Chat Client-Server

Simone Furcas

Giugno 2024

1 Introduzione

Il progetto implementa una chat asincrona utilizzando socket TCP in Python, con un'architettura client-server multithread. In questo documento verrà illustrato il funzionamento della componente client e di quella server, evidenziando il loro ruolo e le loro interazioni.

2 Architettura del Sistema

2.1 Componenti Principali

- **Server**: Accetta connessioni dai client, gestisce l'invio e la ricezione dei messaggi e ne permette la trasmissione a tutti gli utenti connessi.
- **Client**: Si connette al server, consente agli utenti di inviare e ricevere messaggi tramite un'interfaccia grafica e all'occorrenza di terminare la connessione tramite un apposito comando.

2.2 Moduli Pyhton Necessari

- **Socket**: utilizzato per la creazione del lato server.
- **Threading**: utilizzato per la gestione di una moltitudine di client simultaneamente, attraverso thread dedicati.
- **Tkinter**: utilizzato per l'interfaccia grafica a disposizione dell'utente.

3 Implementazione del Server

Il server è progettato per gestire connessioni simultanee da parte di più client utilizzando threading. Qui di seguito ne verranno analizzate le funzioni principali.

3.1 Connessione dei Client

```
1
2 def in_connections_acceptor():
3     while True:
4         client, client_address = SERVER.accept()
5         print("%s:%s joined the chat" % client_address)
6         client.send(bytes("Write your name and confirm with Send", "
    utf8"))
7         indirizzi[client] = client_address
8         Thread(target=client_manager, args=(client,)).start()
```

La funzione si mette in ascolto attraveso un ciclo "while True" aspettando di ricevere una richiesta di connessione da parte di un client. Alla ricezione l'indirizzo del client viene aggiunto ad un dizionario e un thread viene allocato per la gestione dell'utente.

3.2 Gestione di un Client

```
1
2 def client_manager(client):
3     name = client.recv(BUFSIZ).decode("utf8")
4     client.send(bytes("Write {exit} to quit", "utf8"))
5     broadcast(bytes(("%s joined" % name), "utf8"))
6     clients[client] = name
7     while True:
8         msg = client.recv(BUFSIZ)
9         if msg != bytes("{exit}", "utf8"):
10            broadcast(msg, name + ": ")
11        else:
12            client.close()
13            del clients[client]
14            broadcast(bytes("%s left the chat" % name, "utf8"))
15            break
16
17 def broadcast(msg, prefix=""):
18     for user in clients:
19         user.send(bytes(prefix, "utf8") + msg)
```

La funzione inizializza il client all'interno del server, dopodichè aspetta un input dall'utente. Arrivato l'input questo viene condiviso con gli altri utenti attraverso la funzione di broadcast. All'invio del codice di uscita (in questo caso `exit`) la funzione si occupa di chiudere il collegamento con il client e di cancellarlo dal dizionario apposito.

3.3 Corpo del programma: variabili e inizializzazione

```
clients = {}
indirizzi = {}

HOST = ""
PORT = 53000
BUFSIZ = 1024
ADDR = (HOST, PORT)

SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)

if __name__ == "__main__":
    SERVER.listen(10)
    print("Waiting for users...")
    THREAD = Thread(target=in_connections_acceptor)
    THREAD.start()
    THREAD.join()
    SERVER.close()
```

Dopo aver creato e inizializzato le variabili e aver disposto il socket e il bind del server, quest'ultimo viene posto in ascolto attraverso un thread contenente la funzione di connessione dei client vista in precedenza.

4 Implementazione del Client

Il client è progettato per fornire un'interfaccia grafica all'utente utilizzando `tkinter`. Permette agli utenti di inviare e ricevere messaggi in modo asincrono. Qui di seguito ne analizzeremo le funzioni più importanti.

4.1 Ricezione dei Messaggi

```
def receive():
    while True:
        try:
            msg = client_socket.recv(BUFSIZ).decode("utf8")
            msg_list.insert(tkt.END, msg)
        except OSError:
            break
```

La funzione si occupa appunto di ricevere i messaggi inviati dal server a cui si è collegati. Una volta ricevuto il messaggio dal client socket questo viene inserito nella message list della GUI così da poter essere visualizzato dall'utente.

4.2 Invio dei Messaggi

```
1
2 def send(event=None):
3     msg = my_msg.get()
4     my_msg.set("")
5     client_socket.send(bytes(msg, "utf8"))
6     if msg == "{exit}":
7         client_socket.close()
8         window.quit()
```

La funzione si occupa dell'invio dei messaggi ottenendo l'input utente dalla GUI a riga 3, controllando che non si tratti del codice di uscita e passandolo successivamente al client socket con il metodo send.

4.3 Corpo del programma: variabili e inizializzazione

```
1
2 HOST = input("Enter server host: ")
3 PORT = input("Enter port number: ")
4
5 if not PORT:
6     PORT = 53000
7 else:
8     PORT = int(PORT)
9
10 BUFSIZ = 1024
11 ADDR = (HOST, PORT)
12
13 client_socket = socket(AF_INET, SOCK_STREAM)
14 client_socket.connect(ADDR)
15
16 receive_thread = Thread(target=receive)
17 receive_thread.start()
18 tkt.mainloop()
```

Dopo aver creato e inizializzato le variabili, verrà aperto il client socket e creato un thread, chiamato receive thread, per la ricezione dei messaggi.

5 Conclusione

Questo progetto dimostra come implementare una chat asincrona multithread utilizzando socket TCP in Python. Potenziali miglioramenti potrebbero includere ad esempio, l'aggiunta di funzionalità come autenticazione degli utenti, messaggi privati e un'interfaccia utente più avanzata.