Sistema di Chat Client-Server

AUTORE:

Relazione dell'elaborato che realizza la soluzione della traccia numero 1 a cura di **Latini Luca** (matricola: **0001091148**)

RICHIESTA:

implementare un sistema di chat client-server in Python utilizzando socket programming. Il server deve essere in grado di gestire più client contemporaneamente e deve consentire agli utenti di inviare e ricevere messaggi in una chatroom condivisa. Il client deve consentire agli utenti di connettersi al server, inviare messaggi alla chatroom e ricevere messaggi dagli altri utenti.

INTRODUZIONE

Il progetto consiste nella realizzazione di un server di chat multithread e di un client corrispondente, utili per l'implementazione di un sistema di chat asincrona. L'applicazione server consente connessioni multiple da parte di diversi client, gestendo le comunicazioni tra di loro in modo parallelo e asincrono. Il client è sviluppato con una GUI in Tkinter per offrire agli utenti un'interfaccia semplice e intuitiva per partecipare alla chat.

REQUISITI E DIPENDENZE

Per eseguire correttamente il server e il client, è necessario avere installato **Python 3**. I moduli richiesti sono:

- `socket`: per la gestione delle comunicazioni di rete.

  ```
  from socket import AF_INET, socket, SOCK_STREAM
  ```

- `threading`: per la gestione delle connessioni in un contesto multithreading.

  ```
  from threading import Thread
  ```

- `logging`: per la gestione e registrazione degli eventi e degli errori. (lato server)

  ```
  import logging
  ```

- `tkinter`: per la GUI del client.

  ```
  import tkinter as tk
  ```

- `datetime`: per ottenere e formattare timestamp nei messaggi di chat.

  ```
  from datetime import datetime
  ```

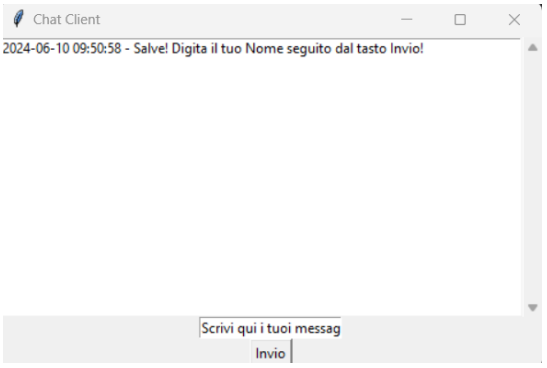
GUIDA ALL'UTILIZZO

- **Esecuzione del Server**
  1. Salva il codice server in un file Python, ad esempio `server.py`.
  2. Esegui il server utilizzando il comando: `python server.py`
  3. Il server inizierà l'ascolto per le connessioni in entrata sulla porta definita (53000 di default).

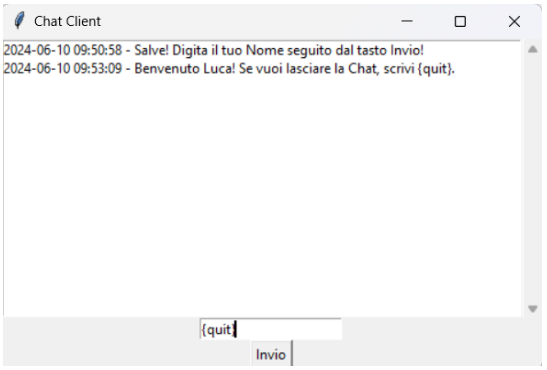
```
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep
11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 8.15.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Luca/OneDrive/Desktop/
chat_server.py', wdir='C:/Users/Luca/OneDrive/Desktop')
In attesa di connessioni...
```

- **Esecuzione del Client**
  1. Salva il codice client in un file Python, ad esempio `client.py`.
  2. Esegui il client utilizzando il comando: `python client.py`. *in caso di più client si necessita l'apertura di una nuova console ed eseguire il codice nuovamente*
  3. Nella finestra di login, inserisci l'indirizzo IP del server **127.0.0.1** e la porta **53000(default)**.
  4. Clicca su "**Connetti**" per avviare la connessione al server e accedere alla chat.

  5. Inserire come primo messaggio il nome con cui si vuole apparire all'interno della chat



  6. Per uscire dalla chat scrivere il messaggio "*{quit}*"



FUNZIONAMENTO

- **Funzionamento del Server**

  Il server si avvia in ascolto delle connessioni in entrata. Per ogni connessione accettata, viene creato un nuovo thread che gestisce la comunicazione con il singolo client. Questo include il ricevimento e l'invio di messaggi, gestione di disconnessioni e broadcast dei messaggi agli altri client connessi.

  Principali Funzioni del Server:

  1. **accetta_connessioni_in_entrata():**
     - Questa funzione è responsabile di accettare le nuove connessioni in entrata dal client.
     - Utilizza il metodo `SERVER.accept()` per accettare la connessione di un nuovo client.
     - Quando un nuovo client si connette, viene stampato un messaggio che indica l'indirizzo IP e la porta del client connesso.

       ```
       In attesa di connessioni...
       2024-06-10 10:03:10,685 - INFO - Connessione da
       ('127.0.0.1', 53936)
       2024-06-10 10:03:32,213 - INFO - Connessione da
       ('127.0.0.1', 53976)
       ```

     - Viene inviato un messaggio di benvenuto al nuovo client, invitandolo a inserire il suo nome.
     - Il nuovo client viene registrato nel dizionario `indirizzi` con il suo socket e indirizzo.
     - Viene avviato un nuovo thread per gestire la connessione del client, chiamando la funzione `gestice_client()`.
  2. **gestice_client(client):**
     - Questa funzione gestisce la comunicazione con un singolo client connesso.
     - Riceve il socket del client come argomento.
     - Legge il nome inviato dal client e lo registra nel dizionario `clients`.
     - Invia un messaggio di benvenuto al client, informandolo su come uscire dalla chat.
     - Invia un messaggio broadcast a tutti i client connessi, informandoli che il nuovo utente si è unito alla chat.
     - Entra in un ciclo di ricezione e invio di messaggi:
       - Riceve i messaggi inviati dal client utilizzando `client.recv(BUFSIZ)`.
       - Se il messaggio non è "{quit}", lo invia in broadcast a tutti i client connessi.
       - Se il messaggio è "{quit}", invia la conferma di uscita al client, lo rimuove dal dizionario `clients` e invia un messaggio broadcast che l'utente ha abbandonato la chat.
  3. **broadcast(msg, prefisso=""):**
     - Questa funzione invia un messaggio in broadcast a tutti i client connessi.

- Itera attraverso il dizionario `clients` e invia il messaggio a ciascun client connesso.
- Se si verifica un errore durante l'invio del messaggio (ad esempio, se un client si è disconnesso), il client viene rimosso dal dizionario `clients` e viene registrato l'errore nel log.

- **Funzionamento del Client**

  Il client offre un'interfaccia grafica per la chat, sviluppata con Tkinter. Gli utenti possono connettersi al server inserendo l'indirizzo IP e la porta e, una volta connessi, possono inviare e ricevere messaggi.

  Principali Funzioni del Client:

  1. **connect_to_server():**
     - Questa funzione gestisce la connessione del client al server.
     - Legge l'indirizzo IP e la porta del server dall'interfaccia grafica.
     - Crea un socket client utilizzando `socket(AF_INET, SOCK_STREAM)`.
     - Tenta di connettersi al server utilizzando `client_socket.connect(ADDR)`.
     - In caso di successo, chiude la finestra di login e avvia la finestra della chat chiamando `show_chat_window()`.
     - In caso di errore, visualizza un messaggio di errore nella finestra di login.
  2. **show_chat_window():**
     - Questa funzione crea e configura l'interfaccia grafica della chat utilizzando Tkinter.
     - Crea il frame per visualizzare i messaggi, con una barra di scorrimento.
     - Crea il campo di input per l'utente e il pulsante di invio.
     - Associa la funzione `send()` all'evento di pressione del tasto Invio.
     - Associa la funzione `on_closing()` all'evento di chiusura della finestra.
     - Avvia un thread separato chiamando `receive()` per gestire la ricezione dei messaggi dal server.
     - Avvia il ciclo principale di Tkinter per gestire gli eventi dell'interfaccia grafica.
  3. **receive():**
     - Questa funzione viene eseguita in un thread separato per ricevere i messaggi dal server.
     - Utilizza client_socket.recv(BUFSIZ)

STRUTTURA DELL'APPLICAZIONE

- **Struttura del Server:**

  struttura del server si concentra principalmente sulla gestione delle connessioni in entrata, la comunicazione con i singoli client, l'invio di messaggi in broadcast e la gestione delle disconnessioni. Queste sono le funzionalità chiave del server per implementare una chat room multi-utente.

  1. **importazioni configurazione del logging**

     ```
     from socket import AF_INET, socket, SOCK_STREAM
        from threading import Thread
        import logging
        logging.basicConfig(level=logging.INFO, format='%(asctime)s -
        %(levelname)s - %(message)s')
     ```

  2. **funzioni Principali (accetta_connessioni_in_entrata, gestice_client, broadcast)**

     ogni funzione prevede l'implementazione della gestione degli errori e delle eccezioni

```
    def accetta_connessioni_in_entrata():
    """Accetta le connessioni dei client in entrata."""
    while True:
        try:
            client, client_address = SERVER.accept()
        except OSError:
            logging.warning("Server chiuso.")
            break
        except Exception as e:
            logging.error(f"Errore durante l'accettazione della connessione: {e
            continue
        logging.info(f"Connessione da {client_address}")
        client.send(bytes("Salve! Digita il tuo Nome seguito
        dal tasto Invio!", "utf8"))

        indirizzi[client] = client_address
        Thread(target=gestione_client, args=(client,)).start()

def gestione_client(client):
    """Gestisce la connessione di un singolo client."""
    nome = None
    try:
        nome = client.recv(BUFSIZ).decode("utf8")
        benvenuto = f'Benvenuto {nome}! Se vuoi lasciare la Chat,
         scrivi {{quit}}.'

        client.send(bytes(benvenuto, "utf8"))
        msg = f"{nome} si è unito alla chat!"
        broadcast(bytes(msg, "utf8"))
        clients[client] = nome
        while True:
            try:
                msg = client.recv(BUFSIZ)
                if msg == bytes("{quit}", "utf8"):
                    client.send(bytes("{quit}", "utf8"))
                    client.close()
                    del clients[client]
                    broadcast(bytes(f"{nome} ha abbandonato la Chat.",
                     "utf8"))
                    break
                else:
                    broadcast(msg, nome + ": ")
            except (OSError, ConnectionResetError):
                break
    except Exception as e:
        logging.error(f"Errore durante la gestione del client: {e}")
    finally:
        if client in clients:
            if nome:
                broadcast(bytes(f"{nome} ha abbandonato la Chat.", "utf8"))
            del clients[client]
        client.close()

def broadcast(msg, prefisso=""):
```

```
    """Invia un messaggio in broadcast a tutti i client."""
    for utente in clients:
        try:
            utente.send(bytes(prefisso, "utf8") + msg)
        except (BrokenPipeError, ConnectionResetError) as e:
            logging.error(f"Errore durante l'invio del
            messaggio in broadcast: {e}")

            utente.close()
```

3. **Configurazione del Server e Avvio**

```
clients = {}
indirizzi = {}

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

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

if __name__ == "__main__":
    try:
        SERVER.listen(5)
        print("In attesa di connessioni...")
        ACCEPT_THREAD = Thread(target=accetta_connessioni_in_entrata)
        ACCEPT_THREAD.start()
        ACCEPT_THREAD.join()
        SERVER.close()
    except  Exception as e:
        print("Errore generale:", e )
    finally:
       SERVER.close()
```

ogni funzione presentata in uno dei tre punti è consultabile ,con commenti dettagliati, all'interno del codice `chat_server.py` , inserito nella repository

- **Struttura del Client:**

  La struttura del client si concentra principalmente sulla creazione dell'interfaccia grafica utilizzando Tkinter, la gestione della connessione al server e l'invio/ricezione di messaggi. Le funzioni `connect_to_server` e `show_chat_window` sono i punti di ingresso principali per l'esecuzione del client.

1. Importazioni e Definizione delle Funzioni

```
    from socket import AF_INET, socket, SOCK_STREAM
    from threading import Thread
    import tkinter as tk
    from datetime import datetime

   def receive():
    while True:
        try:
```

```
            msg = client_socket.recv(BUFSIZ).decode("utf8")
            if msg:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                formatted_msg = f"{timestamp} - {msg}"
                msg_list.insert(tk.END, formatted_msg)
            else:
                break
        except OSError as e:
            if str(e) == "[Errno 9] Bad file descriptor":
                break
            else:
                print(f"Errore di ricezione: {e}")
                break

def send(event=None):
    """Sends messages."""
    msg = my_msg.get()
    my_msg.set("")
    try:
        client_socket.send(bytes(msg, "utf8"))
        if msg == "{quit}":
            client_socket.close()
            finestra.quit()
    except OSError:
        pass

def on_closing(event=None):
    my_msg.set("{quit}")
    send()
    try:
        client_socket.close()
    except OSError:
        pass
    finestra.quit()
    finestra.destroy()

def show_chat_window():
    global finestra, my_msg, msg_list
    finestra = tk.Tk()
    finestra.title("Chat Client")


    messages_frame = tk.Frame(finestra)
    my_msg = tk.StringVar()
    my_msg.set("Scrivi qui i tuoi messaggi.")
    scrollbar = tk.Scrollbar(messages_frame)
    msg_list = tk.Listbox(messages_frame, height=15, width=75,
        yscrollcommand=scrollbar.set)

    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    msg_list.pack(side=tk.LEFT, fill=tk.BOTH)
    msg_list.pack()
    messages_frame.pack()

    # Creazione del campo di input per i messaggi
```

```
    entry_field = tk.Entry(finestra, textvariable=my_msg)
    entry_field.bind("<Return>", send)
    entry_field.pack()

    # Creazione del pulsante di invio
    send_button = tk.Button(finestra, text="Invio", command=send)
    send_button.pack()

    finestra.protocol("WM_DELETE_WINDOW", on_closing)

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

    tk.mainloop()

def connect_to_server():
    global client_socket, BUFSIZ, login_window
    HOST = host_entry.get()
    PORT = port_entry.get()
    if not PORT:
        PORT = 53000
    else:
        PORT = int(PORT)
    BUFSIZ = 1024
    ADDR = (HOST, PORT)  #

    client_socket = socket(AF_INET, SOCK_STREAM)
    try:
        client_socket.connect(ADDR)
        login_window.destroy()
        show_chat_window()
    except Exception as e:
        error_label.config(text=f"Errore di connessione: {e}")
```

2. Creazione della Finestra di Login

```
    login_window = tk.Tk()
    login_window.title("Login Chat Client")

    tk.Label(login_window, text="inserisci Server Host:").pack(pady=10)
    host_entry = tk.Entry(login_window)
    host_entry.pack(pady=10)

    tk.Label(login_window, text="inserisciPortadelserver :").pack(pady=10)
    port_entry = tk.Entry(login_window)
    port_entry.pack(pady=10)

    error_label = tk.Label(login_window, text="", fg="red")
    error_label.pack(pady=10)

    connect_button = tk.Button(login_window, text="Connetti",
    command=connect_to_server)
    connect_button.pack(pady=20)
```

```
    login_window.mainloop()
```

ogni funzione presentata nei 2 punti è consultabile ,con commenti dettagliati ,all'interno del codice `client.py` inserito nella repository