Projeto Conectividade de Sistemas Ciberfísicos

Alunos: Vinicius Padilha, Arthur Cidral, Murilo Regnier

Requisitos

RF1: O sistema deve mostrar na tela de chat de todos os usuários o nome do novo participante que acabou de se conectar ao chat.

RF2: O sistema deve permitir que um usuário envie uma mensagem privada para outro usuário conectado, digitando `@nome_destinatario` antes da mensagem.

RF3: O sistema deve permitir que os clientes enviem mensagens para todos os usuários conectados ao chat.

RF4: O sistema deve atualizar e exibir em tempo real a lista de usuários conectados para todos os participantes sempre que um usuário sair ou for desconectado do chat.

RF5: O sistema deve atualizar e exibir em tempo real a lista de usuários conectados para todos os participantes sempre que um usuário se conectar no chat.

RF6: O sistema deve enviar uma notificação para todos os usuários conectados informando o nome do participante que saiu do chat.

RF8: O sistema deve informar ao remetente caso o destinatário de uma mensagem privada (unicast) não seja encontrado na lista de usuários conectados.

RF9: O servidor deve exibir no chat do servidor informações adicionais de conexão de cada cliente, como endereço IP e porta, além de notificações sobre conexões e desconexões.

Explicação do uso dos Sockets

No servidor:

- O servidor utiliza sockets TCP para aceitar conexões de múltiplos clientes e gerenciar a comunicação bidirecional.
- O servidor cria um socket TCP com `sock.AF_INET` (IPv4) e `sock.SOCK_STREAM` (TCP), vincula-o ao IP e à porta especificados e inicia a escuta por conexões.

```
94    sock_server = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
95    sock_server.bind((HOST, PORTA))
96    sock_server.listen() # permite que vários clientes se conectem ao servidor.
```

- Para cada cliente que tenta se conectar, o servidor aceita a conexão e cria um novo socket dedicado (`sock_conn`), que será usado para comunicação com aquele cliente.

```
100     sock_conn, ender = sock_server.accept() # Para cada novo cliente que se conecta, o servidor aceita a conexão (sock_server.accept()) e inicia
        uma thread dedicada para atender esse cliente, chamando a função recebe_dados
```

No cliente:

- O cliente utiliza um socket TCP para se conectar ao servidor e enviar/receber mensagens.

- O cliente cria um socket TCP e se conecta ao servidor usando o IP e a porta configurados.

```
socket_cliente = sock.socket(sock.AF_INET, sock.SOCK_STREAM) # criação de socket para se conectar com o servidor
# Cliente solicita conexão com servidor
socket_cliente.connect((HOST, PORTA))
```

Tratamento de Broadcast

O broadcast é a funcionalidade que permite que uma mensagem seja enviada para todos os clientes conectados, exceto o remetente.

```
def broadcast(mensagem, remetente=None): #evita que o remetente receba sua mensagem
    for cliente in lista_clientes: # percorre todos os clientes conectados
        if cliente != remetente: #verifica se o cliente atual do loop não é o remetente
            try:
                cliente[0].sendall(mensagem.encode()) # cliente[0] representa o socket do cliente
                # sendall(mensagem.encode()) converte a mensagem em bytes e a envia através do socket do cliente
                # sendall() garante que a mensagem inteira será enviada
            except:
                remover(cliente) #chamado para remover o cliente problemático da lista_clientes
```

- A função percorre a lista de clientes conectados (`lista_clientes`) e envia a mensagem para cada cliente.

```
    # Notificar todos sobre o novo cliente e atualizar lista de conectados
    broadcast(f"{nome} entrou no chat.")
    atualizar_lista_conectados()
```

- Exemplo de quando um cliente entra no chat.

Tratamento de Unicast

O unicast permite o envio de mensagens privadas de um cliente para outro cliente conectado.

```
# Função para envio de mensagens privadas (unicast)
def unicast(mensagem, remetente_socket, destinatario_nome, remetente_nome):
    # mensagem: conteudo da mensagem a ser enviado
    # remetente_socket: socket do remetente que está enviando a mensagem
    # destinatario_nome: o nome do cliente que deverá receber a mensagem
    # remetente_nome: o nome do cliente que está enviando a mensagem, para exibir ao destinatário
    for cliente in lista_clientes:
        if cliente[1] == destinatario_nome:
            try:
                cliente[0].sendall(f"[Privado] {remetente_nome} >> {mensagem}".encode())
                # cliente[0] representa o socket do destinatário
                # sendall() envia a mensagem convertida em bytes com .encode()

                remetente_socket.sendall(f"[Privado para {destinatario_nome}] {mensagem}".encode())
                # envia a mensagem para o socket do remetente
                # informa o remetente que a mensagem foi enviada para o (destinatario_nome), junto com o conteudo da mensagem
                return
            except:
                remover(cliente)
    remetente_socket.sendall(f"Usuário {destinatario_nome} não encontrado.".encode())
```

- A função percorre a lista de clientes conectados (`lista_clientes`) e verifica se o nome do destinatário (`destinatario_nome`) corresponde a um cliente conectado.
- Caso o destinatário seja encontrado, a mensagem é enviada para o destinatário e uma confirmação é enviada ao remetente.
- Se o destinatário não for encontrado, o remetente é notificado.

Uso de Threads

Servidor:

Cada cliente conectado ao servidor é gerenciado por uma thread dedicada, permitindo comunicação simultânea com múltiplos clientes.

Criação de Thread para cada cliente:

```
101         thread_cliente = threading.Thread(target=recebe_dados, args=[sock_conn, ender])
102         thread_cliente.start()
```

- Cada nova conexão aceita pelo servidor inicia uma thread que executa a função `recebe_dados(sock_cliente, endereco)`.

Cliente:

O cliente utiliza uma thread separada para receber mensagens do servidor enquanto a interface gráfica permanece responsiva.

Criação de Thread para receber mensagens:

```
70          # Iniciar thread para receber mensagens
71          thread_receber = threading.Thread(target=recebe_mensagens)
72          thread_receber.daemon = True
73          thread_receber.start()
```

- A função `recebe_mensagens()` é executada em uma thread separada para que o recebimento de mensagens não bloqueie a interface gráfica.

Validação e remoção de clientes

Clientes desconectados ou que causam erro são removidos da lista de clientes conectados.

```
43  # Função para remoção de clientes da lista
44  def remover(cliente):
45      if cliente in lista_clientes:
46          lista_clientes.remove(cliente)
47          broadcast(f"{cliente[1]} saiu do chat.") # após a remoção o servidor envia uma mensagem broadcast informando quem saiu
48          atualizar_lista_conectados() #é chamada, e nela é exibido os users conectados
```

- A função remove o cliente problemático da lista (`lista_clientes`) e notifica os demais sobre sua saída.

Atualização de lista de usuários

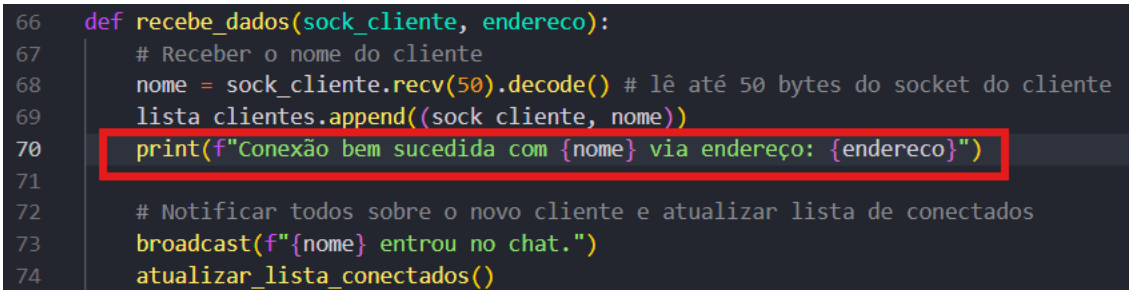
A lista de usuários conectados é atualizada em tempo real e enviada para todos os clientes sempre que alguém entra ou sai do chat.

```
52  # Função para atualizar e enviar a lista de clientes conectados a todos os clientes
53  # função que é chamada acima na funçao remover()
54  def atualizar_lista_conectados():
55      clientes_conectados = "Usuários conectados: " + ", ".join([cliente[1] for cliente in lista_clientes])
56      for cliente in lista_clientes:
57          try:
58              cliente[0].sendall(clientes_conectados.encode())
59          except:
60              remover(cliente)
```

- A função compila os nomes dos clientes conectados e envia a lista atualizada para todos os clientes.

Informações de Conexão no Servidor

O servidor registra no console o endereço IP e a porta de cada cliente conectado.



```
66  def recebe_dados(sock_cliente, endereco):
67      # Receber o nome do cliente
68      nome = sock_cliente.recv(50).decode() # lê até 50 bytes do socket do cliente
69      lista_clientes.append((sock_cliente, nome))
70      print(f"Conexão bem sucedida com {nome} via endereço: {endereco}")
71
72      # Notificar todos sobre o novo cliente e atualizar lista de conectados
73      broadcast(f"{nome} entrou no chat.")
74      atualizar_lista_conectados()
```