Altre possibili estensioni includono la gestione dei nomi utente e la creazione di stanze di chat multiple.

3.4 Error Handling

L'implementazione include handling degli errori per gestire la disconnessione dei client e altri problemi di comunicazione. Tuttavia, per un'applicazione più robusta, si potrebbe migliorare la gestione degli errori e implementare ad esempio meccanismi di riconnessione automatica.