Università degli studi di Bologna
Dipartimento di Informatica – Scienza e Ingegneria - DISI

A.A. 2023-2024

Laurea: Ing. e scienze informatiche
Corso: programmazione di reti

RELAZIONE PROGETTO LABORATORIO
TRACCIA N.1

Prof.: Andrea Piroddi

‘‘ Sistema di Chat Client-Server”

LUDOVICO RICCIO
MATRICOLA: 1069058

INDICE

1. INTRODUZIONE

Il sistema di chat client-server sviluppato in Python permette a più utenti di comunicare in tempo reale tramite una rete. Il server gestisce le connessioni multiple dei client e facilita la trasmissione dei messaggi tra gli utenti. Questa documentazione descrive dettagliatamente l'implementazione del server e del client, fornendo una guida completa su come eseguire il codice, i requisiti necessari e le considerazioni aggiuntive riguardanti la gestione degli errori e delle connessioni.

Requisiti di Sistema

Per eseguire il sistema di chat, sono necessari i seguenti requisiti:

- **Python 3.x**: Il codice è scritto in Python 3, quindi è necessario avere una versione recente di Python installata.
- **Libreria standard di Python**: Il codice utilizza la libreria standard di Python, quindi non sono necessarie librerie esterne.
- **Tkinter**: Utilizzato per la GUI del client. È una libreria standard inclusa nella maggior parte delle distribuzioni Python.

2. DESCRIZIONE DEL CODICE SERVER

Il server è il componente centrale del sistema, responsabile dell'accettazione delle connessioni dei client, della gestione dei messaggi inviati dai client e della trasmissione dei messaggi agli altri client connessi. Il codice del server è strutturato in diverse funzioni chiave:

1. **Gestione delle connessioni in entrata (manageIncomingConnections)**
2. **Gestione di un singolo client (manageClient)**
3. **Broadcast dei messaggi (broadcast)**
4. **Aggiornamento della lista degli utenti (update_user_list)**

Di seguito, viene fornita una spiegazione dettagliata di ciascuna funzione e delle loro interazioni.

2.1. Gestione delle connessioni in entrata

La funzione ‘manageIncomingConnections’ è responsabile dell'accettazione delle nuove connessioni. Utilizza un ciclo infinito per rimanere in ascolto delle richieste di connessione in entrata. Quando un nuovo client si connette, viene creato un nuovo thread per gestire quella connessione specifica, permettendo al server di gestire più connessioni simultaneamente.

```
1   def manageIncomingConnections():
2       while True:
3           try:
4               client, clientAddress = server.accept()
5               print(f"{clientAddress[0]}:{clientAddress[1]} si è collegato.")
6               client.send(bytes("Benvenuto/a! Inserisci il tuo nome.", "utf8"))
7               ipAddress[client] = clientAddress
8               Thread(target=manageClient, args=(client,)).start()
9           except OSError as e:
10              print(f"Errore nell'accettare connessioni: {e}")
11              break
```

Spiegazione:

1. **Accettazione della connessione:** Il server accetta una nuova connessione con ‘server.accept()’ e ottiene l'oggetto socket del client e il suo indirizzo.
2. **Invio del messaggio di benvenuto:** Il server invia un messaggio di benvenuto al nuovo client chiedendo di inserire il proprio nome.

3. **Memorizzazione dell'indirizzo del client:** L'indirizzo del client viene memorizzato in un dizionario ‘ipAddress’.
4. **Creazione di un nuovo thread:** Viene creato un nuovo thread che esegue la funzione ‘manageClient’ per gestire la connessione del client.

2.2. Gestione di un singolo client

La funzione ‘manageClient’ gestisce la comunicazione con un singolo client. Riceve il nome del client, gestisce i messaggi inviati dal client e si occupa della disconnessione del client.

```python
def manageClient(client):
    try:
        name = client.recv(BUFSIZ).decode("utf8")
        msgWelcome = f'{name} ti sei loggato correttamente,\nper abbandonare la Chat clicca su "Disconnetti".'
        client.send(bytes(msgWelcome, "utf8"))
        msg = f"{name} si è unito alla chat!"
        broadcast(bytes(msg, "utf8"))
        clients[client] = name
        update_user_list()

        while True:
            msg = client.recv(BUFSIZ)
            if not msg:
                break
            if msg.decode("utf8") == "/quit":
                client.send(bytes("Sei stato disconnesso.", "utf8"))
                client.close()
                del clients[client]
                broadcast(bytes(f"{name} ha lasciato la chat.", "utf8"))
                print(f"L'utente {name} si è disconnesso volontariamente.")
                update_user_list()
                break
            else:
                broadcast(msg, name + ": ")
    except OSError as e:
        print(f"Errore nella gestione del client: {e}")
    finally:
        if client in clients:
            del clients[client]
            update_user_list()
```

Spiegazione:

- **Ricezione del nome del client**: Il server riceve il nome del client e invia un messaggio di benvenuto personalizzato.
- **Notifica di nuovo utente**: Il server notifica a tutti i client che un nuovo utente si è unito alla chat.
- **Memorizzazione del client**: Il client e il suo nome vengono memorizzati in un dizionario ‘clients’.
- **Gestione dei messaggi**: In un ciclo infinito, il server riceve i messaggi dal client. Se il client invia ‘/quit’, viene disconnesso; altrimenti, il messaggio viene inviato a tutti i client tramite la funzione ‘broadcast’.

2.3. Broadcast dei messaggi

La funzione broadcast invia un messaggio a tutti i client connessi, con un prefisso opzionale che solitamente contiene il nome dell'utente che ha inviato il messaggio.

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

Spiegazione

- **Invio del messaggio:** Il messaggio, con un eventuale prefisso, viene inviato a tutti i client memorizzati nel dizionario ‘clients’.

2.4. Aggiornamento della lista degli utenti

La funzione ‘update_user_list’ invia la lista aggiornata degli utenti a tutti i client connessi.

```
def update_user_list():
    user_list = ",".join(clients.values())
    for sock in clients:
        sock.send(bytes(f"/userlist {user_list}", "utf8"))
```

Spiegazione:

- **Creazione della lista degli utenti:** La lista dei nomi degli utenti viene creata concatenando i nomi memorizzati nel dizionario clients.
- **Invio della lista:** La lista degli utenti viene inviata a tutti i client connessi con un prefisso ‘/userlist’.

2.5. Struttura principale del server

La parte principale del codice server crea il socket del server, lo associa a un indirizzo e porta specificati, e avvia il thread per gestire le connessioni in entrata.

```
clients = {}
ipAddress = {}

HOST = ''
PORT = 1606
BUFSIZ = 1024
ADDR = (HOST, PORT)

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

if __name__ == "__main__":
    server.listen(5)
    print("Server in attesa di connessioni...")
    ACCEPT_THREAD = Thread(target=manageIncomingConnections)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    server.close()
```

Spiegazione

- **Inizializzazione delle variabili:** Vengono inizializzati i dizionari clients e ipAddress, e vengono definiti l'indirizzo e la porta del server.
- **Creazione e binding del socket del Server:** Viene creato un socket del server e associato all'indirizzo e porta specificati.
- **Avvio del thread per le connessioni in entrata:** Viene avviato un thread per gestire le connessioni in entrata, mantenendo il server in ascolto delle nuove connessioni.

3. DESCRIZIONE DEL CODICE DEL CLIENT

Il client è l'interfaccia utente del sistema di chat. Permette agli utenti di connettersi al server, inviare e ricevere messaggi e visualizzare la lista degli utenti online. Il codice del client utilizza la libreria Tkinter per creare una GUI semplice e intuitiva.

3.1. La struttura della GUI

La GUI del client è composta da diverse parti:

1. Frame di login
2. Frame dei messaggi
3. Frame degli utenti online
4. Campo di input per i messaggi
5. Pulsanti di invio e disconnessione

3.2. Frame di login

Il frame di login permette all'utente di inserire il proprio nome e connettersi al server.

```
1   mainWindow = tkt.Tk()
2   mainWindow.title("MultiChatPy")
3
4   nameVar = tkt.StringVar()
5
6   loginFrame = tkt.Frame(mainWindow)
7   tkt.Label(loginFrame, text="Inserisci il tuo nome:").pack(side=tkt.LEFT)
8   entryName = tkt.Entry(loginFrame, textvariable=nameVar)
9   entryName.pack(side=tkt.LEFT)
10  entryName.bind("<Return>", login)
11  loginButton = tkt.Button(loginFrame, text="Login", command=login)
12  loginButton.pack(side=tkt.LEFT)
13  loginFrame.pack()
```

Spiegazione:

- **Creazione della finestra principale**: Viene creata la finestra principale dell'applicazione Tkinter.
- **Creazione del frame di login**: Viene creato un frame per il login con un campo di input per il nome e un pulsante di login.

3.3. Frame dei messaggi

Il frame dei messaggi contiene una Listbox per visualizzare i messaggi della chat e una scrollbar per scorrere i messaggi.

```
messagesFrame = tkt.Frame(mainWindow)
myMsg = tkt.StringVar()
myMsg.set("Scrivi qui i tuoi messaggi.")
scrollbar = tkt.Scrollbar(messagesFrame)

msgList = tkt.Listbox(messagesFrame, height=15, width=60, yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkt.RIGHT, fill=tkt.Y)
msgList.pack(side=tkt.LEFT, fill=tkt.BOTH)
msgList.pack()
messagesFrame.pack()
```

Spiegazione:

- **Creazione del frame dei messaggi**: Viene creato un frame contenente una Listbox per visualizzare i messaggi e una scrollbar per scorrere i messaggi.

3.4. Frame degli utenti online

Il frame degli utenti online contiene una Listbox per visualizzare la lista degli utenti connessi.

```
onlineUsersFrame = tkt.Frame(mainWindow)
onlineUsersLabel = tkt.Label(onlineUsersFrame, text="User online:")
onlineUsersLabel.pack()
onlineUsersList = tkt.Listbox(onlineUsersFrame, height=10, width=30)
onlineUsersList.pack()
onlineUsersFrame.pack(side=tkt.RIGHT, padx=10)
```

Spiegazione:

- **Creazione del frame degli utenti online**: Viene creato un frame contenente una Listbox per visualizzare la lista degli utenti online.

3.5. Campo degli input per i messaggi

Il campo di input permette all'utente di digitare i messaggi da inviare. È presente anche un placeholder che scompare quando l'utente inizia a digitare.

```
    entryField = tkt.Entry(mainWindow, textvariable=myMsg, state=tkt.DISABLED, fg='grey')
    entryField.bind("<Return>", send)
    entryField.bind("<FocusIn>", onEntryClick)
    entryField.bind("<FocusOut>", onFocusout)
    entryField.pack()
    sendButton = tkt.Button(mainWindow, text="Send", command=send, state=tkt.DISABLED)
    sendButton.pack()
```

Spiegazione

- **Creazione del campo di input**: Viene creato un campo di input per i messaggi con un placeholder e un pulsante di invio.

3.6. Pulsanti di invio e disconnessione

Il pulsante di invio permette di inviare i messaggi, mentre il pulsante di disconnessione permette di disconnettersi dal server.

```
    disconnectButton = tkt.Button(mainWindow, text="Logout", command=onDisconnect, state=tkt.DISABLED)
    disconnectButton.pack()
```

Spiegazione:

- **Creazione del pulsante di disconnessione**: Viene creato un pulsante per disconnettersi dal server.

4. FUNZIONI PRINCIPALI DEL CLIENT

Le funzioni principali del client gestiscono la connessione al server, l'invio e la ricezione dei messaggi, e la gestione della disconnessione.

1. Ricezione dei messaggi (receive)
2. Invio dei messaggi (send)
3. Disconnessione (onDisconnect)
4. Chiusura della finestra (onClosing)
5. Login dell'utente (login)
6. Gestione del placeholder (onEntryClick, onFocusout)

4.1. Ricezione dei messaggi

La funzione ‘receive’ riceve i messaggi dal server in modo continuo e aggiorna la GUI di conseguenza.

```
def receive():
    while connected:
        try:
            msg = clientSocket.recv(BUFSIZ).decode("utf8")
            if msg.startswith("/userlist"):
                user_list = msg.split(" ", 1)[1].split(",")
                onlineUsersList.delete(0, tkt.END)
                for user in user_list:
                    onlineUsersList.insert(tkt.END, user)
            elif msg:
                msgList.insert(tkt.END, msg)
        except OSError as e:
            if connected:
                print(f"Errore nel ricevere messaggi: {e}")
            break
```

Spiegazione:

- **Ricezione continuativa**: La funzione riceve i messaggi dal server e aggiorna la Listbox dei messaggi o la lista degli utenti online.

4.2. Invio dei messaggi

La funzione ‘send’ invia i messaggi digitati dall'utente al server.

```python
def send(event=None):
    msg = myMsg.get()
    myMsg.set("")
    try:
        clientSocket.send(bytes(msg, "utf8"))
    except OSError as e:
        msgList.insert(tkt.END, f"Errore di connessione. Impossibile inviare il messaggio: {e}")
```

Spiegazione

- **Invio del Messaggio**: il messaggio viene inviato al server e il campo di input viene pulito.

4.3. Disconnessione

La funzione ‘onDisconnect’ permette all'utente di disconnettersi dal server.

```python
def onDisconnect():
    global connected
    connected = False
    try:
        clientSocket.send(bytes("/quit", "utf8"))
        clientSocket.close()
        entryField.config(state=tkt.DISABLED)
        sendButton.config(state=tkt.DISABLED)
        disconnectButton.config(state=tkt.DISABLED)
        msgList.insert(tkt.END, "Sei stato disconnesso.")
    except OSError as e:
        msgList.insert(tkt.END, f"Errore nella disconnessione: {e}")
```

Spiegazione:

- **Aggiornamento dello stato di connessione**: lo stato di connessione viene aggiornato e il client invia il comando di disconnessione al server.
- **Chiusura della connessione**: il socket del client viene chiuso e la GUI viene aggiornata per riflettere la disconnessione.

4.4. Chiusura della finestra

La funzione ‘onClosing’ gestisce la chiusura della finestra dell'applicazione.

```python
def onClosing(event=None):
    onDisconnect()
    mainWindow.quit()
```

Spiegazione:

- **Chiusura della connessione**: la funzione ‘onDisconnect’ viene chiamata per assicurarsi che il client si disconnetta correttamente prima di chiudere la finestra.

4.5. Login dell’utente

La funzione ‘login’ gestisce il login dell'utente inviando il nome al server.

```
def login(event=None):
    name = nameVar.get()
    if name:
        try:
            clientSocket.send(bytes(name, "utf8"))
            entryName.config(state=tkt.DISABLED)
            loginButton.config(state=tkt.DISABLED)
            entryField.config(state=tkt.NORMAL)
            sendButton.config(state=tkt.NORMAL)
            disconnectButton.config(state=tkt.NORMAL)
        except OSError as e:
            messagebox.showerror("Errore", f"Errore di connessione: {e}")
    else:
        messagebox.showwarning("Attenzione", "Il nome utente non può essere vuoto.")
```

Spiegazione

- **Invio del nome utente**: il nome utente viene inviato al server e la GUI viene aggiornata per abilitare l'invio dei messaggi e la disconnessione.

4.6. Gestione del placeholder

Le funzioni ‘onEntryClick’ e ‘onFocusout’ gestiscono il placeholder del campo di input dei messaggi.

```
def onEntryClick(event):
    if myMsg.get() == "Scrivi qui i tuoi messaggi.":
        myMsg.set("")
        entryField.config(fg='black')

def onFocusout(event):
    if myMsg.get() == "":
        myMsg.set("Scrivi qui i tuoi messaggi.")
        entryField.config(fg='grey')
```

Spiegazione:

- **Rimozione del placeholder**: Quando l'utente clicca sul campo di input, il placeholder viene rimosso.
- **Reimpostazione del placeholder**: Se il campo di input perde il focus e il campo è vuoto, il placeholder viene reimpostato.

5. CONFIGURAZIONE E CONNESSIONE AL SERVER

Il client si connette al server utilizzando le informazioni di host e porta fornite dall'utente.

```
HOST = input('Inserire il Server host (default: localhost): ')
if not HOST:
    HOST = 'localhost'
PORT = input('Inserire la porta del server host (default: 1606): ')
if not PORT:
    PORT = 1606
else:
    PORT = int(PORT)

BUFSIZ = 1024
ADDR = (HOST, PORT)

clientSocket = socket(AF_INET, SOCK_STREAM)

try:
    clientSocket.connect(ADDR)
except OSError as e:
    print(f"Errore di connessione: {e}")
    exit(1)

connected = True

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

Spiegazione:

- **Input dell'utente**: L'utente inserisce l'host e la porta del server.
- **Connessione al server**: Il client tenta di connettersi al server utilizzando le informazioni fornite.
- **Avvio del thread di ricezione**: Viene avviato un thread per ricevere i messaggi dal server.
- **Avvio del loop principale di tkinter**: Viene avviato il loop principale di Tkinter per la gestione della GUI.

6. CONCLUSIONI

La chat client-server implementata in Python utilizza socket per la comunicazione e Tkinter per l'interfaccia grafica. Il server gestisce le connessioni dei client, l'invio e la ricezione dei messaggi, e mantiene una lista degli utenti connessi. Il client permette agli utenti di inviare e ricevere messaggi e di visualizzare la lista degli utenti online. La documentazione e il codice forniti dovrebbero permettere una facile comprensione e utilizzo del sistema di chat.

6.1. Considerazioni aggiuntive

- **Sicurezza**: Il codice attuale non implementa misure di sicurezza come la crittografia dei messaggi. Per un'applicazione reale, sarebbe importante considerare l'uso di TLS/SSL per proteggere i dati trasmessi.
- **Scalabilità**: Questo esempio è adatto per un numero limitato di utenti. Per supportare più utenti, sarebbe necessario implementare meccanismi di bilanciamento del carico e utilizzare server più potenti.

Con questi miglioramenti, il sistema di chat potrebbe diventare una soluzione più completa e adatta per un uso in produzione.