UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SISTEMA DE COMPARTILHAMENTO DE ARQUIVOS PEER-TO-PEER

Exercício Programa: parte 1

FILIPE BISON DE SOUZA - 14577653 - Turma 04
LUAN PEREIRA PINHEIRO - 13672471 - Turma 04

SÃO PAULO
2025

SUMÁRIO

1. PROPOSTA

1.1. Objetivo

Este projeto tem como objetivo implementar um sistema de compartilhamento de arquivos peer-to-peer simplificado chamado EACHare.

1.2. Escolha da linguagem de programação

Para a implementação do EACHare, optamos por utilizar a linguagem Python. A escolha se deu principalmente pela sua simplicidade e legibilidade, facilitando tanto o desenvolvimento quanto a manutenção do código. Além disso, Python possui uma biblioteca nativa para manipulação de sockets, permitindo a comunicação entre peers de maneira eficiente e simplificada, a qual já tivemos contato anteriormente.

1.3. Escolha do paradigma de programação

Embora o Python não seja sempre usado com orientação a objetos, escolhemos seguir com esse paradigma. Essa abordagem permite uma organização mais modular e reutilizável do código, facilitando a manutenção e a expansão do sistema à medida que novas funcionalidades são adicionadas, especialmente em projetos mais complexos, como o EACHare.

2. IMPLEMENTAÇÃO

A implementação foi feita parte em classes e parte em funções definidas no arquivo principal do projeto.

2.1. Estrutura de classes

Buscando manter um desacoplamento que funcionasse bem para o projeto, seu desenvolvimento e manutenção definimos três classes:

- PeerManager

A PeerManager é responsável por armazenar as listas de peers, fazer validações, adicionar novos peers. Essa classe é instanciada globalmente na main.py.

- Peer

A Peer busca representar e armazenar os dados de todos os peers que o servidor teve contato, responsável por alterar seu estado e descrevê-lo como uma mensagem para quando necessário pelo list_peers.

- Connection:

Já a classe Connection concentra toda a lógica de receber e abrir conexões TCPs, fazendo o controle de threads para elas e tratando as mensagens recebidas.

Inicialmente, foi difícil manter o desacoplamento visto que logicamente as ações de uma classe dependem ou são iniciadas por responsabilidades de outras classes, mas adicionando uma instância do PeerManager no Connection foi possível realizar essa comunicação sem acoplar o projeto.

2.2. Separação em threads

Na thread principal do programa, realizamos as operações principais, como exibição do menu e tratamento das entradas do usuário.

Utilizamos uma thread daemon para aceitar conexões em segundo plano. Threads desse tipo são finalizadas junto com o programa, o que facilita o desenvolvimento do nosso programa, visto que não há operações dessa parte do EP que precisam continuar executando após a finalização da thread principal.

As únicas funções dessa thread são aceitar conexões e iniciar outras threads daemon para tratar essas conexões, por isso ela possui operação bloqueante, que bloqueia a execução da thread até receber uma conexão nova.

As threads que lidam com as conexões possuem operações que bloqueiam o acesso a recursos de uso compartilhado entre as threads, como o clock e a lista de peers, evitando condições de corrida. Esta foi uma parte difícil de implementar, devido à facilidade de entrar em deadlocks.

Essas threads realizam a comunicação com o cliente, tratando a mensagem e fazendo as atualizações necessárias nas variáveis do programa, bem como respondendo a mensagem, quando for o caso. Por simplicidade de código, todas as mensagens são enviadas em uma nova conexão.

2.3. Estruturas de dados

A principal estrutura de dados que foi necessária para o projeto foi o dicionário dos peers (dict na linguagem Python), que mapeia um par (IP, porta) para um objeto do tipo Peer. Usamos essa estrutura por ela ter acesso e inserção rápidos - em O(1) -, por ser implementada como uma tabela hash.

Além dessas operações, fazemos cópia dos valores do dicionário para uma lista quando precisamos listar os peers, como na operação PEER_LIST. Essa cópia ocorre em O(n), ótima complexidade para esse tipo de operação.

2.4. Decisões de projeto

Algumas decisões foram tomadas por não serem especificadas no projeto entre elas:

- Ao receber qualquer mensagem de qualquer peer, alteramos o estado dele para online.
- Na especificação não é definido se o menu de lista de peers deveria ficar em loop, mas deixamos assim para facilitar a usabilidade e por ser mais intuitivo

3. TESTES REALIZADOS

Realizamos testes unitários de cada método e função do programa ao longo do desenvolvimento do projeto. Realizamos testes fim-a-fim entre diversos peers, testando todas as funcionalidades do programa a partir de todos os diversos peers e com fim em todos os diversos peers. Incluímos testes com diretórios não existentes, arquivos não existentes e peers offline.