DOCUMENTAZIONE FUNZIONAMENTO DEL SISTEMA

Il sistema permette a più clienti di comunicare tra loro via un server che deve realizzare il broadcast dei messaggi inviati da un client ad altri clients.

Per poterlo eseguire bisogna prima eseguire il server **server.py**. Una volta eseguito, sarà in attesa di connessioni da parte di più clients sulla porta **“5001”**. Poi, si deve eseguire **“client.py”** su più computer; dopo di che i clients si connetteranno al server all’indirizzo **"127.0.0.1"** e alla porta **“5001”**.

A. Il server(server.py)

Questo file contiene le seguenti parti:

- Una parte principale per gestire l’inizializzazione del server

```
if __name__ == "__main__":
    # Creazione del socket per il server
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((SERVER_HOST, SERVER_PORT))
    server.listen(NUM)
    print("Server waiting for connections...")
    ACCEPT_THREAD = Thread(target=receive_connections()).start()
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    server.close()
```

- La funzione receive_connections() che riceve le richieste di connessione dei client e stabilisce la connessione tra client e server

```
def receive_connections():
    while True:
        client_socket, client_address = server.accept()
        print(f"Connection from {client_address} has been established.")
        #aggiunge il nuovo client che si connette alla lista dei client connessi
        clients.append(client_socket)
        client_socket.send(bytes("Hello, connection to the server ok!","utf-8"))
        #diamo inizio all'attività del Thread - uno per ciascun client
        Thread(target=manage_client, args=(client_socket,)).start()
```

- La funzione manage_client(client_socket) gestisce la ricezione dei messaggi inviati dai clients e coordina il broadcasting

```
def manage_client(client_socket):
    while True:
```

```
        try:
            message = client_socket.recv(BUFFER_SIZE).decode('utf-8')
            if message:
                print(f"Received message: {message}")
                broadcast(message, client_socket)
            else:
                remove(client_socket)
                break
        except:
            continue
```

- La funzione broadcast caricato di condivedere i messaggi inviati da un client con gli altri clients

```
def broadcast(message, client_socket):
    # Invia il messaggio agli altri client
    for client in clients:
        if client != client_socket:
            try:
                client.send(message.encode('utf-8'))
            except:
                #Rimuove il client nel caso l'invio non andasse
                #a buon fine
                remove(client)
```

- La funzione remove(client) permette di rimuovere un client dalla lista dei clients connessi

```
def remove(client):
    if client in clients:
        clients.remove(client)
```

B. Il client (client.py)

Questo file contiene le seguenti parti:

- Una parte principale per l'inizializzazione del client

```
if __name__ == "__main__":
    # Creazione del socket del client
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((SERVER_HOST, SERVER_PORT))  # Connessione al server

    # Avvio dei thread per ricevere e inviare messaggi
    receive_thread = Thread(target=receive)
    receive_thread.start()

    send_thread = Thread(target=send)
    send_thread.start()
```

- La funzione send() che permette di inviare i messaggi al server

```
def send():
    # gestisce l'invio dei messaggi
    while True:
        message = input('')
        client_socket.send(bytes(message, "utf8"))
```

- La funzione receive() che riceve messaggi dal server

```
def receive():
    # gestisce la ricezione di messaggi
    while True:
        try:
            message = client_socket.recv(BUFFER_SIZE).decode('utf-8')
            if message:
                print(message)
        except:
            print("An error occurred!")
            client_socket.close()
            break
```

➢ Connessione al server

-il client prova a stabilire una connessione col server

```
client_socket.connect((SERVER_HOST, SERVER_PORT))
```

-il server accetta la connessione

```
client_socket, client_address = server.accept()
```

-il server poi aggiunge quel client ad una lista che mantiene traccia sui clients connessi al server ed attiva il suo Thread

```
clients.append(client_socket)
```

```
Thread(target=manage_client, args=(client_socket,)).start()
```

➢ Comunicazione con invio di messaggi

Dopo essersi connesso al server, quando un client manda un messaggio con la funzione **“send”**, la funzione **“manage_client(client_socket)”** del server recupera il messaggio e lo condivide con gli altri clients connessi mediante la funzione **“broadcast”** e quindi grazie alla

funzione **“reveice()”** di ogni client che li consente di ricevere i messaggi dal server inviati. Durante questa procedura, se il server non riesce a recuperare il messaggio di un client o non riesce a condividere un messaggio ricevuto con un client, significa che ha perso la connessione con quest’ultimo che viene quindi rimosso dalla lista dei clients connessi con la funzione **“remove(client)”** del server.