DOCUMENTAZIONE ELABORATO 1
Sistema di Chat Client-Server in Python

Leonte Lucas Antonio
Matricola 0001071582
Dipartimento di Ingegneria e Scienze Informatiche
Anno 2023/2024

Corso di Laurea in Ingegneria e Scienze Informatiche
– Università di Bologna, Sede di Cesena –

Introduzione

Questa documentazione descrive il funzionamento di un sistema di chat client-server implementato in Python utilizzando la programmazione con i socket. Il server è in grado di gestire più client contemporaneamente e permette agli utenti di inviare e ricevere messaggi in una chatroom condivisa.

Requisiti

- **Python 3.x**: Assicurarsi di avere installato Python 3.x sul proprio sistema.
- **Librerie standard di Python**: `socket`, `threading`.

Script del Server: `server.py`

Funzioni Principali

1. **broadcast(message)**:
    - **Descrizione**: Invia il messaggio ricevuto a tutti i client connessi.
    - **Parametri**:
        - `message`: il messaggio da inviare a tutti i client.
    - **Funzionamento**:
        - Itera su tutti i client nella lista `clients` e invia il messaggio a ciascuno di essi.
    - **Codice**:

```
def broadcast(message):
    for client in clients:
        client.send(message)
```

2. **handle_client(client)**:
    - **Descrizione**: Gestisce la comunicazione con un singolo client.
    - **Parametri**:
        - `client`: il socket del client connesso.

- **Funzionamento**:
  - Riceve messaggi dal client e li trasmette a tutti i client connessi usando `broadcast`.
  - Gestisce le eccezioni per rilevare disconnessioni e rimuovere il client dalla lista `clients`.
- **Codice**:

```
def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            broadcast(message)
        except:
            if client in clients:
                index = clients.index(client)
                alias = aliases[index]
                broadcast(f'{alias} ha lasciato la
chat.'.encode('utf-8'))
                clients.remove(client)
                aliases.remove(alias)
                client.close()
            break
```

3. **receive()**:
   - **Descrizione**: Accetta nuove connessioni dai client.
   - **Funzionamento**:
     - Attende connessioni in entrata.
     - Quando un client si connette, richiede un nickname e avvia un thread per gestire la comunicazione con quel client tramite `handle_client`.
   - **Codice**:

```
def receive():
    while True:
        client, address = server.accept()
        print(f"Connessione stabilita con {str(address)}")
        client.send('NICK'.encode('utf-8'))
        alias = client.recv(1024).decode('utf-8')
        aliases.append(alias)
        clients.append(client)
        print(f"Il nickname del client è {alias}")
        broadcast(f"{alias} si è unito alla chat.".encode('utf-
8'))
        client.send('Connesso al server'.encode('utf-8'))
        thread = threading.Thread(target=handle_client,
args=(client,))
        thread.start()
```

4. **Configurazione del server**:
   - **Descrizione**: Configura il socket del server per ascoltare le connessioni in entrata.
   - **Codice**:

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
server.bind(('127.0.0.1', 55555))
server.listen()

print("Server in ascolto...")
receive()
```

Script del Client: `clients.py`

Funzioni Principali

1. **receive()**:
   - **Descrizione**: Riceve messaggi dal server e li stampa.
   - **Funzionamento**:
     - In un loop infinito, riceve messaggi dal server.
     - Se il messaggio è 'NICK', invia il proprio nickname al server.
     - Altrimenti, stampa il messaggio ricevuto.
     - Gestisce le eccezioni per chiudere il client in caso di errore nella connessione al server.
   - **Codice**:

```
def receive():
    while True:
        try:
            message = client.recv(1024).decode('utf-8')
            if message == 'NICK':
                client.send(alias.encode('utf-8'))
            else:
                print(message)
        except:
            print("Errore nella connessione al server.")
            client.close()
            break
```

2. **write()**:
   - **Descrizione**: Invia messaggi al server.
   - **Funzionamento**:
     - In un loop infinito, legge l'input dell'utente e lo invia al server.
   - **Codice**:

```
def write():
    while True:
        message = f'{alias}: {input("")}'
        client.send(message.encode('utf-8'))
```

3. **Configurazione del client**:
   - **Descrizione**: Configura il socket del client e avvia i thread per ricevere e inviare messaggi.
   - **Codice**:

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
client.connect(('127.0.0.1', 55555))

alias = input("Scegli un nickname: ")

receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write)
write_thread.start()
```

Interazione tra Server e Client

1. **Connessione**:
   - Il client si connette al server utilizzando l'indirizzo IP e la porta specificati.
   - Il server accetta la connessione e richiede un nickname al client.
2. **Scambio di Messaggi**:
   - Il client invia il proprio nickname al server.
   - Il server annuncia l'ingresso del nuovo client a tutti i client connessi.
   - I client possono inviare messaggi che vengono trasmessi dal server a tutti i client connessi.
3. **Gestione delle Disconnessioni**:
   - Se un client si disconnette (volontariamente o per errore), il server rileva la disconnessione, rimuove il client dalla lista dei client connessi e notifica gli altri client della disconnessione.

Gestione delle Connessioni Perse

- **Server**:
  - La funzione `handle_client` gestisce le eccezioni per rilevare quando un client si disconnette e rimuove il client dalla lista `clients`, notificando gli altri client.
- **Client**:
  - La funzione `receive` gestisce le eccezioni per chiudere il client in caso di errore nella connessione al server.

Considerazioni Aggiuntive

- **Scalabilità**: Il sistema può essere esteso per includere funzionalità aggiuntive come autenticazione degli utenti, crittografia dei messaggi e gestione delle chatroom multiple.
- **Sicurezza**: Per un'applicazione reale, andrebbe considerato l'uso di protocolli sicuri come TLS/SSL per proteggere la trasmissione dei dati.
- **Logging**: Aggiungere il logging per tenere traccia delle connessioni e disconnessioni dei client in modo dettagliato potrebbe essere opportuno se si desidera utilizzare frequentemente questo sistema