PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

Conectividade em Sistemas Ciberfísicos

Relatório Projeto Final - CHAT

Membros:

Gustavo Yuri Ferreira Imoto - 40095029

Leonardo Martineli - 40093508

Curitiba

2024

Relatório

No código, os sockets foram utilizados para permitir a comunicação entre o cliente e o servidor por meio do protocolo TCP/IP.

```
87    sock_server = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
88    sock_server.bind((HOST, PORTA))
89    sock_server.listen()
```

O servidor utiliza o socket para aceitar conexões TCP (com AF_INET e SOCK_STREAM), utilizando o endereço e porta especificados o AF_INET quer dizer que o protocolo é o IPV4 e o SOCK_STREAM que é TCP.

```
        cliente_socket, endereco = sock_server.accept()
```

Esse método aguarda que um cliente inicie uma conexão. Cada cliente recebe um socket exclusivo (cliente_socket) que permite o envio e recebimento de mensagens entre o servidor e esse cliente.

```
            cliente.sendall(mensagem.encode())
```

```
nome = cliente_socket.recv(1024).decode().strip()
```

O servidor utiliza métodos como sendall (para enviar dados) e recv (para receber dados), tanto no envio de mensagens para clientes individuais quanto no broadcast.

```
11    socket_cliente = sock.socket(sock.AF_INET, sock.SOCK_STREAM)
12    socket_cliente.connect((HOST, PORTA))
```

Aqui, o socket do cliente se conecta ao endereço e porta do servidor, estabelecendo uma conexão bidirecional.

```
mensagem = socket_cliente.recv(1024).decode()
```

```
socket_cliente.sendall(mensagem.encode())
```

O cliente utiliza o método sendall para enviar mensagens ao servidor e o método recv para receber mensagens.

Broadcast

```python
def broadcast(mensagem, remetente=None):
    print(f"[Broadcast] Enviando mensagem: {mensagem}")
    with lock:
        for cliente, nome in clientes.items():
            try:
                cliente.sendall(mensagem.encode())
            except Exception as e:
                print(f"Erro ao enviar mensagem para {nome}: {e}")
                cliente.close()
                remove_cliente(cliente)
```

Para o broadcast foi criado uma função dentro do código do servidor, ou seja o tratamento é feito no servidor, chamada broadcast que recebe como parâmetro a mensagem e o remetente, e dentro da função é criado um for que percorre o dicionário clientes e envia a mensagem para cada cliente dentro desse dicionário e se caso ocorrer algum erro ao enviar uma mensagem, o cliente problemático é removido com remove_cliente, Como múltiplas threads podem tentar modificar a lista de clientes simultaneamente, o uso de um lock garante que o acesso à lista seja seguro.

Uso de Threads

```python
thread_receber = threading.Thread(target=receber_mensagens, daemon=True)
thread_receber.start()
```

No cliente uma thread foi criada para executar a função receber_mensagens, essa thread roda em paralelo ao loop principal da interface gráfica, escutando mensagens enviadas pelo servidor e atualizando a interface com elas. Isso permite que o cliente envie e receba mensagens sem que a interface fique travada.

```python
thread_cliente = threading.Thread(target=recebe_dados, args=(cliente_socket, endereco))
thread_cliente.start()
```

No servidor cada cliente conectado gera uma nova thread que executa a função recebe_dados, essas threads permitem que o servidor gerencie simultaneamente a comunicação com vários clientes. Cada thread é responsável por: Receber mensagens do cliente, processar as mensagens (privadas ou broadcasts), remover o cliente se ele se desconectar. Temos também a Thread principal do servidor que tem um loop principal responsável por aceitar conexões de clientes.

Lista de Requisitos

RF-001	O sistema deve imprimir na tela de todos os usuários conectados, incluindo a tela do servidor, o nome do usuário que acabou de entrar no servidor, no instante em que isso ocorrer.
RF-002	O sistema deve permitir o envio de mensagens do tipo broadcast, garantindo que todas as mensagens enviadas por um cliente sejam exibidas a todos os demais clientes conectados ao servidor.
RF-003	O servidor deve gerenciar conexões simultâneas de múltiplos clientes, criando uma thread para cada cliente conectado.
RF-004	O cliente deve disponibilizar uma interface para envio de mensagens e visualização de mensagens recebidas.
RF-005	O sistema deve notificar os clientes e o servidor quando um cliente desconectar, exibindo uma mensagem na tela de todos os usuários conectados
RF-006	O servidor deve manter um registro de mensagens e eventos (entrada/saída de usuários) em um log interno para fins de monitoramento.
RF-007	O sistema deve verificar e informar erros de conexão aos clientes, caso não consigam se conectar ao servidor
RF-008	sistema deve identificar cada cliente por um nome único informado no momento da conexão
RF-009	O cliente deve se conectar ao servidor utilizando o IP e a porta configurados no código.
RF-010	O cliente deve ser capaz de enviar mensagens para o servidor.
RF-011	O servidor deve encaminhar todas as mensagens recebidas de um cliente para todos os outros clientes conectados, incluindo o próprio remetente.
RF-012	O cliente deve ser capaz de receber mensagens enviadas pelo servidor (ou de outros clientes) em tempo real.
RF-013	O cliente deve ser capaz de enviar mensagens privadas para outros clientes conectados.
RF-014	O servidor deve processar e entregar mensagens privadas corretamente, encaminhando para o cliente correto com base no nome fornecido.
RF-015	O servidor deve informar ao cliente se o destinatário da mensagem privada não foi encontrado.
RF-016	O cliente deve poder encerrar sua conexão com o servidor de forma voluntária ao enviar o comando /sair.

RF-017	O servidor deve ser capaz de remover clientes da lista de clientes conectados quando um cliente se desconectar, e notificar os outros clientes de que o cliente saiu.
RF-018	O servidor deve continuar funcionando normalmente e aceitar novas conexões após a desconexão de um cliente.
RF-019	A interface gráfica do cliente deve ser construída usando o Tkinter, contendo: Uma área de exibição de mensagens (usando Listbox). Uma caixa de texto para entrada de mensagens (usando Entry). Um botão "Enviar" para enviar as mensagens digitadas. Verificação: Verificar se os componentes (caixa de texto, área de exibição e botão) estão funcionando corretamente, e se o envio de mensagens funciona como esperado.
RF-020	A interface gráfica deve permitir que o cliente envie mensagens pressionando o botão "Enviar" ou a tecla Enter.
RF-021	A interface gráfica do cliente deve permitir o scroll automático para exibir a última mensagem recebida.
RF-022	O servidor deve ser capaz de lidar com erros de envio de mensagem para os clientes (por exemplo, quando a conexão de um cliente é interrompida) e deve remover o cliente da lista de clientes.
RF-023	O cliente deve lidar adequadamente com a perda de conexão com o servidor, exibindo uma mensagem de erro e encerrando a aplicação de forma segura.
RF-024	O cliente deve tratar o formato inválido de mensagens privadas, respondendo com uma mensagem de erro apropriada.
RF-025	O cliente deve conseguir enviar e receber mensagens de maneira eficiente e sem bloqueio da interface, mesmo em redes com latência moderada.
RF-026	O servidor deve aceitar conexões de múltiplos clientes ao mesmo tempo, criando uma nova thread para cada cliente conectado.