


<div></div> <div>Universidade Federal do Ceará – Campus de Russas</div>			AV1	AV2	PF
		Trabalho		X	
		Prova			
Curso: C.C. e E. S.	Disciplina: Sistemas Distribuídos	Data Sigaa: 12 a 26/08/2024			
Professor: Cenez Araújo de Rezende					
Alunos(as): FELIPE GOMES DA SILVA - 498842 FELIPE CESAR DE SOUSA SILVA - 500611 ISAC MAXIMO ALVES DE MOURA - 499430 MATHEUS SANTOS SOARES - 497267					

Introdução

O projeto envolve a criação de um sistema de comunicação entre nós utilizando Python, onde um cliente e um servidor se comunicam através de sockets. O sistema é projetado para enviar e receber mensagens, e implementar um protocolo específico para consultar o detentor de uma chave K. A implementação inclui três principais arquivos Python: main.py, data_com.py, cliente.py, e servidor.py. No arquivo mostraremos nossas dificuldades, aprendizados e lições que ficaram no período de aprendizagem do código.

Obs: O código foi dividido em branches no github, então, para melhor visualização pode se trocar de branch para cada desafio, estaremos disponibilizando o link do github para visualização.

link github: <https://github.com/Felipe-Gs/Chord-peer-to-peer>

Descrição do Código

- main.py: É o ponto de entrada do sistema. Ele configura e inicia as threads para o servidor e o cliente.
- data_com.py: Define a classe DataCom, que configura as portas e os detalhes de conexão para o servidor e o cliente.
- cliente.py: Define a classe Cliente, responsável por enviar e receber mensagens do servidor.
- servidor.py: Define a classe Servidor e o manipulador de solicitações ComunicadorTCPHandler, que processa as mensagens recebidas e responde aos clientes.

Dificuldades Enfrentadas

- Gerenciamento de Erros e Logs

Foi bastante complicado entender e analisar o código, ainda mais por se tratar de um código legado, apesar de ter o vídeo explicando quando vai pra prática é um pouco diferente e sentimos dificuldade em compreender, na maioria das vezes fomos adicionando “prints” ao longo do código para saber o que tal variável taça guardando, o que tal função vazia, esse tipo de coisa.

- Complexidade do Sistema

O desenvolvimento de um sistema com múltiplas partes interconectadas (servidor, cliente e comunicação de dados) pode aumentar a complexidade do projeto, tornando o design, a implementação e a manutenção mais desafiadores.

- Comunicação da Equipe

Coordenar o trabalho entre os membros da equipe, garantindo que todos estejam alinhados com os objetivos do projeto e que as tarefas sejam realizadas de forma eficiente, foi um desafio, pois tínhamos conhecimentos diferentes em programação, tentamos fazer com que todos aprendessem um pouco do que estávamos tentando fazer e implementar.

- Comunicação entre Threads

Garantir que a comunicação entre o servidor e o cliente seja eficiente e livre de problemas de sincronização foi difícil . Isso inclui a transmissão de dados sem perdas e a correta interpretação dos mesmos.

Aprendizado

- **Uso de sockets em python**

Usar sockets em Python envolve a criação e gerenciamento de conexões de rede usando a biblioteca `socket`. Isso possibilitou a configuração de sockets TCP ou UDP, estabelecendo conexões entre cliente e servidor, enviando e recebendo dados de forma eficiente com métodos como `send()` e `recv()`, e garantindo o fechamento correto das conexões para liberar recursos. Também é crucial lidar com erros de conexão e implementar práticas seguras para evitar falhas, como timeouts e interrupções inesperadas. O conhecimento de como os sockets operam, incluindo as diferenças entre TCP e UDP, é essencial para construir aplicações de rede robustas.

- **Gerenciamento de Threads**

O gerenciamento de threads em Python nos permite executar várias tarefas ao mesmo tempo, usando a biblioteca `threading`. Isso é super útil quando precisamos rodar várias operações em paralelo, como o cliente e o servidor no nosso projeto. Garantimos que recursos compartilhados sejam acessados de forma segura, utilizando bloqueios e semáforos para evitar conflitos.

Além disso, controlamos a ordem em que as threads são executadas com métodos como `join()`, e usamos filas (`queue.Queue`) para permitir uma comunicação segura entre elas. Focar na segurança das threads e evitar problemas de concorrência ajuda a manter nossas aplicações mais estáveis e eficientes.

- **Tratamento de Exceções**

O tratamento de exceções em Python é fundamental para garantir que nosso programa continue funcionando, mesmo quando algo dá errado. Usamos blocos `try-except` para capturar erros específicos, como `ConnectionError`, e gerenciá-los de forma controlada. Isso nos ajuda a lidar com problemas de maneira organizada. O bloco `finally` também é importante, pois garante que recursos, como conexões, sejam liberados corretamente, não importando se ocorreu uma exceção ou não. Criar exceções personalizadas torna o código mais claro e facilita a identificação de problemas específicos. Além disso, o uso de `logging` para registrar exceções é uma prática essencial.

● Formatos de Mensagem e Protocolos

Formatos de mensagem e protocolos em comunicação de rede são essenciais para garantir que os dados trocados entre cliente e servidor sigam regras claras e consistentes. Usar formatos como JSON ou XML nos ajuda a organizar as informações de maneira padronizada e fácil de manipular em Python. Criar protocolos simples, como comandos de texto, facilita a compreensão entre as partes envolvidas. Além disso, validar as mensagens e implementar mecanismos para lidar com erros ou dados incorretos é fundamental para tornar o sistema mais confiável e resistente a falhas. Isso nos ajuda a evitar problemas e garantir que a comunicação entre cliente e servidor seja robusta e eficiente.

Conclusão

O projeto foi uma excelente oportunidade para aplicar conceitos de comunicação de rede e programação concorrente, fornecendo uma visão prática de como os sistemas distribuídos operam. Ao implementar um protocolo de comunicação entre nós utilizando sockets, conseguimos explorar aspectos fundamentais da interação em rede, como a criação e gerenciamento de conexões, o envio e recebimento de mensagens, e a sincronização de tarefas.

Conceitos de Comunicação de Rede: Trabalhar com sockets e protocolos de comunicação permitiu uma compreensão mais profunda dos conceitos subjacentes à comunicação entre máquinas em uma rede. Aprendemos a definir e implementar protocolos personalizados, lidar com diferentes formatos de mensagens e gerenciar conexões de forma robusta e eficiente.

Programação Concorrente: A implementação de threads para o servidor e o cliente destacou a importância da programação concorrente em sistemas distribuídos. O uso de threads para executar tarefas simultaneamente é crucial para garantir que o sistema permaneça responsivo e eficiente, especialmente em aplicações que exigem comunicação em tempo real.

Soluções de Problemas e Depuração: As dificuldades encontradas, como problemas de conexão e formatação de mensagens, foram desafiadoras, mas proporcionaram valiosas oportunidades de aprendizado. A depuração e a resolução desses problemas ajudaram a desenvolver habilidades críticas em encontrar e corrigir bugs, além de aprimorar a capacidade de testar e validar o comportamento do sistema.

Aplicação Prática: O sistema desenvolvido demonstra como os sockets podem ser utilizados para criar uma aplicação de rede interativa e resiliente. A implementação do protocolo de consulta para o detentor de uma chave exemplifica como os nós em uma rede distribuída podem interagir e trocar informações de forma eficiente. Isso é particularmente

relevante em sistemas de bancos de dados distribuídos, sistemas de arquivos distribuídos e outras aplicações que dependem de comunicação entre múltiplos componentes.

Impacto e Aprendizado: O projeto não apenas consolidou o conhecimento sobre a comunicação de rede e programação concorrente, mas também proporcionou uma compreensão prática de como construir sistemas distribuídos que podem escalar e se adaptar a diferentes cenários. A experiência adquirida é aplicável a uma ampla gama de projetos e tecnologias, preparando-nos para enfrentar desafios semelhantes em futuros desenvolvimentos de sistemas