Projeto CHAT – Conectividade de Sistemas Ciberfísicos

Alunos: Cesar Yoshio Tsushima Neto, Matheus Muller e Vitor Vieira

---

Requisitos:

RF1: O sistema do chat deve ser capaz de mostrar na tela de todos os clientes o nome de quem se conectou ao chat.
RF2: O sistema deve permitir que um cliente envie uma mensagem privada (UNICAST) para outro cliente conectado, digitado "/w" e em seguida o nome do cliente que deve receber a mensagem.
RF3: O sistema deve permitir que os clientes enviem mensagens para todos os outros clientes conectados ao chat.
RF4: O sistema deve ser capaz de atualizar e mostrar em tempo real a lista de clientes conectados.
RF5: O sistema deve mostrar na tela do chat de todos os cliente que um cliente foi conectado ou desconectado.
RF6:O sistema deve alertar o remetente se o destinatário de uma mensagem privada (unicast) não for encontrado entre os usuários conectados.
RF7: No chat do servidor deve ter informações adicionais de conexão de cada cliente, como endereço IP e porta, além de notificações sobre conexões e desconexões.

Uso dos sockets

Os sockets possuem um papel essencial para estabelecer a comunicação entre as partes. Os sockets permitem que os clientes enviem mensagens ao servidor, que então redistribui essas mensagens para os outros clientes conectados, em caso de broadcast ou para um destinatário específico, em caso de Unicast.

- **No servidor:**

```
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.bind(("0.0.0.0", 5555))  # Aceita conexões de qualque
servidor.listen()
print("[SERVIDOR INICIADO] Aguardando conexões...")
```

onde:

1. socket.AF_INET, define o uso dos endereçamentos IPv4
2. socket.SOCKET_STREAM, Indica o uso do protocolo TCP, que é confiável e garante a entrega das mensagens.
3. servidor.bind(("0.0.0.0", 5555)), o servidor é vinculado a todos os IPs disponíveis na máquina, na porta 5555, o endereço IP utilizado neste momento é apenas um exemplo. No momento de realizar os testes e conectar os computadores, o IP será substituído pelo endereço da máquina que atuará como servidor.
4. servidor.listen(), Prepara o servidor para aceitar conexões.

- **No cliente:**

```
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define o IP padrão do servidor
servidor_ip = "192.168.0.100"  # Substitua pelo IP do servidor
try:
    cliente.connect((servidor_ip, 5555))
```

onde:

1. “socket.AF_INET” e “socket.SOCK_STREAM”, são usados para criar um socket TCP
5. cliente.connect((servidor_ip, 5555)), Estabelece a conexão com o servidor no endereço IP e porta especificados. o endereço IP utilizado neste momento é apenas um exemplo. No momento de realizar os testes e conectar os computadores, o IP será substituído pelo endereço da máquina que atuará como servidor.

Tratamento de Broadcast

O tratamento de mensagens broadcast, é uma funcionalidade que permite que um cliente possa enviar mensagens simultâneas para todos os outros clientes conectados dentro do servidor

```
def enviar_mensagem(msg, cliente_remetente):
    for cliente, nome in clientes_conectados:
        if cliente != cliente_remetente:
            try:
                cliente.send(msg.encode("utf-8"))
            except Exception as e:
                print(f"[ERRO] Falha ao enviar mensagem para {
                cliente.close()
                clientes_conectados.remove((cliente, nome))
```

O socket do cliente que enviou a mensagem, que será excluído do broadcast para evitar que receba a própria mensagem.

A lista clientes_conectados contém pares (socket_cliente, nome_cliente) de todos os clientes ativos

Se o cliente atual da iteração **não** for o remetente (if cliente != cliente_remetente), ele receberá a mensagem.

Tratamento Unicast

O tratamento de mensagens Unicast, é uma funcionalidade que permite que um cliente possa enviar mensagens para um único cliente logado no servidor

```
if msg.startswith("/w "):
    # Formato de mensagem unicast: "/w <nome_destinatario> <me
    partes = msg.split(" ", 2)
    if len(partes) == 3:
        nome_destinatario = partes[1]
        mensagem_unicast = f"{nome} sussurra: {partes[2]}"
        for cliente, nome_cliente in clientes_conectados:
            if nome_cliente == nome_destinatario:
                cliente.send(mensagem_unicast.encode("utf-8")
                socket_cliente.send(mensagem_unicast.encode("u
                break
        else:
            socket_cliente.send(f"[ERRO] Usuário '{nome_destin
    else:
        socket_cliente.send("[ERRO] Formato de unicast inválid
```

Um cliente envia uma mensagem com o comando /w, seguido pelo nome do destinatário e a mensagem

A mensagem é identificada como unicast no trecho

if msg.startswith("/w "):

Uso das Threads

O uso de threads no sistema de chat é essencial para gerenciar múltiplas conexões de clientes simultaneamente no servidor e permitir que o cliente receba mensagens do servidor enquanto continua enviando outras. As threads permitem que diferentes partes do programa sejam executadas de forma concorrente, sem bloquear o fluxo principal.

- **No servidor:** as threads são usadas para lidar com cada cliente individualmente. Cada conexão de cliente é gerenciada em uma thread separada, o que permite que o servidor atenda vários clientes ao mesmo tempo.

```
while True:
    socket_cliente, endereco = servidor.accept()
    thread_cliente = threading.Thread(target=lidar_com_cli
    thread_cliente.start()
    print(f"[CONEXÕES ATIVAS] {threading.active_count() -
```

onde:

1. A função “threading.Thread” cria uma nova thread para executar a função “lidar_com_cliente”.
2. Os argumentos “socket_cliente” e endereco são passados para a função “lidar_com_cliente”.
3. O parâmetro “daemon=True” garante que a thread seja encerrada automaticamente quando o programa principal terminar.

- **No cliente:** as threads são usadas para receber mensagens do servidor de forma contínua, enquanto o usuário pode continuar digitando e enviando mensagens.

```
def receber_mensagens(socket_cliente):
    while True:
        try:
            msg = socket_cliente.recv(1024).decode("utf-8"
            print(msg)  # Exibe a mensagem recebida do ser
        except:
            print("[DESCONECTADO] A conexão com o servidor
            socket_cliente.close()
            break
```

onde:

1. A função “receber_mensagens” é executada em uma thread separada e fica em loop contínuo para receber mensagens do servidor.
2. A thread é criada para executar a função “receber_mensagens”, permitindo que ela funcione paralelamente ao envio de mensagens pelo cliente.