Progetto Reti: Realizzazione di una Chat Server-Client con Socket TCP

Lorenzo Canocchi, Emanuele Polini, Christian Falzetta

Aprile 2025

1 Introduzione

Nel contesto delle reti di calcolatori, i socket TCP sono strumenti fondamentali per la comunicazione tra sistemi remoti. Questo progetto, sviluppato in linguaggio C, ha come obiettivo la realizzazione di una chat server-client che utilizza i socket TCP per la comunicazione. La connessione è persistente, con il client e il server che si scambiano messaggi finché uno dei due non invia il comando `exit` per terminare la conversazione. Il sistema consente l'interazione tra un client e un server in tempo reale.

2 Descrizione

Il programma è composto da due parti principali: il client e il server. Entrambi utilizzano il protocollo TCP per trasmettere i dati e interagiscono in modalità chat. Il client invia messaggi al server, che risponde con un messaggio di ritorno. La comunicazione prosegue finché uno dei due invia il comando `exit`, momento in cui la connessione viene chiusa. Il server gestisce una singola connessione con il client e risponde ad ogni messaggio ricevuto.

3 Organizzazione

Il progetto è suddiviso in due file principali, uno per il client e uno per il server. Ogni file contiene il codice necessario per la gestione delle connessioni, l'invio e la ricezione dei messaggi, e la gestione della chiusura delle connessioni.

3.1 File di codice sorgente

- `client.c`: Implementa il programma client, che stabilisce la connessione con il server, invia e riceve i messaggi, e gestisce il ciclo di comunicazione con il server.

- `server.c`: Implementa il programma server, che attende una connessione con il client, riceve e invia i messaggi, e gestisce la chiusura della connessione.

4 Funzioni e Procedure

In questa sezione vengono descritte le principali funzioni e procedure utilizzate nei programmi client e server. Le descrizioni delle funzioni sono ispirate dalla documentazione e dagli esempi di GeeksforGeeks.

4.1 Client

- `createSocket(int *fd)`: Questa funzione crea una socket TCP per il client. Una socket è necessaria per stabilire una comunicazione tra il client e il server. La funzione accetta un puntatore al descrittore del file, che rappresenta la connessione della socket. La creazione della socket è realizzata tramite la chiamata di sistema `socket()`.

- `connection(int fd, char addrsIP[])`: La funzione stabilisce la connessione con il server. Prende come parametro il descrittore della socket `fd` e l'indirizzo IP del server `addrsIP`. La connessione viene effettuata tramite la chiamata di sistema `connect()`, che stabilisce una connessione TCP con il server, utilizzando l'indirizzo e la porta forniti.

- `comunicateUser(int fd)`: Questa funzione gestisce la comunicazione iniziale con il server per lo scambio dei nomi utente. Il client invia il proprio nome utente al server, che lo riceve e lo utilizza per identificare il client durante la conversazione.

- `comunicate(int fd)`: Gestisce l'invio e la ricezione dei messaggi tra il client e il server. Questa funzione permette al client di inviare messaggi al server e ricevere risposte. Ogni messaggio viene letto dall'input dell'utente e inviato al server, mentre la risposta del server viene visualizzata nel terminale del client. Se il messaggio inviato è `exit`, la connessione viene chiusa e il programma termina.

4.2 Server

- `createSocket(int *fd)`: La funzione crea una socket TCP per il server. La socket viene utilizzata per accettare connessioni in ingresso da parte dei client. La funzione utilizza la chiamata di sistema `socket()` per creare la socket e `bind()` per legarla a un indirizzo IP e una porta.

- `binding(int *fd)`: Questa funzione associa la socket creata ad una specifica porta e indirizzo IP del server. Viene utilizzata la funzione `bind()` per associare la socket alla porta di ascolto del server.

- `listening(int *fd)`: La funzione mette la socket in modalità di ascolto, pronta a ricevere connessioni da parte dei client. La funzione `listen()` è utilizzata per mettere la socket in modalità passiva e attendere che i client effettuino una connessione.

- `accepting(int *fd1, int *fd2)`: Accetta una connessione in ingresso da parte di un client. La funzione `accept()` stabilisce una connessione tra il server e il client, creando una nuova socket dedicata alla comunicazione con il client. Questa funzione è fondamentale per la gestione delle connessioni multiple.

- `comunicateUser(int fd)`: Gestisce l'invio e la ricezione degli username tra il server e il client. Ogni client invia il proprio nome utente, che viene poi visualizzato dal server per identificare l'utente.

- `comunicate(int fd)`: Gestisce la conversazione tra il server e i client. Il server riceve i messaggi dai client e risponde con un messaggio di conferma o con un altro messaggio di ritorno. La funzione termina quando uno dei due invia il comando `exit`.

5 Osservazioni e Conclusioni

Il progetto ha permesso di comprendere il funzionamento di una comunicazione di rete client-server utilizzando i socket TCP. La principale difficoltà riscontrata è stata la gestione della comunicazione in modo sincrono, poiché entrambi i programmi (client e server) devono essere progettati in modo che si aspettino correttamente i messaggi. Questo è stato realizzato implementando un ciclo di lettura e scrittura dei messaggi, gestendo la chiusura della connessione quando viene inviato il comando `exit`.

Un possibile miglioramento del progetto sarebbe l'implementazione di un sistema di gestione di più client contemporaneamente, utilizzando thread o

funzioni di selezione multiple per la gestione delle connessioni in modo più efficiente. Inoltre, l'introduzione di un sistema di crittografia o autenticazione dei messaggi potrebbe incrementare la sicurezza della comunicazione.

In conclusione, il progetto ha raggiunto gli obiettivi prefissati, dimostrando in modo pratico come funziona una connessione di rete di tipo client-server e come è possibile scambiare dati tra i due sistemi in un contesto di chat server-client.