

Relazione Progetto Programmazione di Reti

Chat client-server

Marcatelli Francesco
0001078124

Link: https://github.com/framarcaa/Progetto_Reti_Chat

1 Sviluppo del server

Il server è configurato per utilizzare una connessione TCP ed è gestito da un thread principale che rimane in attesa di connessioni dai client. Il server è configurato sull'indirizzo di loopback "127.0.0.1", noto anche come localhost, e utilizza la porta "60000". Quando un client si connette, viene avviato un nuovo thread dedicato alla gestione della ricezione dei pacchetti da quel client.

```
def server_manager():
    SERVER.bind(ADDRESS)
    SERVER.listen(5)
    print("In attesa di connessioni...")
    while True:
        client, client_address = SERVER.accept()
        print("%s:%s si è connesso." % client_address)
        client.send(bytes("Benvenuto in chatroom! Inserisci il tuo nome!",FORMAT))
        Thread(target=client_manager, args=(client,client_address)).start()
```

Il server è responsabile di trasmettere in broadcast il messaggio appropriato a tutti i client collegati che sono in modalità di ascolto.

```
def broadcast(msg, prefisso=""):
    for utente in clients:
        utente.send(bytes(prefisso, FORMAT)+msg)
```

2 Sviluppo del client

Una volta avviato, il client crea un thread principale che gestisce la connessione al server. Dopo che la connessione è stata stabilita, viene avviato un thread separato che attende di ricevere pacchetti dal server. Se il server non è disponibile o non risponde, viene visualizzato un messaggio che lo indica. Se la connessione viene interrotta, il thread responsabile della ricezione dei pacchetti viene chiuso e il thread principale tenta di ristabilire la connessione.

```
try:
    message = client_socket.recv(BUFSIZE).decode(FORMAT)
    main_text.insert(tkt.END, message)
except ConnectionResetError:
    print("Connessione al server interrotta.")
    break
except Exception:
    pass
```

```
def send(message):
    try:
        client_socket.send(bytes(message, FORMAT))
    except:
        if message != CLOSING:
            main_text.insert(tkt.END, SERVER_OFFLINE)
```

3 Gestione errori di connessione

Nell'implementazione del server, si gestisce l'eccezione "ConnectionResetError", che si verifica quando la connessione con il client viene interrotta in modo involontario. In questo caso, viene registrato l'errore e il thread responsabile viene chiuso.

```
try:
    msg = client.recv(BUFSIZE)
    if msg == bytes("{close}", FORMAT):
        close_client(client, ("%s:%s si è disconnesso." % client_address), name)
        break
    broadcast(msg, name+": ")
except ConnectionResetError:
    close_client(client, ("%s:%s si è disconnesso per un problema di connessione." % client_address), name)
    break
```

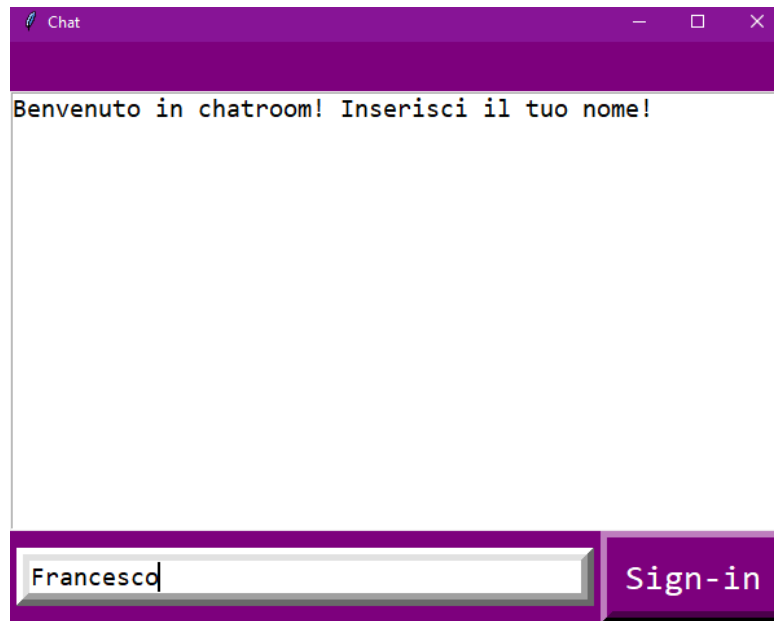
Nel codice del client, vengono affrontati gli errori di "ConnectionResetError" che indicano interruzioni di connessione inaspettate. Inoltre, sono gestite eccezioni generiche durante la ricezione e l'invio di pacchetti, così come durante la connessione al server.


```
try:
    message = client_socket.recv(BUFSIZE).decode(FORMAT)
    main_text.insert(tkt.END, message)
except ConnectionResetError:
    print("Connessione al server interrotta.")
    break
except Exception:
    pass
```

4 Applicazione

Il sistema è formato da un server e uno o più client che stabiliscono una connessione con il server tramite un socket TCP. Una volta avviato, il server entra in uno stato di attesa per accettare le connessioni in arrivo.

L'utente avvia il programma che aprirà la schermata di chat. Il primo passaggio richiesto all'utente è quello di inserire il proprio username e premere il bottone "Sign-in".



Una volta fatto il bottone cambierà nel bottone "Enter" adibito nell'invio dei messaggi. Per uscire dal programma basterà chiudere la finestra tramite la  in alto a destra

