# Programmazione di Reti
# Chat client-server

Palazzini Luca

May 17, 2024

## 1 Funzionamento del sistema

Il server utilizza un funzionamento a socket per mantenere una comunicazione fra il server ed i client. Per assicurarsi che i client riescano a comunicare con il server contemporaneamente si utilizzano thread, sia lato client che lato server.

### 1.1 Specifiche lato server

A lato **server**, si ha un thread per client, per assicurarsi la comunicazione indipendente fra tutti i client.

```
while not shouldClose:
    try:
        # When a new client joins, get the data
        clientData: tuple[socket, AddressInfo] = server.accept()
        clientSocket: socket = clientData[0]
# ----- other server related code ----- #
        clientThread: Thread = Thread(target=lambda: handleClient(
    clientSocket))
        clientThreads.append(clientThread)
        clientThread.start()
    except KeyboardInterrupt:
        # In case there is a keyboard interrupt, end the communications
        shouldClose = True
    except Exception as e:
        # In any exception, end the communications
        print(e)
        shouldClose = True
```

Listing 1: Dizionario server

Inoltre utilizza un dizionario per contenere le informazioni relative ad I client: socket, informazioni relative al suo indirizzo ip, ed il suo username.

```
# Dictionary to keep all the clients in one structure
    connectedClients: dict[socket, tuple[AddressInfo, str]] = {}
# ----- other server related code ----- #
    # When a new client joins, get the data
    clientData: tuple[socket, AddressInfo] = server.accept()
    clientAddressInfo: AddressInfo = clientData[1]
    # Read the new client's name and broadcast it to the chat
    name: str = clientSocket.recv(BUFFER_SIZE).decode("utf8")
```

```
9       connectedClients[clientSocket] = (clientAddressInfo, name)
```

Listing 2: Dizionario server

Ogni volta che un client si disconnette, esso viene rimosso dal dizionario e viene terminato il suo thread.

```
1 if msg == COMMAND_PREFIX + COMMAND_QUIT:
2 # ----- other server related code ----- #
3     del connectedClients[client]
4     client.close()
```

Listing 3: Logout del client

### 1.2 Specifiche lato client

A lato **client**, viene creato un thread apposito una volta che viene stabilita la connessione con il server. Questo thread servirà per assicurarsi solamente la funzionalità separata fra l'ascolto di messaggi nuovi, e il funzionamento della GUI.
La pagina di login contiene già informazioni di default relative ad il server di default, queste vengono eliminate quando ci si clicca sopra
La GUI utilizza diverse callback functions per mandare messaggi dalla GUI al socket

- chatCloseCallback (per quando la schermata della chat viene chiusa)
- serverConnectCallback (quando l'user ha dato le informazioni per il login, tenta a fare un accesso)
- sendMessageCallback (per mandare un messaggio al server una volta premuto "enter" o il tasto "send"

## 2 Utilizzo del codice

Il codice non richiede alcuna libreria o applicazione aggiuntiva oltre a quelle già presenti nella repository.
Nonostante ciò queste sono le librerie utilizzate (native di python3) nel progetto:

- Python 3.x
- Threading (per rendere l'applicazione multithreaded)
- Socket (per la gestione delle connessioni client-server)
- Errno (per il controllo di errori dei socket)
- Tkinter (utilizzata per la GUI)
- Typing (per aggiungere type annotations al codice)
- Re (per la gestione di "regular expressions" in python, ovvero per controllare la correttezza dell'input dell'utente)
- Colorsys (per la gestione migliorata dei colori)
- Random (per una generazione randomica dei colori)

Tutte le variabili principali utilizzate nel codice si trovano nel file common/defaultParams.py, includendo:

- La dimensione del buffer (1024 bytes)
- Il default IP (127.0.0.1, ovvero localhost)
- La porta default (53000)

Seguono sotto i passaggi per eseguire il codice correttamente:

1. Eseguire lo script lato server (chatServer.py) su un terminale:

```
./chatServer.py
# OR
python chatServer.py
```

Listing 4: Esecuzione lato server

2. Da qui il server scriverà in console di essere stato eseguito.
3. Eseguire poi una o più istanze dello script lato client (chatClient.py) per utilizzare la chat:

```
./chatClient.py
# OR
python chatClient.py
```

Listing 5: Esecuzione lato client

4. Da qui il client richiederà tre cose:
   (a) L'ip del server
   (b) La porta del server
   (c) L'username da utilizzare (' ' non ammesso)
5. Una volta passate queste informazioni al client, esso potrà entrare nella chat (affinchè le informazioni siano corrette), e verrà aperta una nuova finestra contenente la schermata della chat.
6. Da qui, basta inserire il messaggio che si vuole mandare nell'apposito input
7. Premere invio o il tasto "send" per inviare il messaggio.
8. Per uscire dalla chat basta premere il tasto "Quit" o chiudere la finestra, e si verrà automaticamente rimossi dal server.
9. Da qui si può chiudere il server.

Nel caso il server venga chiuso mentre ci sono ancora client in ascolto, esso manderà il messaggio di "/quit" a tutti i client rimanenti in ascolto (notare come questo non chiuderà la finestra del client).

# 3 Considerazioni aggiuntive

- Nella creazione del programma ho cercato di creare una GUI che sia "responsive" ovvero che più la finestra è grande, più i contenuti prendono più spazio sullo schermo.
- Ho cercato di gestire la maggior parte delle eccezioni causate da socket o chiusure di finestre e kill di processi
- I colori della GUI della chat sono generati randomicamente, scegliendo un colore casuale, e prendendo altri 4 colori analoghi (ruotando la routa dei colori di $\pm 30°, \pm 15°$)