Sistema di Chat Client-Server

Funzionamento del sistema

Il sistema si compone di due file distinti:

1. Il file che rappresenta il Server (ChatRoom_LatoServer.py),
2. Il file che rappresenta il Client (ChatRoom_LatoClient.py).

Il sistema, tramite l'uso di socket programming e della libreria threading, è in grado di gestire una chatroom condivisa in cui tutti gli utenti, connessi a un Server comune in contemporanea, possono inviare e ricevere messaggi tra di loro.
Proseguendo nella spiegazione del funzionamento del sistema, concentriamoci adesso sul file che svolge la funzione di Server. Il file si compone di 3 metodi: handleClient, clientManager e broadcast. Il primo si occupa di gestire le connessioni dei Client in entrata. Quando viene invocata, essa si mette in ascolto sul socket TCP, e appena arriva una richiesta di connessione da un Client, il Server gli restituisce un messaggio di saluto accompagnato da alcune indicazioni sull'utilizzo della chat. Stampa sulla console in cui è stato eseguito il Server, l'indirizzo ip e il numero di porta del Client che si è connesso. In aggiunta, l'indirizzo del Client viene salvato in un dizionario chiamato "addr", e poi viene messo in esecuzione il thread del Client appena connesso tramite il metodo start().

```
8   #Funzione per gestire le connessioni dei client
9   def handleClient():
10      while True:
11          client,clientAddr = Server.accept()
12          print("%s:%s connected!" % clientAddr)
13          #Invio al client un messaggio di benvenuto
14          client.send(bytes("Welcome!!! Please enter your name and press Return", "utf8"))
15          #Mediante un dizionario salviamo i client connessi
16          addr[client] = clientAddr
17          #Mettiamo in esecuzione il thread del client che si è appena connesso
18          th.Thread(target=clientManager, args=(client,)).start()
```

Metodo handleClient del Server

Il metodo clientManager ha il compito di gestire il thread per ciascun Client che si connette al Server. All'interno del metodo viene inviato un messaggio al Client in cui gli viene comunicato come uscire dalla chatroom. Inoltre, attraverso il metodo broadcast, gli altri utenti già connessi, vengono avvisati che un altro utente si è unito alla chat.

```
20  #Funzione per gestire ognuno dei client connessi
21  def clientManager(client):
22      clientName = client.recv(bufferSize).decode("utf8")
23      #Indica al client come uscire dalla chat
24      entryMessage = "Hi %s, if you want to leave please write {exit}." % clientName
25      client.send(bytes(entryMessage, "utf8"))
26      message = "%s joined the chat" % clientName
27      #Invio a tutti i client un messaggio in cui indico che un client è entrato nella chat
28      broadcast(bytes(message, "utf8"))
29      #Aggiorno il dizionario con il nome del client
30      clients[client] = clientName
31
32      #
33      while True:
34          message = client.recv(bufferSize)
35          if message != bytes("{exit}", "utf8"):
36              broadcast(message, clientName + ": ")
37          else:
38              print("%s:%s disconnected!" % addr[client])
39              client.close()
40              del clients[client]
41              broadcast(bytes("%s left the chat." % clientName, "utf8"))
42              break
```

Metodo clientManager del Server

L'ultimo metodo è il broadcast che viene utilizzato per inviare messaggi a tutti gli utenti della chat. Il metodo esegue un ciclo for sul dizionario contente i Client connessi e invia sul relativo socket il messaggio inviato dal mittente, che è sempre un Client. Insieme al messaggio viene inviato il prefisso che non è altro che il nome utente del mittente.

```
45  #Funzione per inoltrare un messaggio a tutti i client
46  def broadcast(message, sender = ""):
47      for client in clients:
48          client.send(bytes(sender, "utf8") + message)
```

Metodo broadcast del Server

A riga 62 del file ChatRoom_LatoServer.py viene utilizzato un comando che ci permette di eseguire il nostro codice sorgente, sia come script eseguito da terminale, sia come modulo all'interno di un altro programma. Successivamente creiamo un'istanza di thread con la funzione target, che fa riferimento al metodo handleClient, e poi con start() la inizializziamo. Infine, con join() attendiamo che un thread abbia completato il suo lavoro prima di far ritornare il controllo allo script principale. Ora, invece, occupiamoci del file che svolge la funzione di Client. Il file si compone anch'esso di 3 metodi: receiveMessages, sendMessages e close. Il primo, che si occupa di gestire la ricezione dei messaggi, è costituito da un ciclo while infinito poiché il nostro obiettivo è quello di ricevere messaggi da altri utenti quando siamo connessi e di inviarne altri quando vogliamo. Dentro al ciclo utilizziamo due elementi importanti: recv(), che è una funzione bloccante cioè blocca il processo fino a quando non riceve un messaggio, e l'altro è messageList che contiene i messaggi che visualizzeremo a schermo.

```
10  #Funzione per la ricezione dei messaggi
11  def receiveMessages():
12      while True:
13          try:
14              #Il client si mette in ascolto dei messaggi che
15              #arrivano sul socket del server
16              message = clientSocket.recv(bufferSize).decode("utf8")
17              #Visualizziamo tutti i messaggi
18              messageList.insert(tkt.END, message)
19          except OSError:
20              print("Server connection lost.")
21              break
```

Metodo receiveMessages del Client

Il metodo sendMessages, invece, si occupa dell'azione opposta, cioè gestisce l'invio dei messaggi dal Client. La variabile message contiene il messaggio da inviare, infatti, grazie al metodo get() siamo in grado di estrarre il testo scritto in input dalla casella di testo. Successivamente, ripuliamo il campo di input con il metodo set(), impostando la variabile myMessage a un testo vuoto. Poi inviamo il messaggio al Server, che trasmetterà il messaggio a tutti i Client connessi. Se il messaggio è "{exit}" invece, non verrà mandato agli altri utenti, ma chiuderemo il socket TCP e con destroy() sulla window faremo la stessa cosa sulla finestra dell'interfaccia grafica.

```
23  #Funzione che si occupa dell'invio dei messaggi
24  def sendMessages(event = None):
25      #Estraiamo il testo dalla casella di input
26      message = myMessage.get()
27      #Pulisco la casella di input
28      myMessage.set("")
29      #Invio il messaggio al socket del server
30      clientSocket.send(bytes(message, "utf8"))
31      if message == "{exit}":
32          clientSocket.close()
33          #Chiude l'interfaccia grafica del client
34          window.destroy()
```

Metodo sendMessages del Client

L'ultimo metodo è quello che viene invocato quando l'utente vuole chiudere l'interfaccia grafica con il pulsante rosso in alto a destra della finestra. Infatti, il metodo imposta a "{exit}" il messaggio di input e lo invia con sendMessages, facendo eseguire il contenuto del costrutto if del metodo soprastante.

```
36  #Funzione invocata alla chiusura dell'interfaccia grafica della chat
37  def close(event = None):
38      myMessage.set("{exit}")
39      sendMessages()
```

Metodo close del Client

Infine, nel Client da riga 41 a riga 65, è presente il codice che si occupa di creare la finestra GUI (Graphical User Interface) dell'utente. Gli elementi più importanti sono il messagesFrame per contenere l'elenco dei messaggi, la variabile stringa myMessage che viene usata per salvare il messaggio di input e il metodo bind() chiamato sull'entryField (ovvero la casella di input), con il quale faccio corrispondere la pressione del tasto "Return" all'invio del messaggio con il metodo sendMessages.

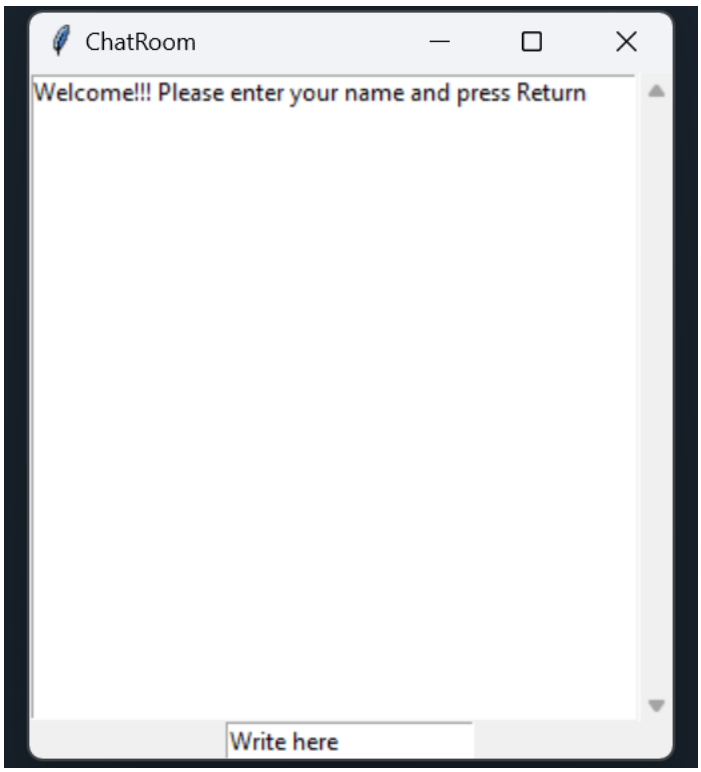
Esecuzione del codice

Per poter eseguire il codice, ad esempio sull'IDE (Integrated Development Environment) di Anaconda Spyder, bisogna star utilizzando la versione 3.11.5 di Python, per evitare di incorrere in problemi legati all'importazione delle librerie. Dopo essere sicuri di ciò, si deve:

1. Aprire entrambi i file (ChatRoom_LatoServer.py e ChatRoom_LatoClient.py) sullo stesso IDE,
2. Aprire una sola console in cui andremo a eseguire il Server e un numero di console pari al numero di Client che vogliamo mettere in esecuzione,
3. Far eseguire il file ChatRoom_LatoServer.py su una console con il pulsante Run file sulla toolbar in alto o premendo il tasto F5,
4. Far eseguire il file ChatRoom_LatoClient.py su ognuna delle console create per i Client con il pulsante Run file sulla toolbar in alto o premendo il tasto F5.

Ogni Client richiederà per la configurazione iniziale 2 parametri: l'Host Server e la porta dell'Host Server. Per il primo parametro si deve digitare l'indirizzo ip di loopback, e cioè 127.0.0.1 oppure si può digitare "localhost" (senza i doppi apici), poi dopo aver premuto il tasto "Return" verrà richiesto il numero di porta, per cui si deve digitare il numero 5555 e premere un'altra volta "Return". A questo punto, sulla console del Server dovrebbe comparire un messaggio simile a questo:

“127.0.0.1:numeroporta connected!”. Questo vuol dire che la connessione è riuscita e si dovrebbe aprire una finestra simile a questa



Finestra con la GUI del Client

Se così avviene, possiamo prima di tutto inserire al posto di “Write here” il nostro nome e poi cliccando “Return” veniamo salutati e ci viene anche indicato il modo con cui uscire dalla chatroom. Ecco, da adesso in avanti possiamo scrivere, sempre nella casella in basso, e ricevere messaggi all’interno della chat. Ovviamente, il procedimento sopra riportato dovrà essere ripetuto per ogni nuova istanza del Client creata.

Considerazioni aggiuntive

Un altro elemento da sottolineare è come funziona l’uscita di un utente dalla chatroom. Ci sono 2 modi per uscire. Il primo consiste nello scrivere nella casella di input “{exit}” (senza i doppi apici) e premere il tasto “Return”, in questo modo si chiuderà la finestra della GUI e sulla console del Client apparirà il messaggio “Server connection lost.”, mentre su quella del Server apparirà un messaggio del genere: “127.0.0.1:numeroporta disconnected!”. Il secondo, invece, consiste nel chiudere la finestra apertasi con la connessione del Client premendo il tasto rosso in alto a destra, anche in questo modo sulla console del Client e su quella del Server appariranno i messaggi citati precedentemente.
Un ultimo elemento che merita attenzione è il codice scritto nel file ChatRoom_LatoClient.py da riga 19 a riga 21. In questa parte viene gestita l’eccezione di tipo “OSError”, cioè un evento che altera il normale flusso di controllo e di esecuzione di un programma che può verificarsi se la connessione al Server viene persa o se si verifica un problema di I/O. Nel caso in cui ciò avvenga, viene stampato sulla console del Client il messaggio “Server connection lost.” e con “break” si esce dal ciclo infinto del metodo receiveMessages, interrompendo quindi la ricezione dei messaggi.