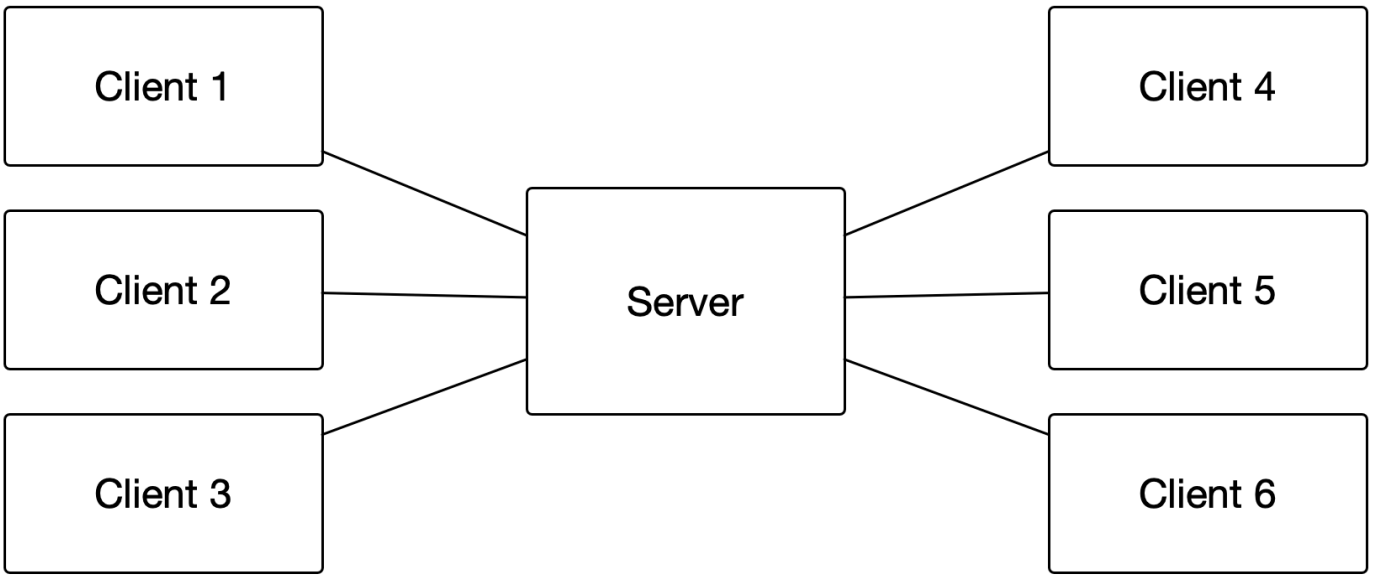
PROGETTO
PROGRAMMAZIONE DI RETI

Traccia 1

Riccardo Balzani



Obiettivo del Progetto

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.

Architettura del sistema

Il sistema è costituito da due componenti principali: il **server** e il **client.** Il server è responsabile di accettare le connessioni dai client, gestire i messaggi e instradarli agli altri client connessi. Il client, d'altra parte, è responsabile di stabilire una connessione con il server, inviare e ricevere messaggi. La **dimensione massima** dei messaggi inviabili e ricevibili è di 1 Kb.

Implementazione del server

Il server è stato implementato utilizzando il linguaggio di programmazione **Python** e la libreria **socket** per la comunicazione di rete. È in grado di **gestire più client** contemporaneamente grazie all'utilizzo di **thread.** Il server utilizza una **coda di backlog** per gestire le connessioni in arrivo e un meccanismo di lock per

garantire l'accesso sicuro alla lista dei client connessi. Ogni qualvolta che viene ricevuto un messaggio da parte di un client esso viene mandato in **broadcast** aggiungendo un header contenente il **nome** del client dal quale proviene. Se un client si disconnette, o ne viene terminata la connessione, esso viene rimosso dalla **client_list** e il corrispondente thread termina naturalmente. Tutti i messaggi arrivati e i messaggi di **gestione del server** vengono visualizzati nel terminale corrispondente. E' possibile eseguire lo **shutdown** del server digitando nel terminale la stringa **'close'.**

Implementazione del client

Il client è anch'esso implementato in **Python** utilizzando la libreria **socket.** È in grado di stabilire una connessione con il server e inviare **messaggi testuali.** Il client offre la possibilità di scrivere il messaggio direttamente da un'apposita **GUI.** Una volta lanciato il programma esso chiederà di inserire il proprio **nome utente.** Il nome non può coincidere con **'Server'.** Digitando la stringa **'exit'** la connessione viene interrotta e il programma terminato. Il client è in grado di **ricevere messaggi** provenienti dal server. I messaggi ricevuti rappresentano stringhe inviate da altri client connessi allo stesso server e quindi vengono stampati a schermo.

Gestione delle eccezioni

Sia il server che il client sono progettati per gestire le eccezioni in modo robusto. In caso di errori di connessione o di trasmissione, entrambi i componenti sono in grado di chiudere la connessione in modo sicuro e di terminare l'esecuzione in modo ordinato. In base al tipo di eccezione incontrata sono stati previsti **differenti meccanismi** di gestione degli errori.

BrokenPipeError: il destinatario non è raggiungibile, la connessione viene **interrotta.**

ConnectionRefusedError: il server ha rifiutato la connessione, si effettua un **nuovo tentativo** che avviene dopo un **delay** di 0.1 secondi. Il massimo numero di tentativi possibili è impostato a tre.

Exception: è avvenuta un'eccezione generica, in base al contesto in cui essa viene catturata la gestione è differente. In ogni caso la connessione viene **ritentata oppure interrotta.**

Per ogni eccezione che si verifica il messaggio viene stampato a schermo.

Per evitare bug, dovuti al messaggio speciale che il server invia ai client in caso di chiusura, il client non può assumere il nome **'Server'.**

Gestione della Concorrenza

All'interno degli script si fa ampio utilizzo della concorrenza tramite la libreria **threading.**

All'interno di **Server.py** si hanno tre tipi di thread:

1) **client_thread:** thread che viene creato per ogni client che si connette al server.
2) **close_server_thread:** thread in ascolto sul terminale. Una volta digitato **'close'** invocherà la funzione **'close_server'.**
3) **main_thread:** thread che **lancia il programma** e si occupa della ricezione dei messaggi.

L'unica variabile condivisa è **'client_list',** essa viene protetta dalla lock **'client_list_lock'.**

All'interno di **Client.py** si hanno tre tipi di thread:

1) **receiving_thread:** thread che si occupa di ricevere i messaggi inviati dal server.
2) **gui_thred:** thread che si occupa di gestire la **GUI.**
3) **main_thread:** thread che lancia il programma.

Le variabili condivise sono **'message_list'** e **'kill_thread',** protette rispettivamente da **'receiving_thread_lock'** e **'kill_thread_lock'.**

Utilizzo del codice

Il codice può essere utilizzato attraverso un terminale di tipo unix-like.

1) Aprire una **shell** e spostarsi nella directory contenente il file:

Server.py

2) Lanciare lo script contenuto in **Server.py** utilizzando il comando:

(a) **python Server.py**

3) Aprire una **shell** differente e spostarsi nella directory contenente il file:

Client.py

4) Lanciare lo script contenuto in Client.py utilizzando il comando:

(b) **python Client.py**

5) Ora è possibile scrivere messaggi all'interno della finestra **GUI** corrispondente a **Client.py.** Tali messaggi verranno inviati al server e successivamente a tutti i client connessi ad esso.
6) Una volta che si vuole effettuare la **disconnessione** del client dal server è sufficiente digitare la stringa:

(c) **‘exit’**

all’interno della shell in cui è stato lanciato lo script **Client.py.**

7) Se si vuole connettere nuovamente al server è sufficiente digitare nuovamente il **comando (b).**
8) Se si vogliono connettere nuovi client al server occorre effettuare nuovamente i punti **3)** e **4).**

Nota bene: *Fino a quando almeno un client risulta connesso al server è necessario mantenere lo script* ***Server.py*** *in esecuzione.*

Nel momento in cui si vuole chiudere il server sarà sufficiente **terminare l’esecuzione** dello script avendo preventivamente disconnesso tutti i client dal server **6).**

Viene anche data la possibilità di **chiudere brutalmente** il server. Ciò avviene digitando il comando:

(d) **‘close’**

all’interno della shell in cui **Server.py** è in esecuzione.

Nel momento in cui viene effettuata questa operazione i client verranno **disconnessi.** I client verranno avvisati con un opportuno messaggio nell’interfaccia grafica.