Chat Client-Server

Relazione per il progetto
del corso di
Programmazione di Reti
A.A. 2023/24

Lorenzo Tordi 0001042969
Alex Frisoni 0001089191,
Lukasz Wojnicz 0001071295

14 maggio 2024

Indice

Capitolo 1

Introduzione

Il programma in Python permette di eseguire un server TCP e connettersi ad esso come client utilizzando i socket.

- Requisiti: Python 3.x
- Avviare il Server
  - Assicurarsi di avere Python installato sul proprio sistema. Aprire un terminale o prompt dei comandi. Navigare nella directory del programma utilizzando il comando cd. Eseguire il server eseguendo il file server.py con Python: `python server.py`
- Connessione come Client
  - Aprire un altro terminale o prompt dei comandi. Navigare nella stessa directory del programma. Eseguire il client eseguendo il file client.py con Python: `python client.py`

Capitolo 2

TCP-Client

È un client di chat che si connette a un server su una data porta e indirizzo IP utilizzando socket TCP/IP. Ecco come funziona:

- Importiamo i moduli `socket` e `threading`: il modulo `socket` viene utilizzato per la comunicazione di rete, mentre `threading` viene utilizzato per creare e gestire i thread.
- Definiamo un indirizzo IP e una porta del server a cui il client si connetterà. Nel nostro caso sono:
  - HOST = 127.0.0.1
  - PORT = 9090
- Definiamo due funzioni principali:
  - `receiveMsg(client-socket)`:Questa funzione è responsabile per la ricezione dei messaggi dal server. Viene eseguita in un loop continuo finché la connessione non viene chiusa. Se si verifica un errore di connessione, viene gestito in modo appropriato.

```python
# Funzione per ricevere i messaggi dal server
def receiveMsg(client_socket):
    try:
        while True:
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                print("Connessione al server chiusa.")
                break
            print(message)
    except ConnectionResetError:
        print("Connessione al server persa.")
    except ConnectionAbortedError:
        print("Connessione al server interrotta.")
    except Exception as e:
        print(f"Si è verificato un errore: {e}")
```

Figura 2.1: receiveMsg(client-socket)

– `sendMsg(client-socket)`:Questa funzione è responsabile per l'invio dei messaggi al server. Viene eseguita in un loop continuo, consentendo all'utente di inserire i messaggi da inviare.

```python
# Funzione per inviare un messaggio al server
def sendMsg(client_socket):
    while True:
        try:
            message = input()
            if message:
                client_socket.send(message.encode('utf-8'))
        except Exception as e:
            print(f"Messaggio di sistema: {e}")
            break
```

Figura 2.2: sendMsg(client-socket)

- All'interno di un ciclo `while True`, viene creato il socket del client e viene tentata la connessione al server utilizzando `client-socket.connect((HOST, PORT))`.Viene richiesto all'utente di inserire un username che verrà inviato al server. Se la connessione viene chiusa durante questa fase (ad esempio premendo Ctrl+C), il client esce dal programma.Se l'username è accettato dal server, il client avvia due thread:

– Un thread per ricevere i messaggi dal server utilizzando la funzione `receiveMsg`.
– Un thread per inviare i messaggi al server utilizzando la funzione `sendMsg`.

I due thread vengono avviati con il metodo `start()`. Il thread di invio viene atteso utilizzando il metodo `join()`. Questo fa sì che il programma principale rimanga in attesa finché il thread di invio non termina. Se l'utente interrompe il programma premendo Ctrl+C durante questa fase, il client chiude la connessione e esce dal programma. Se l'username è già in uso, il client chiude immediatamente la connessione. Se si verifica un errore durante la connessione al server, viene gestito in modo appropriato. Ad esempio, se la connessione viene rifiutata, il client esce dal programma.

```
# Creazione del socket del client
while True:
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((HOST, PORT))

        # Richiesta del nome utente
        try:
            username = input("Benvenuto! Inserisci il tuo username (Ctrl+C per chiudere la chat): ")
            client_socket.sendall(username.encode('utf-8'))
            # Ricezione della risposta del server
            response = client_socket.recv(1024).decode('utf-8')
            print(response)  # Stampa la risposta del server
        except KeyboardInterrupt:
            print("\nHai abbandonato la chat.")
            break


        if response != "Username già in uso. Per favore, prova un altro.":
            # Se l'username è accettato dal server, avvia i thread per ricevere e inviare messaggi
            receive_thread = threading.Thread(target=receiveMsg, args=(client_socket,))
            send_thread = threading.Thread(target=sendMsg, args=(client_socket,))

            # Avvio dei thread
            receive_thread.start()
            send_thread.start()

            try:
                # Attendo la terminazione del thread di invio
                send_thread.join()
            except KeyboardInterrupt:
                print("Hai abbandonato la chat.")
                client_socket.close()
                break
        else:
            # Se l'username è già in uso, chiudi la connessione
            client_socket.close()

    except ConnectionRefusedError:
        print("Connessione con il server rifiutata")
        break
```

Figura 2.3: Creazione del socket del client

Capitolo 3

TCP-Server

È un server di chat TCP utilizzando il modulo `socketserver`. Ecco come funziona:

- Importazione del modulo `socketserver`.
- Definizione di un dizionario vuoto `clients` per memorizzare i client connessi.
- Definizione di un indirizzo IP e una porta per il server.
- Definizione della funzione `broadcast-message(message)`, che invia un messaggio a tutti i client connessi. Itera attraverso tutti i valori nel dizionario `clients` (cioè tutte le connessioni dei client) e invia il messaggio a ciascuno di essi. Se si verifica un errore durante l'invio del messaggio a un client, viene stampato un messaggio di errore.

```
def broadcast_message(message):
    # Invia il messaggio a tutti i client connessi
    for client_conn in clients.values():
        try:
            client_conn.send(message.encode('utf-8'))
        except Exception as e:
            print(f"Errore nell'invio del messaggio a un cliente: {e}")
```

Figura 3.1: Broadcast-message

- Definizione di una classe `ChatRequestHandler` che eredita da `socketserver.BaseRequestH` Questa classe gestisce le richieste di connessione dei client. Quando viene ricevuta una nuova connessione, il metodo `handle()` viene eseguito.

- Il metodo `handle()` gestisce una nuova connessione client. In un loop infinito, riceve lo username inviato dal client, verifica se è già presente nel dizionario `clients`. Se lo username è già in uso, invia un messaggio di errore al client e continua a chiedere uno username valido. Se lo username è nuovo, aggiunge il client al dizionario `clients` e interrompe il loop.
- Dopo aver ricevuto uno username valido, invia un messaggio di benvenuto a tutti i client connessi e stampa un messaggio di notifica sulla console del server.
- Successivamente, gestisce la ricezione e l'invio dei messaggi tra i client. Se un client invia un messaggio, lo inoltra a tutti gli altri client connessi.
- Se si verifica un errore durante la gestione della connessione (ad esempio, se il client si disconnette), interrompe il loop, rimuove il client dal dizionario `clients`, invia un messaggio di uscita a tutti i client e stampa un messaggio di notifica sulla console del server.

```
class ChatRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):        # Gestisce una nuova connessione client
        while True:
            username = self.request.recv(1024).decode('utf-8').strip()  # Riceve lo username dal client
            if username in clients:
                # Se lo username è già in uso, invia un messaggio di errore al client:
                self.request.send("Username è già in uso. Riprova con un altro nome:".encode('utf-8'))
            else:
                clients[username] = self.request
                break

        try:
            broadcast_message(f"{username} si è unito alla chat.")  # Invia un messaggio a tutti i client
            print(f"{username} si è unito alla chat.")  # Stampa un messaggio sulla console del server
        except Exception as e:
            print(f"Errore nell'invio del messaggio di benvenuto: {e}")

        while True:
            try:
                message = self.request.recv(1024).decode('utf-8')  # Riceve un messaggio dal client
                if not message:
                    break
                else:
                    # Invia il messaggio a tutti i client
                    broadcast_message(f"{username}: {message}")
            except (ConnectionResetError, ConnectionAbortedError):
                break
            except Exception as e:
                print(f"Errore durante la gestione del messaggio: {e}")

        del clients[username]  # Rimuove la connessione client dal dizionario
        try:
            broadcast_message(f"{username} ha lasciato la chat.")  # Invia un messaggio a tutti i client
            print(f"{username} ha lasciato la chat.")  # Stampa un messaggio sulla console del server
        except Exception as e:
            print(f"Errore nell'invio del messaggio di uscita: {e}")
```

Figura 3.2: ChatRequestHandler

- Creazione di un'istanza di `socketserver.ThreadingTCPServer`, passando l'indirizzo IP e la porta del server, e la classe `ChatRequestHandler` come gestore delle richieste. Questa istanza avvierà il server e inizierà ad accettare le connessioni dei client. All'interno di un blocco `try`, viene chiamato `server.serve-forever()` per avviare il server in modo da ascoltare continuamente le richieste dei client. Il server resta in esecuzione indefinitamente fino a quando non viene interrotto da una KeyboardInterrupt (ad esempio, premendo Ctrl+C). In tal caso, viene eseguito il codice nel blocco `except KeyboardInterrupt` per chiudere correttamente il server. Se si verifica un altro tipo di eccezione durante l'esecuzione del server, viene gestita nel blocco `except Exception` stampando un messaggio di errore.

```
# Crea un server TCP e inizia a servire per sempre
server = socketserver.ThreadingTCPServer((HOST, PORT), ChatRequestHandler)
try:
    server.serve_forever()
except KeyboardInterrupt:
    print("Chiusura del server su richiesta dell'utente.")
    server.shutdown()
    server.server_close()
except Exception as e:
    print(f"Errore generale durante l'esecuzione del server: {e}")
```

Figura 3.3: Server TCP