Lucas Reali, Gabriel João, Leonardo Ceccon

Relatório chat:

EXPLICAR COMO FORAM UTILIZADOS OS SOCKETS:

Foi criado um socket para o servidor e os outros sockets são criados conforme os clientes são conectados a esse servidor, onde cada sockets é atribuído a um cliente distinto, onde há um dicionário onde o nome do cliente e seu socket é armazenado; o servidor e o cliente se associam aos sockets AF_INET que correspondem aos endereços ipv4, e o SOCK_STREAM indica que eles se comunicarão por protocolo TCP(o escolhido para o nosso projeto por ser uma conexão mais segura em relação a troca de pacotes), TCP não faz broadcast naturalmente, então fizemos um tratamento de broadcast como indicado no tópico abaixo. O servidor usa o bind() para associar o socket com host e porta, o cliente utiliza o connect() para fazer o mesmo e se conectar ao servidor; no servidor é usado o listen() para fazer o servidor estar aberto, “ouvindo” posteriores conexões.

```
self.clientes = {} # Armazena os clientes    '6666': 'Lucas'


self.servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Cria o secket
self.servidor.bind((self.host, self.port)) # Vincula o secket ao host e a pora fornecidas
self.servidor.listen() # coloca o socket para escutar as conexões
```

EXPLICAR COMO É FEITO O TRATAMENTO DE BROADCAST:

Foi criado uma função no código, que recebe como parâmetros a mensagem que é recebida pelo cliente relacionado, e a própria conexão do cliente que está na função de gerenciamento de clientes; começa usando um “for” que passa por todos os índices/chaves do dicionário e, nesse caso, faz referência a conexão associada ao nome inserido pelo usuário; o “if” valida que o cliente não envie a mensagem para ele mesmo, caso isso seja cumprido o server realiza o broadcast; o “try except” faz um tratamento de erros, caso o fluxo básico não ocorra e haja um erro o cliente será desconectado do servidor e no servidor será “printado” que houve um erro.

```
def enviar_broadcast(self, mensagem, cliente_socket):
    for cliente in self.clientes: # Passa em todos os elementos da lista
        if cliente != cliente_socket:
            try:
                cliente.send(mensagem.encode()) # Envia mensagem
            except Exception as e:
                print(f"Erro ao enviar mensagem: {e}")
                self.remover_cliente(cliente)
```

Lucas Reali, Gabriel João, Leonardo Ceccon

EXPLICAR COMO FORAM UTILIZADAS AS THREADS NO CLIENTE E NO SERVIDOR:

Os threads foram utilizados para que o servidor possa receber as mensagens e enviar essas mensagens recebidas de forma simultânea e paralela para que não haja interrupções pois precisa sempre estar aberto a novas conexão, como por exemplo o servidor precisar ser interrompido para a adição de um novo usuário ou a troca de mensagens simultâneas não gere um gargalo no servidor. Assim, os threads são distribuídos como uma para o servidor e duas para cada usuário, sendo uma para enviar mensagens e outra para recebê-las.

```
def iniciar_servidor(self):
    while True:
        cliente_socket, _ = self.servidor.accept() # Aguarda novas conexoes
        threading.Thread(target=self.gerenciar_cliente, args=(cliente_socket,)).start() # Cria uma thread
```

```
def conectar():
    global cliente_chat
    cliente_chat = cliente.ClienteChat()  # Substitua pelo IP do servidor
    threading.Thread(target=cliente_chat.conectar).start()
    threading.Thread(target=receber_mensagens).start()
```

REQUISITOS:

1. O sistema deve possuir uma interface simples que possua um campo para digitar mensagens, botão de enviar, e uma tela que mostre as mensagens enviadas;

2. O sistema deve solicitar o nome do usuário no momento em que ele entrar;

3. O sistema deve ter a possibilidade de mandar mensagens públicas, onde um cliente mandará a mensagem e todos os outros conectados a receberão;

4. O sistema deve possibilitar a troca de mensagens privadas, que o usuário possa digitar um símbolo específico, junto do destinatário que quer enviar a mensagem e digitar a mensagem, essa mensagem será impressa apenas na tela do usuário digitado;

5. O sistema deve mostrar o nome do usuário que enviou cada mensagem pública e mostrar para o destinatário na mensagem privada;

6. O sistema deve possuir tratamento de erros, que impeçam que o código do servidor quebre e todos caiam, caso haja um possível erro por parte do usuário o sistema deverá remover a conexão respectiva;

7. O sistema deve usar threads para conseguir manter a conexão de/com múltiplos usuários simultâneos;

8. O sistema deverá permitir que o usuário/cliente possa sair do chat usando o comando “#sair”, após isso o servidor deverá cortar a conexão com respectivo cliente;

9. O sistema deverá imprimir para todos os usuários conectados o nome do usuário juntamente com um aviso de que ele entrou. No servidor deverá ser impresso a conexão respectiva do usuário (seu nome);

10. O sistema deve possibilitar a comunicação de múltiplos usuários conectados à mesma rede.