Progetto "Sistema di Chatroom Client-Server"

- Autore: Mosconi Simone
- Matricola: 0001081246
- Anno accademico: 2023/24
- Mail istituzionale: simone.mosconi2@studio.unibo.it

Descrizione Generale

Il progetto realizza un semplice sistema di chatroom multithread in Python utilizzando la programmazione con socket. Esso si compone di due moduli, i quali implementano rispettivamente le funzionalità del server e del client.
Il server realizzato è in grado di gestire più client contemporaneamente e consente agli utenti di inviare e ricevere messaggi in una chatroom condivisa.
Il client permette agli utenti di connettersi al server, inviare messaggi alla chatroom e ricevere messaggi dagli altri utenti.

Requisiti per eseguire il codice

- Python 3.x
- Le seguenti librerie Python:
  - socket
  - threading
  - tkinter (per il client GUI)

Istruzioni per eseguire il codice

L'applicazione può essere eseguita da una shell qualsiasi con i seguenti comandi:

1. Avviare il server:

```
python server.py
```

2. Avviare uno o più client:

```
python client.py
```

Occorrerà inserire l'indirizzo del server e la porta quando richiesto. Una combinazione valida di default è:

- host: 127.0.0.1
- porta: 53000

Funzionamento del codice

`server.py`

Questo modulo implementa il funzionamento del server del sistema di chatroom.

Codice Rilevante e Spiegazioni

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
```

Importa le librerie necessarie per la gestione dei socket e dei thread.
Questo sistema usa i socket TCP, in quanto si adattano meglio al tipo di sistema realizzato.

```
def accept_connection():
    while True:
        client, client_address = SERVER.accept()
        print("%s:%s connected." % client_address)
        client.send(bytes("Please enter your name and press Enter", "utf8"))
        addresses[client] = client_address
        Thread(target=manage_client, args=(client,)).start()
```

Questa funzione accetta nuove connessioni dei client e avvia un nuovo thread per gestire ciascun client.
Una volta invocata, rimane in ascolto sul socket, e quando arriva una richiesta di connessione il server provvede ad accettarla, fornendo alcune indicazioni sull'utilizzo della chat.

```
def manage_client(client):
    name = client.recv(BUFSIZ).decode("utf8")
    welcome = 'Welcome to the chat %s. Write {quit} to leave.' % name
    client.send(bytes(welcome, "utf8"))
    msg = "%s joined the chat" % name
    broadcast(bytes(msg, "utf8"))
    clients[client] = name

    while True:
        msg = client.recv(BUFSIZ)
        if msg != bytes("{quit}", "utf8"):
            broadcast(msg, name+": ")
        else:
            client.send(bytes("{quit}", "utf8"))
            client.close()
```

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

Questa funzione gestisce la comunicazione con un client connesso. Riceve il nome del client, invia un messaggio di benvenuto, e gestisce l'invio e la ricezione dei messaggi. Se un client invia `{quit}`, chiude la connessione.

```
def broadcast(msg, prefix=""):
    for user in clients:
        user.send(bytes(prefix, "utf8")+msg)
```

Questa funzione invia messaggi a tutti i client connessi.
Per farlo usa il dizionario "clients" contenente le informazioni sui client connessi.

```
clients = {}
addresses = {}
```

Dizionari per tenere traccia dei client e dei loro indirizzi.

```
HOST = ''
PORT = 53000
BUFSIZ = 1024
ADDR = (HOST, PORT)

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

Configurazione del server socket, tra cui la porta e l'interfaccia del server.

```
if __name__ == "__main__":
    SERVER.listen(5)
    print("Waiting for a new connection...")
    ACCEPT_THREAD = Thread(target=accept_connection)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()
```

Il server inizia ad ascoltare nuove connessioni e avvia un thread per accettarle, mettendosi poi in attesa della terminazione del thread in modo da non chiudere subito il server.
L'utilizzo di "__name__" permette di usare il modulo server.py sia come programma che come modulo.

`client.py`

Questo file implementa il client del sistema di chatroom.

Codice Rilevante e Spiegazioni

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
import tkinter as tkt
```

Importa le librerie necessarie per la gestione dei socket, dei thread e per la GUI.
In particolare la GUI è stata realizzata mediante la libreria "Tkinter".

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

Questa funzione gestisce la ricezione dei messaggi dal server e li visualizza nella GUI; il loop infinito permette di mantenere una connessione costante e ricevere i messaggi appena arrivano.
Il metodo "recv()" blocca il ciclo alla ricezione di un messaggio, aggiungendolo alla lisa "msg_list" e permettendone quindi la visualizzazione con Tkinter.

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

Questa funzione invia messaggi al server. Se il messaggio è `{quit}`, chiude la connessione e termina la GUI.

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

Questa funzione gestisce la chiusura della finestra della GUI, inviando `{quit}` al server.

```
window = tkt.Tk()
window.title("ChatRoom")
```

Crea la finestra principale della GUI.

```
messages_frame = tkt.Frame(window)
my_msg = tkt.StringVar()
my_msg.set("Write your messages here.")
scrollbar = tkt.Scrollbar(messages_frame)

msg_list = tkt.Listbox(messages_frame, height=15, width=50,
yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkt.RIGHT, fill=tkt.Y)
msg_list.pack(side=tkt.LEFT, fill=tkt.BOTH)
msg_list.pack()
messages_frame.pack()

entry_field = tkt.Entry(window, textvariable=my_msg)
entry_field.bind("<Return>", send)
entry_field.pack()
send_button = tkt.Button(window, text="Enter", command=send)
send_button.pack()

window.protocol("WM_DELETE_WINDOW", on_closing)
```

In questa sezione viene configurata l'interfaccia grafica della chatroom. La sezione è suddivisa in 4 blocchi che rappresentano le fasi di creazione della GUI:

1. Crea un nuovo frame per mostrare graficamente l'arrivo dei messaggi. Al suo interno è presente una barra di scorrimento della chat;
2. Definisce l'elenco dei messaggi che sarà archiviato in "messages_frame";
3. Crea una box di input per inserire i messaggi da inviare nella chat, il tasto Invio viene associato alla funzione "send()". In alternativa è presente anche un pulsante di invio sulla GUI per permettere all'utente di scegliere con che modalità preferisce inviare i messaggi;
4. La funzione di chiusura "on_closing()" viene chiamata quando l'utente vuole chiudere la finestra e uscire dalla chat.

```
HOST = input('Enter the server host: ')
PORT = input('Enter the port of the server host: ')
if not PORT:
    PORT = 53000
else:
    PORT = int(PORT)

BUFSIZ = 1024
ADDR = (HOST, PORT)

client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect(ADDR)
```

Chiede all'utente di inserire l'indirizzo del server e la porta, quindi crea una connessione socket al server.

```
receive_thread = Thread(target=receive)
receive_thread.start()
tkt.mainloop()
```

Avvia un thread per ricevere messaggi dal server e avvia il loop principale della GUI.