Chatroom

Progetto di Programmazione di reti

Francesco Bittasi: 0001071026

Introduzione

Il progetto si pone come obiettivo quello di realizzare un'applicazione di chat di tipo *client-server* che permetta a più utenti di comunicare tra di loro in una chatroom condivisa.
I vari utenti si possono connettere al server centrale tramite connessioni client-server usando il protocollo TCP, scelto per l'affidabilità dei dati trasmessi, dato che l'obiettivo non era tanto l'avere estrema rapidità nella trasmissione dei messaggi, quanto più affidabilità.
Il server svolge il ruolo di gestore della stanza: riceve i messaggi inviati dai singoli client e li ritrasmette a tutti gli altri utenti connessi garantendo il corretto funzionamento della comunicazione.
Il server inoltre informa tutti gli utenti di eventuali ingressi e uscite dalla stanza, sia che queste siano volute che per motivi di caduta di connessione.
Nel caso di caduta di comunicazione col server il client si deve dimostrare abile nell'intercettare il problema e notificare all'utente della terminazione della connessione.
Per permettere la ricezione e l'invio contemporaneo dei messaggi sia il client che il server operano su più Thread, creando quindi una chat dinamica che non attenda l'intervento manuale degli utenti per ricevere aggiornamenti.
L'applicativo dotato di interfaccia grafica deve permettere sia la creazione che l'accesso ad una chat per facilitare l'uso del sistema senza bisogno di conoscenze preliminari sull'uso di comandi e terminali; nonostante l'uso del sistema tramite terminale sia supportato.

Server

Il server si occupa di gestire la stanza vera e propria: resta in ascolto di richieste di connessione alla stanza su un thread dedicato chiamato "Thread di accettazione", mentre per ogni client connesso tiene attivo un thread specifico chiamato "Client manager" che si occupa di ricevere i messaggi inviati dal client e rigirarli a tutti gli altri utenti connessi.

Inizializzazione

`start_server()`
La creazione del server si occupa di creare il socket e di associarlo alla porta di default scelta '53000'.

Essendo il numero di porta fisso, ogni host può gestire solo una stanza in ogni istante. Se si tenta dunque di avviare un nuovo server quando ce n'è uno già attivo questo ci verrà impedito.

Creato il socket con successo, lo script notifica il proprio indirizzo e avvia il Thread di accettazione.

Thread di accettazione

`accept_connections()`
Fintanto che il server è attivo il Thread di accettazione resta in ascolto di possibili richieste di connessione.
Ricevuta una richiesta la accetta, aggiunge il client alla lista degli utenti connessi e avvia un "Client manager" dedicato.

Client manager

`client_manager()`
Il client manager si distingue in due fasi:

1. Inizializzazione dell'utente
2. Ascolto e invio dei messaggi

Durante la fase di inizializzazione il server attende di ricevere il nome dell'utente appena connesso prima di aggiungerlo all'elenco dei nomi e di inviargli un messaggio di benvenuto.

- Nel caso il nome non arrivi entro un tempo di Timeout o se è riservato al server allora gli verrà assegnato un nome utente di default "USR".
- Nel caso il nome utente sia già usato da qualcuno allora al nome verrà aggiunto un numero per distinguerlo dagli utenti già connessi

Inizializzato l'utente, finché il server è attivo e l'utente resta connesso, il server si mette in ascolto dei possibili messaggi che questo client potrebbe inviare.
Alla ricezione di un messaggio se questo è un comando di terminazione della comunicazione allora il server si occupa di terminare il collegamento con tale client, altrimenti rigira il messaggio a tutti gli altri client nella stanza tramite il metodo `send_message_toAll()`.

Chiusura della connessione

`closeConnection()`
Nel caso di ricezione di un comando di terminazione di connessione da un client, nel caso di errore in ricezione o invio di messaggi ad un client allora il server procederà a terminare la connessione col client problematico.
Questo consiste nella chiusura del socket e nell'eliminazione dell'utente dalle liste degli utenti attivi.

Chiusura del server

`closeServer()`
Nel caso di chiusura del server (tramite comando Ctrl+C se da terminale, o per chiusura

dell'applicativo di hosting) il server procederà a terminare il Thread di accettazione e a terminare le connessioni con tutti i client nella stanza.

Ogni volta che un client si disconnette se la stanza resta vuota il server in automatico termina la propria esecuzione.

Client

Il client identifica un utente singolo connesso alla stanza con un proprio nome.
Per permettere la ricezione continua di messaggi in tempo reale il client ha un thread dedicato per la ricezione di messaggi dal server e, nel caso di desiderasse inviare messaggi, c'è bisogno che sia un altro thread a richiamare la funzione di invio del messaggio.

Apertura della connessione

`connect()`
La connessione richiede un indirizzo e un nome per il client (se non specificati di default saranno l'indirizzo dell'interfaccia di loopback e "USR"); stabilita con successo la connessione col server il client avvia il ricevitore e restituisce l'avvenuto successo della connessione.

Ricevitore

`receiver()`
Il ricevitore è un thread dedicato al solo ascolto dei messaggi: se riceve un messaggio lo notifica all' "ascoltatore".

L'ascoltatore è un oggetto che deve essere assegnato al client, questo deve implementare i metodi `.updateMessages(msg)` e `.closedConnection()`.
Nel caso l'ascoltatore non venga impostato invocando il metodo `addListener(new_listender)` il client procede con la stampa a video dei messaggi ricevuti, altrimenti rigira a questo oggetto ascoltatore il contenuto dei messaggi.

Il ricevitore non resta sempre bloccato in ascolto di messaggi e periodicamente esce dall'attesa per verificare lo stato del server e chiudere la comunicazione in caso di terminazione.

Invio messaggi

Invocando il metodo `send_message(msg)` il client invia al server il messaggio contenuto in msg in formato "utf8".

Disconnessione

`closeConnection()`
Nel caso di invio del comando di terminazione di connessione o nel caso di errori in invio o ricezione di messaggi il client procede con la terminazione delle comunicazione e comunica l'oggetto "ascoltatore" di tale chiusura.

Applicazione

L'applicazione deve rendere disponibili tutte le funzionalità dei due script tramite una semplice interfaccia grafica

Home page

Nella home page possiamo impostare un nome utente e poi possiamo scegliere se hostare una stanza o connetterci ad una.

Nel caso decidessimo di connetterci ad una stanza ci apparirà la schermata per inserire Indirizzo IPv4 e numero di Porta del server a cui ci vogliamo connettere. Se lasciati vuoti vengono usati i valori di default `localhost:53000`.

Nel caso invece decidessimo di hostare il server allora ci si apre direttamente la pagina di chat e come primo messaggio riceveremo dal server l'indirizzo del server lanciato.
Infatti l'applicativo si occupa di avviare il server e poi di connettersi in locale come un normale client.

Se un client termina l'applicazione allora verrà notificato il server della chiusura della connessione e poi verrà terminato il programma.

Sei invece a terminare l'applicazione è l'utente che sta facendo da host allora verrà prima chiuso il server, poi viene terminata la sua connessione locale ed infine viene chiusa l'applicazione.
Nel caso di chiusura del server, gli altri utenti connessi riceveranno un messaggio informativo con l'istruzione di chiudere l'applicazione per uscire, e sono impossibilitati all'invio di ulteriori messaggi.

Uso tramite terminale

Se si desidera hostare una chatroom senza per forza prendervene parte questo si può fare semplicemente lanciando lo script `ChatServer.py` da terminale.
L'esecuzione dello script lancerà il server e lo porrà in ascolto sulla porta di default, da terminale sarà possibile visualizzare gli ingressi e le uscite dalla stanza nonché verificarne la corretta chiusura.
Il server avviato da terminale è terminabile tramite il comando Ctrl+C.

Lo script del client `ChatClient.py` supporta l'esecuzione da terminale come "ascoltatore" dei messaggi per un server avviato localmente. Anche lui può essere terminato tramite il comando Ctrl+C.