UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

SISTEMAS DISTRIBUÍDOS

DOCENTE: RENAN CERQUEIRA AFONSO ALVES

RELATÓRIO: EXERCÍCIO PROGRAMA

Parte 1

Júlia Du Bois Araújo Silva N° USP: 14584360

SÃO PAULO 2024

SUMÁRIO

1. Execução do programa

Primeiramente, é necessária uma instalação da linguagem de programação Python na máquina utilizada. As bibliotecas utilizadas no programa são os, sys, socket e threading. Para executar um peer, utilize o seguinte comando:

python localDoRepositorio/eachare.py <endereco>:<porta> <vizinhos.txt> <diretorio_compartilhado>

Se na sua máquina há outra versão do Python instalado - como, por exemplo, Python3 - , modifique o comando anterior para, em vez de “python”, a palavra-chave da sua versão.

2. Paradigma de programação

1.1 Linguagem

A linguagem de programação escolhida para a realização do EP foi Python, principalmente por conta da facilidade de realizar conexões TCP e de gerenciar threads utilizando as bibliotecas socket e threading.

1.2 Orientação a objetos

Foi escolhido o paradigma de programação de orientação a objetos por sua abordagem modularizada que permite a separação fácil e escalável de diferentes partes do projeto. As classes criadas para execução do EP foram: eachare (classe principal), peer, commandHandler e inputArgumentsChecker.

1.2.1 eachare

Essa é a classe principal do projeto. Ela inclui o gerenciamento das threads e sockets utilizadas, além da inicialização do programa. Também contém métodos auxiliares como o gerenciamento do relógio local e de mensagens.

1.2.2 peer

Essa classe representa um nó da rede, mantendo seu IP, porta utilizada, status e uma lista dos vizinhos conhecidos.

1.2.3 commandHandler

Essa classe é responsável por interpretar e executar comandos locais e remotos. Ela manipula a troca de mensagens, atualizando os peers quando necessário.

1.2.4 inputArgumentsChecker

Essa classe é utilizada como auxiliar para a inicialização do programa. Ela checa os argumentos de entrada, garantindo que sejam válidos, para que a aplicação possa funcionar sem problemas posteriores.

1.3 Threads

A execução principal do programa ocorre dentro da classe eachare, cujo método startProgram() é responsável por inicializar a aplicação. À partir dela, são criadas duas threads principais: a thread de recepção de conexões e a thread de comando do usuário.

1.3.1 Thread de recepção de conexões

Essa thread é responsável por manter o socket escolhido pelo usuário escutando infinitamente. A cada nova conexão aceita, é criada uma nova thread dedicada para processar a mensagem recebida, fazendo com que seja possível receber e reagir a múltiplas mensagens simultaneamente (que será importante no recebimento e envio de mensagens que exijam um tempo de processamento maior). Após o processamento, a nova thread e o socket dedicado a ela são fechados.

1.3.2 Thread de comando do usuário

Essa thread é responsável por receber os comandos enviados pelo usuário via terminal, operando como interface interativa.

2. Testes

Foram realizados testes locais manuais, utilizando apenas uma máquina. Além disso, para simular o uso de múltiplas imagens, foram utilizados containers do Docker, simulando diferentes tempos de lag nos peers (tanto naqueles enviando as mensagens

quanto naqueles recebendo). Não foram encontrados problemas inesperados durante a execução desses testes.

3. Dificuldades, desafios e melhorias para a expansão futura do código

Essa seção do relatório está escrita em primeira pessoa porque muitos dos desafios encontrados vieram de falta de preparação pessoal antes do início da elaboração do EP.

Encontrei certa dificuldade na organização do programa orientada a objetos, pois não é um paradigma com dentro do qual estou muito acostumada a pensar. Com o tempo, porém, me familiarizei com as vantagens de utilizar esse paradigma (especialmente as vantagens para a escalabilidade futura do código), e creio que consegui superar, até certo ponto, o desafio que o novo paradigma apresentou. Para futuras entregas, gostaria de separar ainda classes para o parser de mensagens (separando-o da classe commandHandler) e a lógica de envio e recebimento de mensagens (separando-a da classe eachare), por exemplo, e acredito que outras classes poderão surgir no futuro para melhor modularização do código.

O próprio uso de Python (apesar de facilitar o uso de sockets e threads em comparação com linguagens como C e Java) foi um desafio, pois não nunca havia utilizado essa linguagem em um projeto dessa dimensão, que exigisse múltiplas classes. Por conta da flexibilidade da linguagem em misturar orientação a objetos e programação procedural, me perdi em alguns momentos quanto à organização do código. Isso prejudicou a realização do EP pela perda de tempo que poderia ter sido evitada caso ele fosse elaborado com mais firmeza quanto aos padrões de desenvolvimento e organização do sistema. Ainda não estou feliz com a organização atual e espero que, para as próximas entregas desse EP, consiga melhorar o desenvolvimento quanto a esse aspecto.

O uso de múltiplas threads é um desafio que não creio ter superado completamente ainda. O gerenciamento das threads dentro do programa ainda não é o ideal e permite que seja gerado um excesso de threads concorrentes que pode consumir a memória de uma máquina rapidamente, e as exceções e erros gerados por elas não estão adequadamente tratados dentro do sistema.

No que diz respeito à comunicação entre peers, uma melhoria significativa seria a implementação de um protocolo de resposta estruturado. Atualmente, muitos comandos — como o GET_PEERS — são enviados sem que haja uma resposta formal

e rastreável por parte do receptor, o que compromete a comunicação. Isso também poderia ter sido evitado com uma organização mais rígida do programa criada anteriormente ao início da programação.