UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
REDES DE COMPUTADORES

VINÍCIUS CABRAL LIMA GUERRA DE ALCÂNTARA
ANA MARIA CARDOSO WAGNER
JOÃO PEDRO SIMÕES
EDUARDO VIEIRA DA SILVA

PROJETO DE REDES: CHAT ROOM COM PYTHON

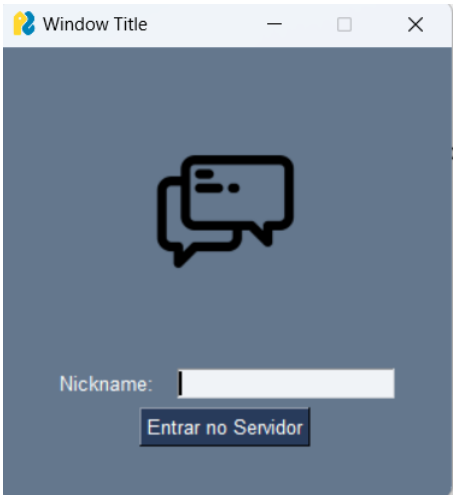
MACEIÓ 2024

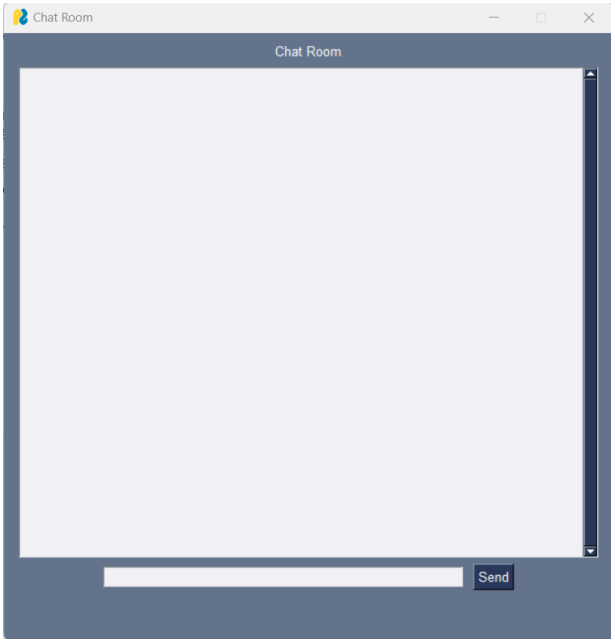
SUMÁRIO

1. INTRODUÇÃO

O projeto feito é a implementação de um “Chat Room” que consiste em uma sala online de bate papo em que os clientes, ao acessarem, podem se comunicar um com o outro em tempo real. Essa comunicação é gerenciada pelo servidor da aplicação que se responsabiliza por lidar com cada cliente e fazer o broadcast de suas mensagens. O protocolo de comunicação utilizado foi o TCP.

A interface da sala de bate papo foi implementada com a biblioteca PySimpleGUI, além disso foram utilizadas as bibliotecas socket e threading para uso respectivamente de sockets e threads.



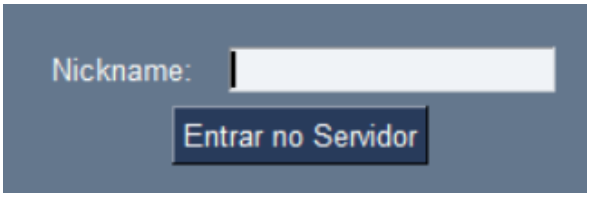


2. FUNCIONALIDADES

Temos duas funcionalidades principais na aplicação: Registro de nicknames e a comunicação em tempo real.

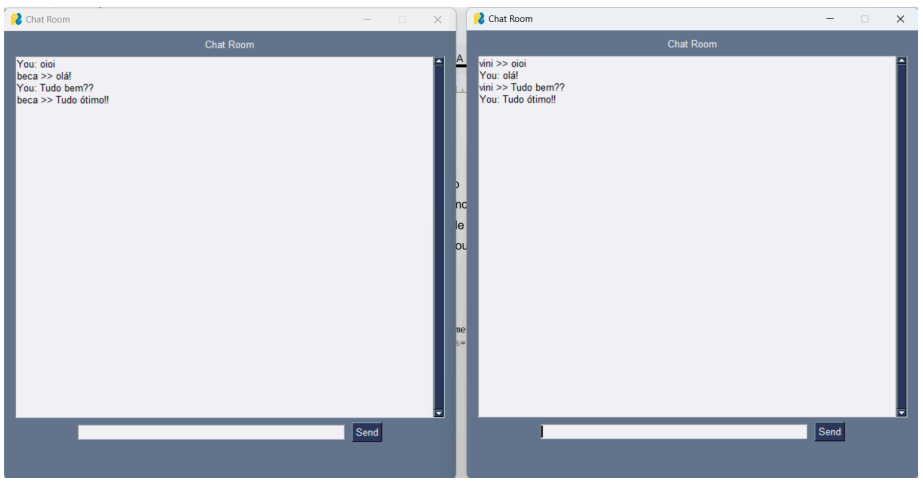
2.1. REGISTRO DE NICKNAMES

A tela de início da aplicação fornece a possibilidade de registro de um nickname para um cliente. Após a escolha de um nome o cliente pode entrar na sala de bate papo clicando em “Entrar no servidor”.



2.2 COMUNICAÇÃO EM TEMPO REAL

Para que a comunicação em tempo real ocorra, cada cliente tem duas threads. Uma thread fica responsável por monitorar o momento em que o cliente enviar alguma mensagem para a sala de bate papo e a outra thread fica responsável por receber as mensagens de outros clientes e exibir na interface do primeiro cliente.



3. FUTURAS IMPLEMENTAÇÕES

Será interessante implementar, futuramente, as seguintes funcionalidades:

- Criação de uma sala de bate papo com um administrador associado.
- Poder de administrador: excluir, banir e mutar usuários da sala de bate papo.
- Criação de conta com senha para os usuários da aplicação.
- formação de vínculos de amizade e de grupo entre os usuários.

Atualmente a aplicação não possui capacidade de persistir os dados e por isso essas funcionalidades ainda não podem ser implementadas. O acréscimo dessa capacidade poderá tornar a aplicação bem interessante.

4. PROTOCOLOS UTILIZADOS

Temos dois protocolos principais que foram utilizados:

- Protocolo TCP: O TCP garante a comunicação confiável entre os usuários da sala de bate papo. Utilizando a biblioteca socket (um wrapper das APIs de socket derivadas dos BSD socket) foi feito o uso desse protocolo utilizando poucos métodos fornecidos pelo objeto socket.

- Protocolo personalizado da aplicação: A aplicação utiliza um protocolo de aplicação personalizado, implementado sobre o TCP, para definir como as mensagens são enviadas e recebidas. As mensagens são codificadas e decodificadas em strings, permitindo, por exemplo, o envio de mensagens de texto. O protocolo também inclui mensagens especiais, como "EXIT", que são usadas para controlar o fluxo e a desconexão dos clientes de forma organizada.

5. DIFICULDADES

Ocorreram 4 momentos de dificuldades na construção do projeto:

5.1. SOCKETS E THREADS

- Socket: Demorou um pouco para compreender que um cliente fica representado como um objeto de socket no lado do servidor.
- Thread: A necessidade de uma instância de Thread para cada cliente não foi de fácil intuição no início

5.2. DUAS THREADS NO LADO DO CLIENTE

Foram necessárias duas threads no lado do cliente para fazer a aplicação. Perceba que a interface do cliente, uma sala de bate papo possui dois fluxos de informação: um fluxo de chegada de mensagens e um fluxo de envio de mensagens.

Para cada um desses fluxos são necessárias duas threads. Uma das threads fica responsável por ouvir as mensagens de broadcast do servidor e exibi-las na tela do cliente. A outra Thread fica responsável por enviar as mensagens do cliente para o servidor quando o cliente clica em “Send”.

A necessidade dessas duas threads não era clara no início.

5.3. IMPOSSIBILIDADE DE MONITORAR A INTERFACE FORA DA THREAD MAIN

Inicialmente, foi utilizada a ideia de enviar o objeto “window” do PySimpleGUI para ambas as threads de envio e recebimento de mensagens e dentro de cada thread realizar o monitoramento dos eventos que ocorreriam da interação do usuário com a interface.

Isso não foi possível pois existe uma restrição inerente à biblioteca. No PySimpleGUI, não é possível monitorar os eventos que ocorrem na “window” fora da main thread.

Esse problema foi contornado da seguinte forma: Obedecemos a restrição de monitorar os eventos da “window” na thread principal da seguinte forma:

- **Thread de envio de mensagens**: Toda vez que ocorre o evento “Send” a mensagem a ser enviada é colocada em uma fila para que possa ser processada na thread de envio de mensagem.

- **Thread de recebimento de mensagens**: Toda vez que o cliente recebe uma mensagem do servidor (pelo broadcast) é criado uma chave de evento dinâmica “-RECEIVE-” no objeto de interface do PySimpleGUI (Isso é uma prática comum em sistemas que utilizam interfaces gráficas reativas). Dessa forma o monitoramento da main thread é capaz de monitorar a chegada de novas mensagens e exibi-las na tela.

Talvez surja, no leitor, o questionamento: como a thread de envio de mensagens tem acesso à fila da thread main, ou então, como a thread de recebimento de mensagens consegue alterar o objeto “window” que se encontra na main thread. Lembre-se de que as threads compartilham a mesma área de memória do processo de que fazem parte, nesse caso, o processo client.py.

5.4. ENCERRAMENTO INCORRETO DO PROCESSO CLIENT.PY

O encerramento das threads não estava sendo realizado. Por conta disso ocorriam erros ao fechar a janela de interface do cliente. Portanto, foi utilizado um objeto da classe “Event” da biblioteca threading para encerrar o loop das threads quando o evento de fechamento de janela fosse acionado pelo usuário.

Sem esse tratamento surgiam erros no terminal ou o processo continuava a executar mesmo com a interface fechada.

6 CÓDIGO FONTE

O código fonte está disponível em um repositório do github nesse link. Apenas os arquivos server.py, client.py e as classes das interfaces utilizadas foram coladas aqui.

6.1 SERVER.PY

```
import os
import socket
from threading import Thread

PROCESS_ID = os.getpid()
HOST = "127.0.0.1"
PORT = 65432
MAX_CLIENTS = 10

# The clients dictionary will store the client's connection as the key
# and the client's nickname and address as the value in a dictionary.
clients = {}

def broadcast_message(message, sender_conn):
    sender_nickname = clients[sender_conn]['nickname']
    encoded_message = f'{sender_nickname} >> {message}'.encode()
    for conn in clients:
        if conn != sender_conn:
            conn.sendall(encoded_message)



def handle_client(conn, addr):
    with conn:
        print(f"New client connected: {addr}")

        nickname = conn.recv(1024).decode()
        clients[conn] = {'nickname': nickname, 'address': addr}
        print(f'Client: {addr} is now aka: {nickname}')


        while True:
            try:
```

```
                message = conn.recv(1024).decode()
                if message:
                    print(f"{nickname} says: {message}")
                    broadcast_message(message, conn)
                else:
                    break
            except:
                break

        print(f"Client {addr} disconnected")
        del clients[conn]
        conn.close()


if __name__ == "__main__":
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        print(f"Process ID: {PROCESS_ID}")

        s.bind((HOST, PORT))
        s.listen()
        print("Server is listening...")

        while True:
            conn, addr = s.accept()
               client_thread = Thread(target=handle_client, args=(conn,
addr))
            client_thread.start()
```

6.2 CLIENT.PY

```
import socket
import os
import PySimpleGUI as sg
from client_interface import ClientInterface, ChatRoomInterface
from client_utils import (
    send_message,
    receive_message,
    set_nickname
)
```

```
from threading import Thread, Event
from queue import Queue
import queue
from server import HOST, PORT

PROCESS_ID = os.getpid()

def send(s: socket.socket, message_queue: Queue, stop_event: Event):
     while not stop_event.is_set():
        try:
            message = message_queue.get(timeout=1)
            if message == "EXIT":
                break
            send_message(s, message)
        except queue.Empty:
            continue

def  receive(s:  socket.socket,  chat_room_interface:  ChatRoomInterface,
stop_event: Event):
    while not stop_event.is_set():
        try:
            message = receive_message(s)
            if message:

chat_room_interface.window.write_event_value('-RECEIVE-', message)
        except (ConnectionAbortedError, ConnectionResetError, OSError):
            break

def client_interface(s: socket.socket):
    client_interface = ClientInterface()
    if set_nickname(s, client_interface) == None:
        return

    chat_room_interface = ChatRoomInterface()
    message_queue = Queue()
    stop_event = Event()

          send_thread   =   Thread(target=send,   args=(s,   message_queue,
stop_event))
                receive_thread    =    Thread(target=receive,    args=(s,
chat_room_interface, stop_event))

    send_thread.start()
```

```
    receive_thread.start()

    while True:
        event, values = chat_room_interface.window.read()

        if event == 'Send':
            message = values['-INPUT-']
                      chat_room_interface.window['-CHAT-'].update(f'You:
{message}\n', append=True)
            chat_room_interface.window['-INPUT-'].update('')
            message_queue.put(message)

        if event == sg.WIN_CLOSED:
            message_queue.put("EXIT")
            stop_event.set()
            break

        if event == '-RECEIVE-':
            message = values['-RECEIVE-']
             chat_room_interface.window['-CHAT-'].update(f'{message}\n',
append=True)


    s.close()
    send_thread.join()
    receive_thread.join()
    chat_room_interface.window.close()


def start_client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        print(f"Process ID: {PROCESS_ID}")

        s.connect((HOST, PORT))
        print("Connected to the server...")

        client_interface(s)

if __name__ == "__main__":
    start_client()
```

6.3 CLIENT_INTERFACE.PY

```
import PySimpleGUI as sg
from threading import Thread
import socket


class ChatRoomInterface:

    def __init__(self):
        self.layout = [
            [sg.Text('Chat Room')],
            [sg.Multiline(size=(80, 30), key='-CHAT-')],
            [sg.Input(size=(50, 1), key='-INPUT-'), sg.Button('Send')],
        ]
        self.window = sg.Window(
            title='Chat Room',
            size=(600, 600),
            layout=self.layout,
            element_justification='center',
        )

class ClientInterface:

    def __init__(self):
        self.layout = [
                [sg.Image(filename='./projeto_de_redes/images/chat.png',
size=(200, 200))],
            [sg.Text('Nickname: '), sg.Input(size=(20, 1))],
            [sg.Button(button_text='Entrar no Servidor')],
        ]
        self.window = sg.Window(
            title='Window Title',
            size=(300, 300),
            layout=self.layout,
            element_justification='center',
        )
```