PROGETTO DI PROGRAMMAZIONE DI RETI

2023/2024

Simone Carpi

Chat Client Server

ORGANIZZAZIONE GENERALE

Il progetto di compone di due script in Python:

- Il primo è **Progetto-Server** che contiene il codice per il funzionamento di un server che si occupa di gestire e controllare una chat-room condivisa tra più client;
- Il secondo è **Progetto-Client** che, invece, contiene lo script lato client della chat-room Client Server ed è l'applicazione che permette ai vari utenti di poter entrare nella chat, di dialogare ed eventualmente uscire da essa.

Il protocollo di trasporto utilizzato nella comunicazione tra i Client e il Server è il TCP perché permette di avere una connessione affidabile garantendo la corretta ricezione di messaggi che sono stati inviati all'interno della chat-room.

In seguito, sarà spiegato il funzionamento dei due script.

LATO SERVER

La componente Server deve poter accettare le varie richieste di connessione da parte dei client per l'accesso alla chat-room. Inoltre, deve anche favorire lo scambio dei messaggi tra più client che sono operativi simultaneamente, facendo sì che il messaggio inviato da un client sia visibile a tutti gli altri connessi alla chat.

Per iniziare, si deve per prima cosa assegnare un numero di porta e un indirizzo di rete alla **socket** che sarà assegnata al Server, in modo che i vari client possano contattarlo. Questa sarà l'interfaccia di dialogo tra i Client e il Server.

```
HOST = ''
PORT = 53000
BUFSIZ = 1024
ADDR = (HOST, PORT)
```

L'indirizzo di rete, presente nella socket principale, non viene specificato perché è l'indirizzo di **loopback**, quindi **127.0.0.1**, che dovrà poi essere inserito dai vari

utenti come parametro per instaurare la connessione iniziale con il Server a partire dal Client.

Sarà necessario creare due strutture dati quali: **clients** e **indirizzi,** due dizionari per la registrazione dei client in entrata e dei loro nomi associati agli indirizzi.

```
clients = {}
indirizzi = {}
```

SOCK_STREAM mi permette di dire che la comunicazione deve avvenire utilizzando il protocollo TCP. **AF_INET** mi definisce il dominio (o famiglia) della socket che si occupa di comunicazione tra host remoti, tramite Internet.

```
SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)

if __name__ == "__main__":
    SERVER.listen(5)
    print("In attesa di connessioni...")
    ACCEPT_THREAD = Thread(target=accetta_connessioni_in_entrata)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()
```

Nel main, viene impostata una coda di lunghezza 5 attraverso l'istruzione "**listen**" e poi viene avviato il Thread generale che si occuperà di gestire eventuali connessioni in entrata. Il comando "**join**" viene

utilizzato per attendere fino a quando un thread ha completato il suo lavoro. In questo caso, lo si utilizza perché si vuole che lo script principale attenda il completamento e non salti alla riga successiva, che causa la chiusura del server tramite l'operazione "**close**".

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
```

Per poter utilizzare le socket è necessario importare il modulo "**socket**", mentre per utilizzare i Thread si è importato il modulo "**threading**" con la classe Thread.

```
def accetta_connessioni_in_entrata():
    while True:
        client, client_address = SERVER.accept()
        print("%s:%s si è collegato." % client_address)
        #al client che si connette per la prima volta fornisce alcune indicazioni di utilizzo
        client.send(bytes("Salve! Digita il tuo Nome seguito dal tasto Invio!", "utf8"))
        # ci serviamo di un dizionario per registrare i client
        indirizzi[client] = client_address
        #diamo inizio all'attività del Thread - uno per ciascun client
        Thread(target=gestice_client, args=(client,)).start()
```

La funzione "**Accetta_connessioni_in_entrata**" si occupa di gestire le connessioni in entrata da parte dei client. Quando viene invocata si mette in ascolto sulla socket principale del server e appena si accorge di una richiesta di un client, lo registra tramite i

dizionari precedentemente creati e lo accoglie con una frase di benvenuto. Poi avvia un nuovo Thread che si occuperà del Client e invoca la funzione che si occuperà di gestirlo.

```
def gestice_client(client):  # Prende il socket del client come argomento della funzione.
    nome = client.recv(BUFSIZ).decode("utf8")
    #da il benvenuto al client e gli indica come fare per uscire dalla chat quando ha terminato
    benvenuto = 'Benvenuto %s! Se vuoi lasciare la Chat, scrivi {quit} per uscire.' % nome

    try:
        client.send(bytes(benvenuto, "utf8"))
        msg = "%s si è unito all chat!" % nome
        #messaggio in broadcast con cui vengono avvisati tutti i client connessi che l'utente x è entrato
        broadcast(bytes(msg, "utf8"))
        #aggiorna il dizionario clients creato all'inizio
        clients[client] = nome

        #si mette in ascolto del thread del singolo client e ne gestisce l'invio dei messaggi o l'uscita dalla Chat
        while True:
            try:
                msg = client.recv(BUFSIZ)
            except (ConnectionAbortedError, ConnectionResetError):
                print("%s:%s si è scollegato prima di entrare nella chat." % indirizzi[client])
                client.close()
                del clients[client]
                break

            if msg != bytes("{quit}", "utf8"):
                if msg is not None:
                    broadcast(msg, nome+": ")
            else:
                try:
                    client.send(bytes("{quit}", "utf8"))
                finally:
                    client.close()
                    del clients[client]
                    broadcast(bytes("%s ha abbandonato la Chat." % nome, "utf8"))
                    print("%s ha abbandonato la Chat." % nome)
                    break

    except ConnectionResetError:
        print(f'{indirizzi[client]} è uscito dalla chat forzatamente')
```

Questa sarà la funzione che si occuperà di gestire il Thread del client. All’interno di questo metodo ci si occupa di avvisare tutti gli altri utenti, che si sono registrati all’interno della chat-room, dell’ingresso del nuovo utente appena arrivato. Successivamente si entra all’interno di un loop infinito nella quale si è deciso di gestire alcune eccezioni come la chiusura

forzata, da parte del client, tramite l'icona presente sulla GUI che viene mostrata a schermo. Infatti, in quella situazione lo script tirava eccezione perché il messaggio "**msg**", siccome l'utente usciva senza digitare "**{quit}**" (comando necessario per la corretta chiusura del Thread), era nullo. Inoltre, l'eccezione era causata anche dal fatto che veniva chiusa la connessione in modo forzato, proprio perché non veniva effettuata la corretta chiusura della socket.

Per la corretta chiusura, nel loop si controlla se il messaggio che è stato digitato dall'utente contiene "{quit}" che avvisa il Thread che gestisce il client di chiudere le risorse, quindi la socket e di avvisare gli altri utenti della chat-room che si sta uscendo tramite la funzione "**broadcast**".

```
def broadcast(msg, prefisso=""):  # il prefisso è usato per l'identificazione del nome.
    if clients:
        for utente in clients:
            utente.send(bytes(prefisso, "utf8")+msg)
```

La funzione "broadcast" si occupa di inviare un messaggio "msg" a tutti gli utenti collegati alla chat-room. All'inizio effettua un controllo per capire se il dizionario clients è diverso da vuoto.

LATO CLIENT

La componente Client è quella che si occupa di inviare i messaggi dell'utente assegnato e di ricevere quelli degli altri utenti presenti nella chat-room.

```
HOST = input('Inserire il Server host: ')
if not HOST:
    HOST = '127.0.0.1'

PORT = input('Inserire la porta del server host: ')
if not PORT:
    PORT = 53000
else:
    PORT = int(PORT)

BUFSIZ = 1024
ADDR = (HOST, PORT)
```

All'inizio, una volta che è stato fatto partire il client, è necessario configurare la socket a cui si deve connettere il client, cioè quella del server. I parametri vengono presi in input e sono quelli riguardanti il server, quali: l'indirizzo del server e il numero di porta in cui il server è in ascolto (entrambi i parametri possono essere omessi perché aggiunti a livello di codice).

```
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect(ADDR)

receive_thread = Thread(target=receive)
receive_thread.start()
# Avvia l'esecuzione della Finestra Chat.
tkt.mainloop()
```

Quindi, si crea la socket del client, la si connette ad una socket remota che è quella del server e si dà inizio al thread per la ricezione dei messaggi. Poi si avvia l'esecuzione del "**mainloop**" che si occupa di tenere in esecuzione la GUI.

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
import tkinter as tkt
```

Per il corretto funzionamento del codice è necessario importare sia il modulo socket che il modulo threading, come per il Server. Invece, per scrivere la GUI si utilizza uno strumento di Python che si chiama Tkinter e si può utilizzare facendo l'import di tkinter.

A questo punto ci si occupa di creare la finestra che permetterà all'utente sia di registrarsi con un proprio

nome utente sia di inviare e ricevere messaggi sulla chat.

Si crea un contenitore che mi permette di contenere la finestra, quindi il frame, e una variabile stringa (my_msg) che si occupa di contenere il messaggio che si vuole inviare.

Nella finestra è possibile inviare i messaggi sia tramite il tasto *Return* della tastiera sia con il pulsante *Invio* che è presente nella GUI. Inoltre, viene definito un modo per poter chiudere la finestra della GUI richiamando la funzione *on_closing*.

```
finestra = tkt.Tk()
finestra.title("Chat_Laboratorio")

#creiamo il Frame per contenere i messaggi
messages_frame = tkt.Frame(finestra)
#creiamo una variabile di tipo stringa per i messaggi da inviare.
my_msg = tkt.StringVar()
#indichiamo all'utente dove deve scrivere i suoi messaggi
my_msg.set("Scrivi qui i tuoi messaggi.")
#creiamo una scrollbar per navigare tra i messaggi precedenti.
scrollbar = tkt.Scrollbar(messages_frame)

# La parte seguente contiene i messaggi.
msg_list = tkt.Listbox(messages_frame, height=15, width=50, yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkt.RIGHT, fill=tkt.Y)
msg_list.pack(side=tkt.LEFT, fill=tkt.BOTH)
msg_list.pack()
messages_frame.pack()

#Creiamo il campo di input e lo associamo alla variabile stringa
entry_field = tkt.Entry(finestra, textvariable=my_msg)
# leghiamo la funzione send al tasto Return
entry_field.bind("<Return>", send)

entry_field.pack()
#creiamo il tasto invio e lo associamo alla funzione send
send_button = tkt.Button(finestra, text="Invio", command=send)
#integriamo il tasto nel pacchetto
send_button.pack()

finestra.protocol("WM_DELETE_WINDOW", on_closing)
```

Per quanto riguarda la funzione ”**receive**” abbiamo un ciclo infinito perché si vuole inviare i messaggi in qualunque momento e riceverli quando siamo connessi. Tutti i messaggi inviati e ricevuti nella chat saranno visualizzabili a schermo come un elenco.

Si è cercato di gestire l’eccezione **ConnectionAbortedError** perché poteva capitare che uscendo forzatamente dalla chat e non essendoci più il client attivo venisse tirata questa eccezione.

```
def receive():
    while True:
        try:
            #quando viene chiamata la funzione receive, si mette in ascolto dei messaggi che
            #arrivano sul socket
            msg = client_socket.recv(BUFSIZ).decode("utf8")
            #visualizziamo l'elenco dei messaggi sullo schermo
            #e facciamo in modo che il cursore sia visibile al termine degli stessi
            msg_list.insert(tkt.END, msg)
            # Nel caso di errore e' probabile che il client abbia abbandonato la chat.
        except (OSError, ConnectionAbortedError):
            break
```

Il metodo **send** mi permette di inviare il messaggio digitato dall’utente sulla chat-room dopo averlo inviato al server che si occuperà di trasmetterlo a tutti gli altri utenti connessi. L’argomento *event* viene messo perché viene implicitamente passato da Tkinter quando si preme il pulsante di invio sulla GUI. Si estrapola il messaggio che l’utente vuole inviare e lo si invia al server. Se l’invio tira eccezione vuol dire che

il server non è più attivo e si consiglia all'utente di uscire, tramite un messaggio testuale, dalla chat. In ogni caso, si controlla se il messaggio che si vuole inviare è "{quit}" così da chiudere la socket e liberare risorse, oltre a chiudere la finestra.

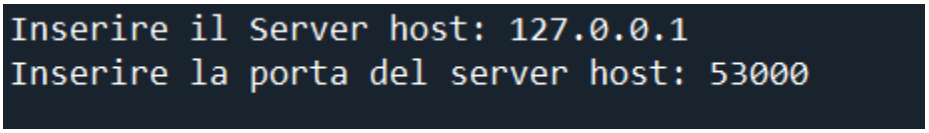
```
def send(event=None):
    # gli eventi vengono passati dai binders.
    msg = my_msg.get()
    # libera la casella di input.
    my_msg.set("")
    # invia il messaggio sul socket
    try:
        client_socket.send(bytes(msg, "utf8"))
    except OSError:
        msg_list.insert(tkt.END,"Necessaria una chiusura!")
    finally:
        if msg == "{quit}":
            client_socket.close()
            finestra.destroy()
```

Infine, abbiamo la funzione **on_closing** che verrà chiamata quando scegliamo di chiudere la finestra dalla GUI. Serve per ripulire, prima della chiusura dell'applicazione, e rilasciare, tramite l'aiuto della *send*, la socket utilizzata nella connessione.

```
def on_closing(event=None):
    my_msg.set("{quit}")
    send()
```

GUIDA D'USO

Per avviare la chat è necessario, prima di tutto avviare/lanciare in esecuzione lo script del server su un nuovo terminale. Il server a questo punto si metterà in attesa di eventuali client che si vogliono connettere. Successivamente si apre un nuovo terminale per il client che si vuole connettere alla chat e bisogna digitare le informazioni riguardanti il server. L'indirizzo del server è **127.0.0.1**, che è consigliato digitare, mentre per quanto riguarda il numero di porta a cui il client deve fare riferimento è la **53000** (il numero di porta può essere omesso).



Dopo aver inserito i dati richiesti e aver premuto invio comparirà la schermata iniziale dove sarà richiesto all'utente di digitare un nome che lo identificherà all'interno della chat dove saranno presenti altri utenti. Per uscire dalla chat, l'utente può o digitare {quit} (che comunicherà agli altri utenti la sua intenzione di uscire) o premere l'icona della GUI, in questo caso l'uscita è forzata.

Sul terminale del server saranno visualizzate delle informazioni riguardo chi è entrato o uscito dalla chat.

Per quanto riguarda la chiusura del server, bisogna chiudere il terminale sul quale è stato lanciato, e nel caso siano ancora presenti degli utenti, quest’ultimi saranno avvisati di tale evenienza.