Eraldo Tabaku, Matricola : 0001089499

Documentazione del progetto di Laboratorio di Programmazione di Reti

Traccia 1: Sistema di Chat Client-Server

Implementare un sistema di chat client-server in Python utilizzando socket programming. Il server deve essere in grado di gestire più client contemporaneamente e deve consentire agli utenti di inviare e ricevere messaggi in una chatroom condivisa. Il client deve consentire agli utenti di connettersi al server, inviare messaggi alla chatroom e ricevere messaggi dagli altri utenti.

Introduzione

Si è scelto di realizzare una sistema di CHAT client - server.
Il funzionamento del sistema è molto semplice, un server rimane in ascolto aspettando la connessione da parte dei client; una volta stabilita la connessione con il server, i client possono inviare dei messaggi al server che si occuperà di reindirizzare il messaggio ricevuto a tutti i client connessi.

Funzionamento del codice

Di seguito vengono riportate alcune porzioni di codice significative con i relativi commenti riguardo il funzionamento.

Lato Server

Il server è programmato per gestire connessioni sulla porta 5555 da parte di qualsiasi indirizzo IP.
Viene utilizzata una socket di tipo TCP/IP che poi viene associata all'indirizzo e alla porta specificati precedentemente mediante una bind.
Il server rimane in ascolto aspettando delle connessioni da parte dei client, all'interno di un ciclo while infinito il server accetta tutte le connessioni entranti e crea un thread per la gestione di ognuna di esse.

```python
# Indirizzo e porta per connessione al server

HOST = '0.0.0.0'
PORT = 5555

clients = []

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind((HOST, PORT))

server_socket.listen(5)
print(f"In attesa di connessioni da parte dei client...")

# Accetta e gestisci le connessioni dei client in modo concorrente
while True:

    client_socket, client_address = server_socket.accept()

    clients.append(client_socket)

    client_thread = threading.Thread(target=gestisciClient, args=(client_socket, client_address))
    client_thread.start()
```

La gestione dei thread è affidata ad una funzione gestisciConnessione, questa funzione si occupa della ricezione dei messaggi da parte dei client e del loro indirizzamento, nello specifico tutti i client ricevono il messaggio preceduto dal client address del mittente.
In caso di disconnessione dei client, il server procede ad inserire i client disconnessi in una lista apposita per poi procedere con l'eliminazione della connessione.
Gli errori vengono gestiti tramite delle eccezioni nel caso in cui un client si sia disconnesso dalla chat room viene comunicato un messaggio sul terminale e viene la connessione al client disconnesso.

```python
def gestisciConnessione(client_socket, client_address):
    print(f"Connessione conç: {client_address} accettata")
    try:
        while True:
            # Vengono ricevuti messaggi dal client
            message = client_socket.recv(1024).decode("utf-8")
            if not message:
                break
            print(f"Client {client_address}: {message}")

            # Rimozione dei client disconnessi dalla lista dei client ancora connessi

            disconnected_clients = []
            for c in clients:
                if c.fileno() == -1:
                    disconnected_clients.append(c)
            for dc in disconnected_clients:
                clients.remove(dc)

            message_with_sender = f"{client_address[0]}:{client_address[1]} - {message}"

            # Indirizzamento del messaggio a tutti i client connessi
            for c in clients:
                if c != client_socket:
                    try:
                        c.send(message_with_sender.encode("utf-8"))
                    except Exception as e:
                        print(f"Errore durante l'invio del messaggio a {c}: {e}")
                        # Rimuovi il client dalla lista dei client ancora connessi
                        clients.remove(c)
    except Exception as e:
        print(f"Si è verificato un errore durante la comunicazione con il client: {client_address}")
    finally:
        # viene chiusa la connessione con il client disconnesso
        client_socket.close()
        print(f"ATTENZIONE : Connessione con il client {client_address} chiusa")
```

Lato Client

A lato client la gestione delle funzionalità necessarie è affidata a due funzioni principali: riceviMessaggi e inviaMessaggi; le quali si occupano anche della gestione degli errori durante la comunicazione di messaggi sempre mediante delle eccezioni che comunicano a console l'avvenimento di tali errori.

```
def riceviMessaggi():
    while True:
        try:
            message = client_socket.recv(1024).decode("utf-8")
            if not message:
                break
            chat_box.insert(tk.END, message + '\n')
        except Exception as e:
            print("Si e' verificato un errore durante la ricezione del messaggio")
            break

def inviaMessaggi(event=None):
    try:
        message = message_entry.get()
        chat_box.insert(tk.END, aggiungiMittente(message) + '\n')
        chat_box.see(tk.END)
        client_socket.send(message.encode("utf-8"))
        message_entry.delete(0, tk.END)
    except Exception as e:
        print("Si e' verificato un errore durante l'invio del messaggio")

def chiusura():
    client_socket.close()
    root.destroy()
    sys.exit()
```

Nel client è stata implementata anche una GUI realizzata in maniera simile a quella vista durante le esercitazioni di laboratorio mediante Tkinter, la quale permette di visualizzare tutti i messaggi inviati all'interno della chatroom oltre che permettere l'invio dopo la scrittura e l'invio del messaggio in una apposita casella di testo.
In seguito alla chiusura della finestra, la funzione "chiusura" chiude la comunicazione con il server.

```
# GUI
root = tk.Tk()
root.title("Chatroom")
root.protocol("WM_DELETE_WINDOW", chiusura)

chat_frame = tk.Frame(root)
chat_frame.pack(pady=10)

scrollbar = tk.Scrollbar(chat_frame)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

chat_box = tk.Text(chat_frame, height=20, width=50, yscrollcommand=scrollbar.set)
chat_box.pack(side=tk.LEFT, fill=tk.BOTH)
scrollbar.config(command=chat_box.yview)

message_entry = tk.Entry(root, width=50)
message_entry.pack(pady=10)
message_entry.bind("<Return>", inviaMessaggi)

send_button = tk.Button(root, text="Invia", command=inviaMessaggi)
send_button.pack()
```

Funzionamento e utilizzo del sistema

Gli step per l'esecuzione e l'utilizzo dello script sono:

1. aprendo il terminale ci si sposta nella directory contenente i due file
2. mediante il comando "python nomeScript.py" si eseguono in ordine prima quello relativo al server e poi successivamento, aprendo altri terminali è possibile connettere i vari client dopo aver eseguito lo script relativo al client

```
C:\Users\erald>cd Desktop

C:\Users\erald\Desktop>python Es1Server.py
In attesa di connessioni da parte dei client...
```

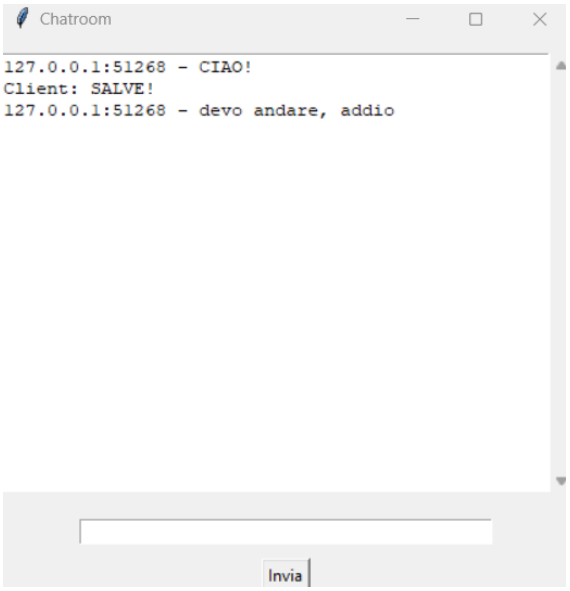
```
C:\Users\erald>cd Desktop

C:\Users\erald\Desktop>python Es1Client.py
```

3. Sul terminale del server appariranno tutti i messaggi relativi a nuove connessioni, connessioni interrotte e messaggi non inviati a causa di errori.

```
C:\Users\erald\Desktop>python Es1Server.py
In attesa di connessioni da parte dei client...
Connessione con: ('127.0.0.1', 51268) accettata
Connessione con: ('127.0.0.1', 51288) accettata
Client ('127.0.0.1', 51268): CIAO!
Client ('127.0.0.1', 51288): SALVE!
Client ('127.0.0.1', 51268): devo andare, addio
Si è verificato un errore durante la comunicazione con il client: ('127.0.0.1', 51268)
ATTENZIONE : Connessione con il client ('127.0.0.1', 51268) chiusa
```

4. La comunicazione tra i client può essere svolta mediante la GUI, scrivendo e inviando i messaggi nell'apposita textbox



(Si rende noto che lo script è stato scritto utilizzando la versione 3.12.3 di Python)