Relazione Progetto
Programmazione di Reti - @unibo

Marco Marrelli - 0001079192

Giugno 2024

Indice

Introduzione

Il progetto si è posto come obiettivo la realizzazione di un'applicazione chat (client-server) che permetta a più Client di comunicare all'interno di un Server. Il Server recepisce i messaggi inviati da un qualsiasi Client e li trasmette a tutti i restanti Client connessi (broadcast). Oltre a ricevere tutti i messaggi, il Server tiene in considerazione tutti gli ingressi e le uscite (sia volontarie che per errori di ogni natura) dei vari Client.

La connessione avviene tramite protocollo TCP (Transmission Control Protocol), un protocollo di livello transport usato per l'affidabilità nel condividere informazioni tra un mittente e un destinatario. La gestione delle varie funzionalità (ricezione messaggi, accettazione Clients, controllo connessioni, ecc...) avviene tramite Threads, rendendo tutto il processo dinamico e in continuo aggiornamento automatico.

Requisiti

Linguaggio di Compilazione:

- Python 3.x

Librerie Python:

- **Socket** - per il networking
- **Threading** - per i threads
- **Tkinter** - per la GUI
- **Logging** - per logging delle informazioni ed errori
- **Time** - per la funzione sleep

Funzionamento del Client

Il Client si connette al Server, permettendo all'utente di inviare e ricevere messaggi, il tutto all'interno di un'interfaccia grafica. Ogni Client ha la sua socket TCP/IP, utilizzata per la connessione con il server.

Quando la classe Client viene istanziata, per prima cosa configura il logging con un formato di messaggio specifico. Poi crea una sua socket TCP/IP personale, con la quale si connetterà al Server, utilizzando l'indirizzo e la porta specificati nella pagina di configurazione. Crea l'interfaccia grafica utilizzando la libreria Tkinter, includendo un'area per visualizzare i messaggi, un campo testuale per scrivere i messaggi da inviare tramite ¡Invio¿ o un pulsante apposito. Imposta il comportamento per la chiusura della finestra e avvia un thread per gestire la ricezione dei messaggi dal server.

Il metodo manageMessage() crea un ciclo infinito per riceve messaggi dal Server. Se un messaggio viene ricevuto, viene visualizzato nell'area dei messaggi, mentre se la connessione viene interrotta, la socket del Client viene chiusa.

Il metodo displayMessage() permette all'utente di Visualizzare un messaggio ricevuto nell'area dei messaggi della GUI.

Il metodo sendMessage() invia il messaggio (non vuoto) inserito nella casella testuale al Server, codificandolo secondo l'encoding specificato nella configurazione (di default 'utf8'). Dopo l'invio, il campo di input viene svuotato.

Il metodo closeConnection() chiude la connessione del Client al Server, terminando l'esecuzione della GUI e dei thread sottostanti.

Il metodo (statico) startClientGUI() crea un Client, avviandone i suoi processi descritti precedentemente, tra cui la sua GUI e i metodi per il ricevimento/invio dei messaggi.

Funzionamento del Server

Il Server è il cuore del nostro programma, provvedendo allo scambio dei messagi dei vari Clients. La gestione delle connessioni avviene tramite una lista di tuple, oggetti formati da una socket e il suo indirizzo relativo.

Quando la classe Server viene istanziata, per prima cosa configura il logging con un formato di messaggio specifico. Poi crea una socket TCP/IP per il Server, assegnandole un indirizzo (localhost, 127.0.0.1) e una porta, per poi infine impostare il numero massimo di connessioni simultanee. Tutti questi parametri sono impostabili nel file di configurazione.

Se le operazioni precedenti avvengono con successo, comincia un ciclo infinito di accettazione Client. Quando un Client vuole collegarsi, esso viene aggiunto alla lista di connessioni. Il Server dovrà poi inviare un messaggio di notifica dell'entrata a tutti i restanti Client (broadcast) e avvierà un nuovo thread per gestire la comunicazione con il Client. Le eccezioni saranno sempre gestite, chiudendo le connessioni se necessario.

Il metodo manageClient() gestisce la comunicazione con ogni singolo Client. Riceverà messaggi dai Client in un ciclo infinito, inviandoli a tutti i Client connessi tramite broadcast. Se la connessione del Client viene interrotta, il Server lo rimuoverà dalla lista delle connessioni e conseguentemente verrà chiuso il processo del Client.

Il metodo broadcast() invia un messaggio a tutti i Client connessi, gestione vari errori di invio o di recezione. Nel caso si verificasse un errore, il Client che ha procurato l'errore verrà rimosso dalla lista delle connessioni.

Il metodo removeClient() rimuove un Client dalla lista delle connessioni, inviando poi in broadcast un messaggio di uscita. Nell'eventualità che il server non abbia più connessioni attive, la funzione chiuderebbe la socket del server.

Esecuzione del Programma

Prima di tutto, ci sono alcuni Settings da configurare all'interno del file 'config.py'.

- **SERVER_NAME** (string, default = 'Unibo Project Server')
  Definisce il nome del Server mostrato nell'interfaccia grafica.
- **SERVER_ADDRESS** (string, default = '127.0.0.1')
  Definisce l'indirizzo IP del Server.
- **SERVER_PORT** (integer, default = 12500)
  Definisce la porta del Server.
- **NUMBER_OF_CLIENTS** (integer, default = 3)
  Definisce il numero di Clients da creare all'interno del main.
- **MAX_NUMBER_OF_CLIENTS** (integer, default = 10)
  Definisce il numero di Clients massimi accettabili dal Server.
- **BUFFER_SIZE** (integer, default = 1024)
  Definisce la dimensione del buffer di un messaggio.
- **MESSAGE_ENCODING** (string, default = "utf8")
  Definisce la tipologia di codifica da utilizzare per i messaggi.
- **EXCEPTIONS_INFO** (boolean, default = False)
  Utilizzata per il logging, mostra il traceback completo di un errore.

Il programma potrà poi essere eseguito in due maniere diverse:

- Tramite lo script **'main.py'**
  Vengono inizialmente verificate e corrette alcune impostazioni di configurazione. Avviene poi l'avvio del server e dei client automaticamente, utilizzato il threading per permettere il funzionamento concorrente dei due componenti. Tutto questo tramite un solo comando di compilazione, 'python main.py'.

- Facendo partire lo script **'server.py'** e **'client.py'** separatamente.
  I file del Server e del Client però sono stati creati in maniera tale da permettere allo sviluppatore di far partire le due componenti in console separate (un solo thread Server ma più thread Client).
  Basterà avviare i due script:

  – 'python server.py'
    Crea il Server.

  – 'python client.py'
    Eseguibile più volte su varie consoles, crea i Client.

  Ovviamente, per il corretto funzionamento, andrà prima creato il Server e poi i Client.