PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

CONECTIVIDADE EM SISTEMAS CIBERFISÍCOS

PROJETO CHAT

MEMBROS DA EQUIPE

Anna Quézia dos Santos,

Deborah Santos Lizardo

SUMÁRIO

1. REQUISITOS FUNCIONAIS

ID	DESCRIÇÃO	CONDIÇÕES	TRATAMENTO DE ERROS
RF001	O sistema deve enviar uma mensagem de boas-vindas (broadcast) ao conectar: "O cliente <nome_usuario> entrou no chat dos Little Cats."	Mensagem exibida a todos os clientes conectados.	**Erro de Envio:** Se ocorrer erro ao enviar a mensagem para algum cliente, o sistema registrará o erro e continuará tentando enviar para os outros clientes.
RF002	O sistema deve permitir o envio de mensagens privadas para um usuário específico (unicast).	Usuário destinatário deve ser identificado pelo seu nickname.	**Erro de Envio:** Se o destinatário não for encontrado ou ocorrer erro na conexão do cliente, o sistema retornará uma mensagem de erro informando o problema.
RF003	O servidor deve exibir o IP e a porta de cada cliente ao conectar no formato: <nome_usuario> se conectou com o IP <ip_cliente> por meio da porta <porta_cliente>.	IP deve estar no formato IPv4 válido. Porta atribuída dinamicamente pelo sistema.	**Erro de Conexão:** Se o servidor não conseguir identificar o IP ou porta de um cliente, o servidor exibirá uma mensagem de erro e não permitirá a conexão.
RF004	O sistema deve enviar uma mensagem de saída (broadcast) quando um cliente sai: **"<nome_usuario> saiu do chat dos Little Cats."**	Mensagem exibida a todos os clientes conectados.	**Erro de Envio:** Se ocorrer erro ao enviar a mensagem de saída, uma mensagem de erro será registrada e o cliente será removido da lista de clientes conectados.
RF005	O sistema deve validar o nickname na conexão: deve ser único e não vazio.	Nicknames vazios não são permitidos.	**Erro de Validação:** Caso o nickname seja vazio, o sistema solicitará que o cliente forneça um nome.
RF006	O servidor deve permitir ao cliente	Lista exibida com nicknames ativos.	**Erro de Listagem:** Se não houver clientes conectados é

	visualizar a lista de usuários conectados.		exibido uma mensagem informando ao usuário isso.
RF007	O sistema deve exibir ao destinatário o nickname do remetente ao enviar mensagens privadas.	Destinatário deve identificar claramente o remetente.	**Erro de Envio:** Se não houver clientes conectados ou erro ao enviar a lista, o servidor exibirá uma mensagem de erro adequada.
RF009	O usuário deve fornecer um nickname correto ao servidor para se conectar ao chat. O sistema deve validar o nickname na conexão inicial.	Se o remetente informar um nickname inválido ou incorreto ao enviar uma mensagem privada, o sistema deve retornar uma mensagem de erro ao remetente também.	**Erro de Envio:** Se ocorrer erro ao enviar a mensagem privada, o sistema retornará um erro indicando falha na entrega da mensagem.

2. A FORMA COMO FOI UTILIZADO OS SOCKETS

Os sockets são um dos componentes essenciais para permitir a comunicação entre sistemas em rede, e no contexto do nosso projeto, foram utilizados para permitir a troca de mensagens entre o servidor e os clientes em um chat. O código implementa uma arquitetura cliente-servidor, onde o servidor se encarrega de gerenciar a comunicação entre múltiplos clientes, enquanto os clientes interagem com o servidor para enviar e receber mensagens.

No servidor, o socket foi configurado para usar o protocolo TCP, que garante uma comunicação confiável. O servidor primeiro cria um socket do tipo SOCK_STREAM (que corresponde ao TCP) e o associa ao endereço IP e à porta na qual ficará aguardando conexões. Para isso, o servidor utiliza a função bind(), que vincula o socket ao endereço -**substituir-** e à porta **-substituir-**. Uma vez feito isso, o servidor começa a ouvir as solicitações de conexão usando o método listen(), que coloca o socket em estado de escuta. O servidor, então, aceita a conexão de novos clientes com a função accept(), o que cria um novo socket de comunicação dedicado a cada cliente conectado. Este processo permite que o servidor se comunique com múltiplos clientes de forma independente, criando canais de comunicação conforme necessário.

Sendo assim, quando um cliente se conecta ao servidor, a conexão é estabelecida por meio de um socket específico, através do qual o servidor pode enviar e receber mensagens do cliente. O código de criação do socket do servidor é o seguinte:

```
sock_server = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
sock_server.bind((HOST, PORT))
sock_server.listen(5)
```

Do lado do cliente, o processo é similar, mas a principal diferença é que, enquanto o servidor espera as conexões, o cliente inicia a conexão com o servidor. O cliente cria um socket com o mesmo tipo de protocolo (TCP) e se conecta ao servidor utilizando o método connect(). Após a conexão ser estabelecida, o cliente pode então enviar e receber mensagens, e pode fornecer um *nickname* para se identificar no chat. O código no cliente que realiza essa conexão é o seguinte:

```
sock_client = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
sock_client.connect((HOST, PORT))
sock_client.sendall(nickname.encode())
```

Esses sockets são responsáveis pela comunicação contínua entre o servidor e o cliente, usando os métodos sendall() para enviar mensagens e recv() para receber dados. Eles permitem que tanto o servidor quanto os clientes enviem e recebam dados de forma síncrona, garantindo que o chat funcione em tempo real.

3. TRATAMENTO DE BROADCAST NO TIPO DE CONEXÃO UTILIZADO

No contexto do sistema de chat, o broadcast é utilizado para garantir que as mensagens enviadas por um cliente sejam recebidas por todos os outros clientes conectados ao servidor. A principal vantagem do broadcast é criar um ambiente em que todos os usuários podem ver as interações uns dos outros, promovendo uma experiência de comunicação em grupo.

No código do servidor, o broadcast é implementado de forma simples e eficaz. Quando o servidor recebe uma mensagem de um cliente, ele percorre a lista de todos os clientes conectados e envia a mesma mensagem a cada cliente, exceto ao remetente, para evitar que o cliente veja sua própria mensagem novamente. Esse processo é feito dentro de uma iteração que passa por todos os clientes da lista, e a mensagem é enviada para cada um usando o método sendall(). O código que realiza o broadcast da mensagem é o seguinte:

```
for client in clients:
    if client['connection'] != sock_conn:
        client['connection'].sendall(f"{nickname}:
{message}".encode())
```

O código acima percorre a lista de clientes conectados e envia a mensagem para todos, com a exceção do cliente que a enviou. Esse tipo de comunicação é fundamental para que todos os participantes do chat possam acompanhar o que está sendo discutido em tempo real, sem a necessidade de interação direta com um destinatário específico.

Além das mensagens de texto, o servidor também pode enviar mensagens de entrada e saída de clientes para todos os usuários, garantindo que todos saibam quando alguém entrou ou saiu do chat. Por exemplo, quando um novo cliente entra no chat, o servidor envia uma mensagem de boas-vindas para todos os outros clientes conectados, informando que o novo usuário entrou no chat. O mesmo ocorre quando um cliente se desconecta: uma mensagem é enviada para os demais clientes notificando sua saída.

Em síntese, o tratamento de broadcast ajuda a manter todos os participantes do chat informados e engajados com o que está acontecendo no chat em tempo real, promovendo a interação entre os usuários.

4. EXPLICAÇÃO DAS THREADS NO CLIENTE E NO SERVIDOR

4.1. THREADS NO CLIENTE

No cliente, uma thread separada é usada para ouvir mensagens do servidor enquanto o usuário pode interagir livremente com o sistema, enviando mensagens. Isso evita que o cliente fique bloqueado esperando por respostas, permitindo a interação contínua. A função receive_messages () é executada em um loop contínuo para receber

e exibir as mensagens do servidor, enquanto a função send_message() permite que o cliente envie mensagens sem interrupção.A thread do cliente que escuta mensagens foi criada da seguinte forma:

```
receive_thread = threading.Thread(target=receive_messages)
receive_thread.start()
```

4.2 EXPLICAÇÃO DAS THREADS NO SERVIDOR

No servidor, cada novo cliente que se conecta é gerido por uma thread separada, permitindo que o servidor continue aceitando novas conexões sem ser bloqueado pelo processamento de mensagens. A criação de uma thread para cada cliente é feita com a função threading.Thread(), que executa a função handle_client() de forma independente para cada cliente. Isso garante que o servidor possa processar as interações de um cliente (como enviar mensagens públicas ou privadas, e realizar broadcasts) enquanto aguarda a conexão de novos clientes cliente para atender a outro, tornando a comunicação assíncrona e eficiente. No nosso código, a criação de thread para novo cliente foi:. Assim, o servidor não precisa esperar a conclusão de um

```
def start_server():
    sock_server = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
    sock_server.bind((HOST, PORT))
    sock_server.listen(5)
    print_server_started()  # Inicia a escuta do servidor


    while True:
        try:
            sock_conn, ender = sock_server.accept()  # Aceita uma nova conexão de cliente
            client_thread = threading.Thread(target=handle_client, args=(sock_conn, ender,
clients))
            client_thread.start()  # Cria uma nova thread para atender o cliente
        except sock.error as e:
            print(f"Error accepting connection: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")
```