Sistema di chat client- server

Gabriele Fronzoni

14 luglio 2024

1 Client

Lato Client, vengono presi in input quelli che sono i valori dell'host e della porta sui quali connettersi. Viene poi inizializzato il socket e il client si connette al Server. Viene inoltre fatto partire il thread per la ricezione e gestione dei messaggi che provengono dal server. La funzione `receive` del client ha il compito di ricevere i messaggi provenienti dal server e poi li mostra nella finestra di dialogo che viene mostrata all'utente.

La funzione `send` ha il compito di mandare al server ciò che l'utente scrive. Nel caso in cui l'utente digiti il testo per uscire dal server, viene chiusa la finestra con l'utente e chiuso il socket del client.

Per la gestione della GUI viene utilizzata la libreria grafica *Tkinter*. Mediante questa libreria vengono create una finestra in cui vengono salvati i messaggi che vengono inviati dagli utenti sulla chat, viene data la possibilità di scrivere all'utente in un apposita casella di testo e gli viene fornito un comando per inviare i messaggi.

Nel momento in cui vengono chiamate le funzioni `send` e `receive`, vengono chiamati gli opportuni metodi della finestra Tkinter per poter ottenere il testo digitato dall'utente e per poter mostrare a schermo i messaggi della chat.

```python
def receive():
    while True:
        try:
            msg = client_socket.recv(BUFSIZE).decode("utf8")
            if not msg:
                break
            msg_list.insert(tkt.END, msg )
        except OSError:
            break

def send(event=None):
    msg = my_msg.get()
    my_msg.set("")
    client_socket.send(bytes(msg, "utf8"))
    if msg == "{quit}":
        window.quit()
        window.destroy()
        client_socket.close()


def on_closing(event=None):
    my_msg.set("{quit}")
    send()
```

Figura 1: Funzioni lato client

2 Server

Lato Server, viene settato come indirizzo host quello di loopback e viene scelto un valore della porta. Viene poi inizializzato il socket e legato all'indirizzo del server mediante la funzione socket.bind(). Viene inoltre inizializzato il thread per la gestione delle chiamate in entrata e viene chiamato il metodo .join() che aspetta che il thread chiamato termini i lavori prima di proseguire.

Nel momento in cui in client tenta di connettersi viene chiamata la funzione `accept-connection` la quale ha il compito di accettare il collegamento del client al server, registra l'indirizzo del client in un opportuno dizionario dei registri e fa partire il thread che gestisce la comunicazione con il client.

```python
def accept_connection():
    while True:
        client,client_adress = SERVER.accept()
        print("%s: %s has connected." % client_adress)
        indirizzi[client] = client_adress
        Thread(target=handle_client, args=(client,)).start()
```

Figura 2: Funzione accept-connection

Ogni client che si collega viene poi gestito mediante la funzione `handle-client`: la funzione inizialmente attenda che l'utente digiti il proprio nome utente. A questo viene poi mandato un messaggio di benvenuto e viene mandato un messaggio in broadcast a tutti gli altri utenti connessi al server per segnalare la connessione del nuovo utente. Viene inoltre registrato in un opportuno dizionario il nome utente e viene associato all'indirizzo di rete. Si entra poi in un ciclo nel quale il server si mette in ascolto di messaggi da parte del client. Se si tratta

di una richiesta di uscita da parte dell'utente, il server chiude la connessione con l'utente, cancella il relativo record dal dizionario e manda in broadcast un messaggio a tutti gli altri utenti connessi. Se invece si tratta di un messaggio qualsiasi, il server semplicemente lo manda in broadcast a tutti gli altri utenti connessi.

```
def handle_client(client):
    try:
        nome = client.recv(BUFSIZ).decode("utf8")
        benvenuto = "Welcome %s: if you want left the chat, write {quit} " % nome
        client.send(bytes(benvenuto,"utf8"))
        msg_br = "User %s has joined the chat " % nome
        broadcast(bytes(msg_br, "utf8"))
        clients[client] = nome
    except ConnectionResetError:
        print("Client has not logged in the chat")


    while True:
        try:
            msg = client.recv(BUFSIZ)
        except:
            print("Connection with client was interrupted")
            break
        if msg != bytes("{quit}", "utf8"):
            broadcast(msg,nome + " : ")
        else:
            try:
                client.send(bytes("{quit}", "utf8"))
            except ConnectionResetError:
                print("Connection was closed by client")

            client.close()
            del clients[client]
            broadcast(bytes("%s has left the chat. " % nome, "utf8"))
            break
```

Figura 3: Funzione handle client

Per poter inviare i messaggi di un determinato utente a tutti gli altri utenti connessi, il Server usa la funzione `broadcast`: questa funzione ha semplicemente il compito di ciclare in quello che è il dizionario degli utenti connessi al server e mandare ad ognuno il messaggio, che viene opportunamente passato come parametro di input alla funzione.

Durante tutto il processo di comunicazione con l'utente, le chiamate alle funzioni di comunicazione con il client vengono messe all'interno di un blocco try...except per riuscire a gestire il caso in cui la connessione con l'utente venga persa in maniera imprevista.

3 Funzionamento

Per prima cosa deve essere fatto partire il server, il quale si metterà in ascolto. Poi è possibile eseguire il programma client. Si aprirà una finestra nella quale vi sarà uno spazio con i vari messaggi inviati dal server, uno spazio in cui l'utente potrà scrivere e un bottone con la scritta "Send" per inviare i messaggi.