Documentazione del Progetto
Sistema di Chat di gruppo Client-Server

Venturi Luca

1 Introduzione

Questo documento fornisce la documentazione per il mio progetto di un sistema di chat client-server implementato in Python utilizzando socket programming. Il sistema consente a più client di connettersi a un server centrale e comunicare in tempo reale attraverso una chatroom condivisa.

2 Funzionamento

Server Il server viene avviato su localhost e sulla porta 53000. É però possibile avviarlo da linea di comando specifando un indirizzo diverso.

```
SERVER = socket(AF_INET, SOCK_STREAM)
ADDRESS = 'localhost'
PORT = 53000
if len(sys.argv) == 3:
    ADDRESS = sys.argv[1]
    PORT = int(sys.argv[2])
# Binding del socket sull'indirizzo e messa in ascolto.
SERVER.bind( (ADDRESS, PORT) )
SERVER.listen(MAX_CONN_QUEUE)
```

Successivamente avvia un thread ACCEPT_THREAD che si occupa si gestire le connessioni in entrata.

```
# Avvia il thread che gestisce le connessioni in entrata.
ACCEPT_THREAD = Thread(target = handle_connections)
ACCEPT_THREAD.start()
```

La funzione target di questo thread accetta le connessioni in entrata e per ognuna di esse crea un nuovo thread che si occupa di gestire il client specifico che ha fatto la richiesta.

```
def handle_connections():
    """ Accetta le connessioni in entrata """
    while True:
        try:
            # Accetta la connessione in entrata.
            client, client_address = SERVER.accept()
            print("%s:%s si e' connesso."%client_address)

# Invia le istruzioni per iniziare la chat al client.
            client.send(bytes("Ciao! Scrivi il tuo nome e premi invio per confermarlo!", "utf8"))

# Registra l'indirizzo del client.
            addresses[client] = client_address

# Crea un nuovo thread che da ora in poi si occupera'
# di gestire quel client.
            Thread(target=handle_client, args=(client,))
                .start()

        except Exception as e:
            print(f"Errore durante l'accept della\
                connessione in entrata: {e}")
```

Questo thread inzia prendendo l'username del client e inoltrandolo a tutti membri della chat. Ma la funzione più importante è la successiva gestione dei suoi messaggi.

```
connected = True
# Gestisce tutti i messaggi successivi con il client.
while connected:
    try:
        # Riceve il messaggio e se e' diverso da /quit lo inoltra a tutti.
        msg = client.recv(BUFSIZ)
        if msg and msg != bytes("/quit", "utf8"):
            broadcast(msg, username + ": ")
        else:
            # Se era /quit cancella i suoi dati e notifica tutti della sua dipartita.
            connected = False
    except Exception as e:
        connected = False
```

Continua a leggere tutti i messaggi ricevuti dal client fino a quando non riceve /quit o fino a quando il client non chiude la connessione. Ogni messaggio rivevuto lo manda in broadcast a tutti i client connessi.

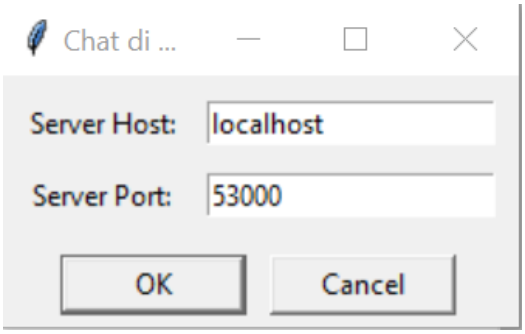
Quando la connessione viene chiusa si occupa di chiudere la connes-

sione, eliminare i dati del client e di avvertire tutti che il client è uscito.

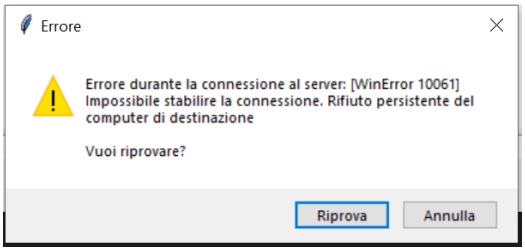
```
while connected:
    ...

del users[client]
del addresses[client]
broadcast(bytes("%s ha abbandonato la chat." % username,
    "utf8"))
client.close()
```

Client Il client si avvia creando una dialog che chiede all'utente l'indirizzo IP e la porta del server a cui vuole connettersi. In caso di dati non validi o mancanti usa quelli di default, ovvero localhost e porta 53000

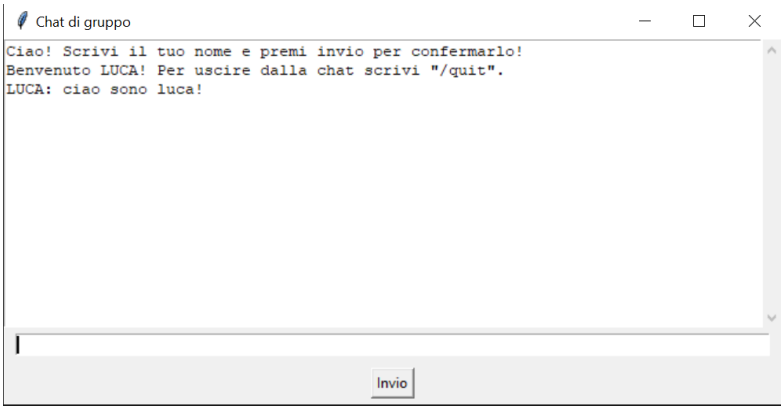


Una volta ottenuto l'indirizzo del server prova a connettersi e in caso non ci riesca chiede all'utente se vuole riprovare.



Se invece riesce a connettersi crea un thread che riceve i messaggi dal server e li scrive in una listbox. Viene anche creata una GUI molto semplice in che contiene la listbox con tutti i messaggi, una Entry per scrivere il messaggio e un pulsante invio.

Quando l'utente scrive un messaggio nella Entry questo viene inviato al server. (Per come è fatto il server inizia ricevendo l'istruzione di comunicare il proprio nome come primo messaggio).



Se il messaggio inviato è /quit viene chiusa la connessione con il server, così come quando viene premuta la x

Se viene persa la connessione viene chiesto se si vuole provare a riconnettersi al server, in caso negativo si chiude la chat.

3 Requisiti di Sistema

- Python 3.x installato sul sistema.
- Libreria Tkinter installata per eseguire il client con GUI.
- Connessione di rete per la comunicazione tra client e server.

4 Istruzioni per l'Esecuzione

É possibile avviarli con doppio click nella maggior parte dei casi. Altrimenti di seguito le istruzioni per avviarli da terminale.

4.1 Server

Per avviare il server da terminale

1. Aprire un terminale.
2. Navigare nella directory contenente il file del server.
3. Eseguire il seguente comando:

```
    py serverChatRoom.py [host] [port]
    # host e port sono opzionali, se non specificati sono
    localhost e 53000
    # Se non funziona provare con 'python' invece di 'py'

```

4. Il server si metterà in ascolto sull'indirizzo specificato (default-¿ localhost:53000).

4.2 Client

Per avviare il client con GUI da linea di comando:

1. Aprire un terminale.
2. Navigare nella directory contenente il file del client.
3. Eseguire il seguente comando:

```
    py clientChatRoom.py
    # Se non funziona provare con 'python' invece di 'py'

```

4. Si aprirà una finestra di Tkinter con l'interfaccia della chat.
5. Inserire l'indirizzo del server e la porta quando richiesto, oppure utilizzare i valori di default (localhost:53000).
6. Premere il tasto "OK" per avviare la connessione al server.
7. Scrivere il proprio username in chat come primo messaggio.