Alunos: Matheus Augusto, Tiago Gorri, Germano Lagana

Requisitos Funcionais

RF1: O sistema deve permitir que o usuário coloque o seu nome para entrar no chat. O nome não pode ter espaço e nem “@”.

RF2: O sistema deve bloquear a entrada de alguém com o mesmo nome de um usuário já cadastrado, e pedir para que seja utilizado outro nome.

RF3: Todos os usuários conectados recebem a mensagem quando um novo usuário entra no chat, incluindo IP e porta.

RF4: O sistema deve permitir que o chat seja em tempo real, utilizando threads.

RF5: O sistema deve permitir que os usuários possam enviar mensagens no chat para que todos possam ver

RF6: O sistema permite mensagens privadas usando "@" para direcionar a mensagem a um destinatário específico.

RF7: O sistema não pode permitir que o usuário envie uma mensagem privada para si mesmo

RF8: O sistema deve enviar uma mensagem para todos avisando quando um usuário sair do chat.

RF9: O cliente é removido do chat ao clicar em "Sair".

RF10: O sistema deve permitir que, mesmo que um usuário tenha saído do chat, o sistema siga funcionando, fazendo com que outros usuários possam entrar.

RF11: Caso o cliente tente conectar sem o servidor estar ativo, aparece um pop-up com uma mensagem de erro.

RF12: O IP e a porta do cliente são exibidos para o próprio usuário e para os demais.

Sockets

Servidor:

- Aqui, é criado um socket de servidor usando a família de endereços AF_ INET (IPv4) e o tipo de socket SOCK_STREAM (TCP).

```
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Cria o socket TCP
```

- O servidor é vinculado a um endereço IP (HOST) e uma porta (PORT) específicos. Isso significa que ele escutará conexões nessa combinação de endereço e porta.

```
servidor.bind((HOST, PORT))  # Associa o socket ao endereço e porta
```

- O servidor começa a escutar conexões de entrada. O argumento 5 especifica o número máximo de conexões pendentes que podem ser enfileiradas antes de serem aceitas.

```
servidor.listen(5)  # Coloca o socket em modo de escuta (até 5 conexões pendentes)
```

- O servidor chama o método accept() para aceitar uma conexão de um cliente. Isso cria um novo socket específico para a comunicação com esse cliente

```
cliente_socket, endereco = servidor.accept()  # Aceita uma nova conexão
```

- Dentro da função handle_cliente(), o servidor usa o método recv() para receber dados do cliente. O servidor pode receber mensagens do cliente, como o nome de usuário ou mensagens de chat.

```
nome_usuario = cliente_socket.recv(1024).decode()
```

Cliente:

- A conexão cliente-servidor no sistema de chat é baseada no protocolo TCP/IP e utiliza sockets para estabelecer comunicação.

- A primeira linha tenta estabelecer uma conexão com o servidor usando o endereço IP (HOST) e a porta (PORT) especificados. A segunda linha tenta estabelecer uma conexão com o servidor usando o endereço IP (HOST) e a porta (PORT) especificados. A terceira linha está enviando o nome de usuário para o servidor logo após a conexão ser estabelecida.

```
socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_cliente.connect((HOST, PORT))  # Conecta ao servidor
socket_cliente.sendall(nome_usuario.encode())  # Envia o nome de usuário para o servidor
```

- O cliente também pode receber dados do servidor, como as mensagens de outros usuários ou uma lista de usuários ativos.

```
mensagem = socket_cliente.recv(1024).decode()  # Recebe mensagens do servidor
```

Broadcast

- O servidor possui uma função chamada broadcast que é responsável por enviar uma mensagem para todos os clientes conectados, exceto o remetente (aquele que enviou a mensagem original). Isso é útil para mensagens de grupo, como quando um cliente entra ou sai do chat, ou para mensagens gerais enviadas por qualquer participante.

```
def broadcast(mensagem, cliente_socket):
    """Envia uma mensagem para todos os clientes conectados."""
    with lock:
        # Itera sobre todos os clientes conectados
        for cliente in clientes:
            # Verifica se o cliente atual não é o remetente da mensagem
            if cliente != cliente_socket:
                try:
                    # Envia a mensagem para o cliente
                    cliente.sendall(mensagem.encode())
                except:
                    # Fecha o socket do cliente em caso de erro
                    cliente.close()
                    with lock:
                        # Remove o cliente com erro do dicionário
                        if cliente in clientes:
                            del clientes[cliente]
```

Unicast

O unicast no servidor permite que os usuários enviem mensagens privadas diretamente a outro participante do chat, utilizando o símbolo @ seguido do nome do destinatário (@nome_destinatario mensagem).

```python
def enviar_unicast(destinatario, mensagem, remetente_socket):
    """Envia uma mensagem privada ao destinatário e exibe no remetente também."""
    with lock:
        # Obtém o nome do remetente
        remetente_nome = clientes[remetente_socket]
        # Procura pelo destinatário no dicionário de clientes
        for cliente_socket, nome in clientes.items():
            if nome == destinatario:  # Verifica se encontrou o destinatário
                try:
                    # Envia a mensagem para o destinatário
                    cliente_socket.sendall(mensagem.encode())
                    # Envia uma cópia da mensagem para o remetente
                    mensagem_remetente = f"{remetente_nome} (privado para @{destinatario}): {mensagem.split(': ', 1)[1]}"
                    remetente_socket.sendall(mensagem_remetente.encode())
                except Exception:
                    pass
                return  # Sai da função após enviar a mensagem
        # Caso o destinatário não seja encontrado, envia um erro para o remetente
        remetente_socket.sendall(f"Usuário {destinatario} não encontrado.\n".encode())
```

Threads

Servidor

No servidor, os threads permitem que ele atenda múltiplos clientes simultaneamente. Cada vez que um cliente se conecta, um novo thread é criado para gerenciar a comunicação com esse cliente específico. A função handle_cliente roda em um thread separado para cada cliente, permitindo que o servidor continue aceitando novas conexões e distribuindo mensagens em tempo real para todos os participantes do chat.

```python
# Configuração do servidor
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Cria um socket TCP
servidor.bind(("192.168.1.12", 9999))  # Vincula o socket ao IP e porta do servidor
servidor.listen(5)  # Define o número máximo de conexões pendentes
print(
    f"Servidor iniciado em {servidor.getsockname()}"
)  # Exibe a mensagem de inicialização do servidor

while True:
    # Aceita uma nova conexão de cliente
    cliente_socket, endereco_cliente = servidor.accept()
    print(
        f"Nova conexão de {endereco_cliente}"
    )  # Exibe o endereço do cliente no console

    # Cria uma thread para gerenciar o cliente
    thread = threading.Thread(
        target=handle_cliente, args=(cliente_socket, endereco_cliente)
    )  # Passa o socket e o endereço do cliente como argumentos
    thread.daemon = True  # Define a thread como daemon
    thread.start()  # Inicia a thread
```

```python
def handle_cliente(cliente_socket, endereco_cliente):
    """Gerencia as mensagens recebidas de um cliente."""
    try:
        # Recebe o nome do usuário enviado pelo cliente
        nome_usuario = cliente_socket.recv(1024).decode()

        with lock:
            # Verifica se o nome do usuário já está em uso
            if nome_usuario in clientes.values():
                # Envia uma mensagem de erro para o cliente caso o nome esteja em uso
                cliente_socket.sendall(
                    "Erro: Já existe um usuário ativo com esse nome.\n".encode()
                )
                cliente_socket.close()  # Fecha a conexão com o cliente
                return  # Sai da função

            # Adiciona o cliente ao dicionário de clientes conectados
            clientes[cliente_socket] = nome_usuario
            # Log dos clientes conectados para o console do servidor
            print(
                f"[Servidor] Clientes conectados: {[clientes[s] for s in clientes.keys()]}"
            )

        # Captura o IP e a porta do cliente
        ip, porta = endereco_cliente
        # Formata a mensagem de entrada do cliente com IP e porta
        mensagem_entrada = f"{nome_usuario} entrou no chat. (IP: {ip}, Porta: {porta})"

        # Envia uma mensagem de boas-vindas com IP e porta para o próprio cliente
        cliente_socket.sendall(
            f"Bem-vindo, {nome_usuario}! Seu IP: {ip}, Porta: {porta}\n".encode()
        )
```

```python
    # Envia uma mensagem de confirmação de conexão ao servidor
    cliente_socket.sendall("Conectado ao servidor.\n".encode())
    # Envia a lista de usuários ativos para o cliente
    enviar_usuarios_ativos(cliente_socket)

    # Envia a mensagem de entrada para todos os outros clientes
    broadcast(mensagem_entrada, cliente_socket)
    # Exibe a mensagem de entrada no console do servidor
    print(mensagem_entrada)

    # Loop para receber mensagens do cliente
    while True:
        # Aguarda e recebe mensagens enviadas pelo cliente
        mensagem = cliente_socket.recv(1024).decode()
        if mensagem:
            # Exibe a mensagem recebida no console do servidor
            print(f"Recebido de {nome_usuario}: {mensagem}")

            # Verifica se o cliente enviou uma mensagem indicando que saiu do chat
            if mensagem.lower() == f"{nome_usuario} saiu do chat.":
                # Formata a mensagem de saída do cliente
                mensagem_saida = (
                    f"{nome_usuario} saiu do chat. (IP: {ip}, Porta: {porta})"
                )
                # Envia a mensagem de saída para todos os outros clientes
                broadcast(mensagem_saida, cliente_socket)
                with lock:
                    # Remove o cliente do dicionário de clientes conectados
                    if cliente_socket in clientes:
                        del clientes[cliente_socket]
                        print(f"[Servidor] Cliente {nome_usuario} removido.")
                break  # Sai do loop
```

```
            # Verifica se a mensagem é privada (começa com "@")
            if mensagem.startswith("@"):
                # Divide a mensagem para obter o destinatário e o conteúdo
                destinatario, mensagem_unicast = mensagem[1:].split(" ", 1)
                # Envia a mensagem privada para o destinatário
                enviar_unicast(
                    destinatario,
                    f"{nome_usuario} (privado): {mensagem_unicast}",
                    cliente_socket,
                )
            else:
                # Envia a mensagem como broadcast para todos os outros clientes
                broadcast(f"{nome_usuario}: {mensagem}", cliente_socket)
    except (ConnectionResetError, BrokenPipeError):
        # Trata a desconexão inesperada do cliente
        print(f"Conexão perdida com {clientes.get(cliente_socket, 'desconhecido')}")
    finally:
        # Remove o cliente da lista ao desconectar
        with lock:
            if cliente_socket in clientes:
                # Envia uma mensagem de saída para os outros clientes
                broadcast(f"{clientes[cliente_socket]} saiu do chat.", cliente_socket)
                print(f"[Servidor] Removendo cliente {clientes[cliente_socket]}")
                # Remove o cliente do dicionário
                del clientes[cliente_socket]
        try:
            cliente_socket.shutdown(socket.SHUT_RDWR)  # Fecha a conexão completamente
        except Exception:
            pass  # Ignora erros ao fechar o socket
        cliente_socket.close()  # Fecha o socket do cliente
```

Cliente

No cliente, um thread separado é usado para receber mensagens do servidor em tempo real, enquanto o usuário continua interagindo com a interface gráfica sem interrupções. A thread executa a função recebe_mensagens(), que exibe as mensagens recebidas na interface assim que elas chegam, mantendo a comunicação fluida e responsiva.

```python
def enviar_mensagem():
    """Função para enviar mensagem quando o botão é clicado ou o Enter é pressionado."""
    global socket_cliente, nome_usuario
    mensagem = (
        entrada_mensagem.get().strip()
    )  # Obtém a mensagem digitada pelo usuário e remove espaços desnecessários
    if mensagem:  # Verifica se a mensagem não está vazia
        try:
            if mensagem.startswith("@"):  # Detecta mensagem privada
                destinatario = mensagem.split(" ")[0][1:]  # Extrai o destinatário
                if (
                    destinatario == nome_usuario
                ):  # Verifica se o destinatário é o próprio usuário
                    lista_chat.config(state=tk.NORMAL)
                    lista_chat.insert(
                        tk.END,
                        "Você não pode enviar mensagens privadas para si mesmo.\n",
                        "erro",
                    )  # Exibe erro na interface
                    lista_chat.config(state=tk.DISABLED)
                    entrada_mensagem.delete(0, tk.END)  # Limpa o campo de entrada
                    return  # Sai da função

            else:
                lista_chat.config(state=tk.NORMAL)
                lista_chat.insert(
                    tk.END, f"Você: {mensagem}\n", "enviado"
                )  # Mostra a mensagem enviada na interface
                lista_chat.config(state=tk.DISABLED)
```

```python
            else:
                lista_chat.config(state=tk.NORMAL)
                lista_chat.insert(
                    tk.END, f"Você: {mensagem}\n", "enviado"
                )  # Mostra a mensagem enviada na interface
                lista_chat.config(state=tk.DISABLED)

            socket_cliente.sendall(mensagem.encode())  # Envia a mensagem ao servidor
            entrada_mensagem.delete(0, tk.END)  # Limpa o campo de entrada
            lista_chat.yview(tk.END)  # Rola a interface para mostrar a mensagem enviada
        except Exception as e:
            lista_chat.config(state=tk.NORMAL)
            lista_chat.insert(
                tk.END, f"Erro ao enviar mensagem: {e}\n", "erro"
            )  # Mostra erro caso ocorra
            lista_chat.config(state=tk.DISABLED)
```

- As linhas abaixo são responsáveis por iniciar um thread que ficará encarregada de receber mensagens do servidor enquanto o cliente está interagindo com ele.

```
# Inicia uma thread para receber mensagens
thread_recebe = threading.Thread(target=recebe_mensagens)
thread_recebe.daemon = True
thread_recebe.start()
```