UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS e HUMANIDADES
GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Prof. Dr. Renan Cerqueira Afonso Alves
Aaron Ferraz do Amaral Martins da Silva - 14554474
Renan Gonçalves Mastropaolo - 10892593

Relatório de Desenvolvimento de Sistemas de Informação Distribuídos - EP1

São Paulo
2025

Relatório do Sistema de Compartilhamento de Arquivos (EACHare)

O sistema de compartilhamento de arquivos em uma rede P2P, denominada EACHare, foi realizado para a primeira parte do EP da matéria de Desenvolvimento de Sistemas de Informação Distribuídos, no qual quis evidenciar a implementação de uma arquitetura peer to peer estruturada, que contém uma estrutura global com pares chave:valor que organiza os nós e recursos, além de facilitar uma busca eficiente. Nessa primeira parte, o interesse é praticar o comportamento inicial dos nós da rede em se conhecerem, ou seja, manipular o status de cada nó através de trocas de mensagens em um socket TCP.

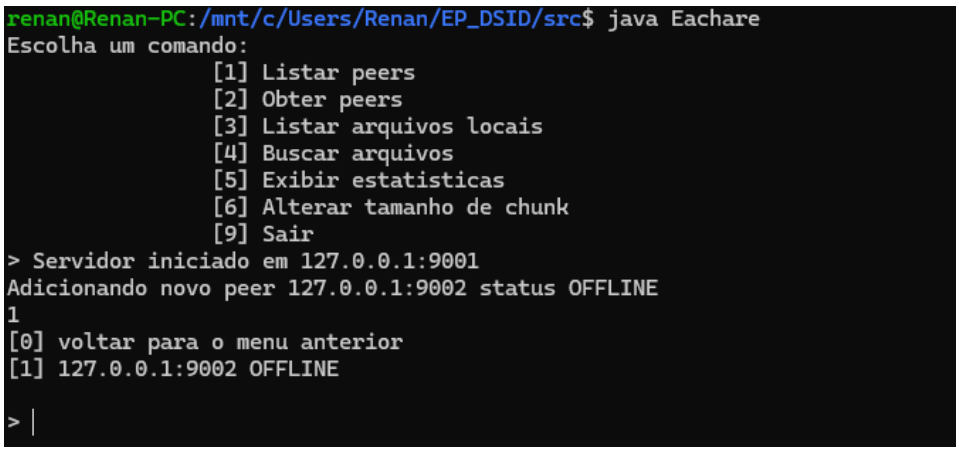
Como Compilar e Executar o Código

A compilação pode ser feita pela classe main “EACHare” que invoca as outras classes que são utilizadas no programa, então temos que compilar ela em bytecode (.class) e após isso executarmos. Tendo em vista a JVM do java que utiliza o conceito de “Write once, run anywhere” que contribui com a portabilidade da linguagem, podemos rodar aplicações java em qualquer ambiente, sendo da mesma forma no Windows ou na distribuição Ubuntu 22.04 do Linux. Segue o passo a passo:

diretorio/do/projeto javac Eachare.java

diretório/do/projeto java Eachare <ip:porta> <path/para/arquivos/vizinhos.txt>
<path/dir/compartilhado>

E.g.: java Eachare 127.0.0.1:9001 files/peer1.txt files/p1/



Obs: Este projeto foi executado e testado na versão 21 do jdk.
Obs2: Caso o programa seja executado sem nenhum parâmetro, os três valores assumidos serão os definidos por padrão:

- 127.0.0.1:9001
- files/peer1.txt
- files/p1/

A Escolha do Paradigma e Organização

A seleção da linguagem de programação influenciou diretamente em nossa forma de modelar o problema, neste caso foi o Java, com forte ligação com o paradigma orientado a objetos. Em virtude da experiência e familiaridade com a linguagem através de utilização em projetos pessoais e matérias passadas, além disso um dos pontos que mais nos atraiu foi a simplicidade trazida na construção de programação paralela e sockets, conceitos basilares na implementação do programa.

O código está estruturado em três pacotes, cada um com um propósito diferente e classes com funções relacionadas. O pacote “logger” apresenta um jeito de formatar e exibir logs mais padronizados, sendo utilizado ao longo do código para auxiliar na visualização da execução do programa.

O pacote “Peer” representa os nós propriamente ditos, sendo uma thread cliente que implementa a interface Runnable que executa outra thread que atua como um servidor, afinal, o mesmo nó tem que atuar das duas maneiras, evidenciando uma implementação multi thread que é necessária no contexto de arquiteturas peers to peers, pelo comportamento supracitado. Cada peer tem um ip, porta, caminho relativo para o arquivo txt de vizinhos, clock inicializado com zero, um objeto Path para podermos acessar o diretório compartilhado. Além disso, existem os métodos que combinados realizam as funcionalidades pedidas no enunciado. A classe “PeerHandler” foi feita para gerenciar a estrutura compartilhada, um HashMap pertencente a classe, criado com uma chave sendo o par ip:porta e como valor um objeto PeerInfo com informações e status do peer, que pode eventualmente trazer uma condição de disputa entre os nós. Optamos por fazer operações bloqueantes para que as threads esperem a sua vez, priorizamos em utilizar ao adicionar um vizinho à estrutura, mudar o status de um peer na lista de vizinhos e ao mandar mensagens BYE para desativar o peer. Além dessas operações bloqueantes, existe a de incrementar o clock e exibir a mensagem de que o mesmo foi alterado, percebemos que não era estritamente necessário, mas fizemos para garantir.

A classe “PeerInfo” é usada para armazenar apenas os valores essenciais de um nó, como seu ip, porta e status, tendo getters para eles e um setter para poder alterar o status do peer em determinadas operações.

O pacote “Protocol” é responsável por administrar a comunicação de mensagens trocadas entre as máquinas. Na classe “MessageHelper” temos a função de “createMessage()” focada em colocar a mensagem na sintaxe correta estipulada pelo enunciado e garantir que as mensagens trocadas pelo programa sigam o mesmo padrão. Na classe “MessageHandler” consiste em funções para gerenciar o está sendo transmitido ou recebido através do socket, na função “handleSendMessage” abre um canal de comunicação com o vizinho, envia uma mensagem em binário e depois recebe uma resposta convertida em caracteres e armazenada em um buffer, caso dê certo é setado como ONLINE o vizinho, retornando a primeira linha desse buffer, no caso sendo a resposta. A função “handleReceiveMessage” fragmenta a mensagem para poder abstrair cada campo, ao identificar o tipo da mensagem chamará determinados métodos da classe Peer. Por último, temos o “handleAnswerMessage” que envia uma mensagem de resposta pelo socket.

A classe com o método main se chama “Eachare”, nela criamos o peer padrão que pode receber argumentos do método main ou se não for passado nada, vai ser inicializado com valores fixos e também será criada uma thread do tipo peer para esse novo peer. Temos o menu textual e as funcionalidades estabelecidas pelo enunciado.

Implementação das funcionalidades

Corroborando com o menu textual, na escolha do primeiro valor temos a listagem dos vizinhos e também mandamos uma mensagem do tipo “HELLO” a um dos vizinhos especificado.

```
case 1:
    PeerInfo listarPeersResult = peer.listarPeers();
    if (listarPeersResult!=null) {
        try {
            peer.sendMessage(listarPeersResult, message: "HELLO");
        } catch (RuntimeException re){
            log.log( message: "Não deu certo mandar a mensagem", breakLine: true);
        }
    }
    break;
```

listarPeers() - lê o HashMap e mostra na tela no formato de opções das quais o usuário pode escolher, retorna um objeto PeerInfo quando um nó é selecionado.
sendMessage() - método que invoca o tratamento de uma mensagem que defini como ONLINE o peer que mandou na lista de vizinhos do peer que recebe.

```
case "HELLO":
    peer.addNeighborByAddress(source);
    handleAnswerMessage(clientSocket, answer: "");
    break;
```

handleAnswerMessage - envia uma mensagem de resposta ao cliente, ao ser enviado uma string vazia porque nesse tipo de mensagem, nenhuma resposta é esperada.
addNeighborByAddress - procura um nó com o endereço:porta no HashMap. Caso encontre, adicione nesta coleção com status ONLINE. No entanto, se já estiver no map e for OFFLINE, mude o status. Portanto, essa é a função que implementa o que o HELLO deve provocar, isto é, uma mudança dos status dos peers.

Peer principal (127.0.0.1:9001):

```
> 1
[0] voltar para o menu anterior
[1] 127.0.0.1:9003 OFFLINE
[2] 127.0.0.1:9002 OFFLINE

> 2
[02/04/2025 12:11:08] [main] [Peer] => Atualizando relogio para 1
[02/04/2025 12:11:08] [main] [Peer] Encaminhando mensagem "127.0.0.1:9001 1 HELLO" para 127.0.0.1:9002
[02/04/2025 12:11:08] [main] [PeerInfo] Atualizando peer 127.0.0.1:9002 status ONLINE
Escolha um comando:
        [1] Listar peers
        [2] Obter peers
        [3] Listar arquivos locais
        [4] Buscar arquivos
        [5] Exibir estatisticas
        [6] Alterar tamanho de chunk
        [9] Sair
> 1
[0] voltar para o menu anterior
[1] 127.0.0.1:9003 OFFLINE
[2] 127.0.0.1:9002 ONLINE
```

Vizinho:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA C
[02/04/2025 12:10:51] [main] [PeerInfo] Adicionando novo peer 127.0.0.1:9003 status OFFLINE
[02/04/2025 12:10:51] [main] [PeerInfo] Adicionando novo peer 127.0.0.1:9001 status OFFLINE
[02/04/2025 12:10:51] [Thread-0] [Peer] Servidor iniciado em 127.0.0.1:9002
Escolha um comando:
        [1] Listar peers
        [2] Obter peers
        [3] Listar arquivos locais
        [4] Buscar arquivos
        [5] Exibir estatisticas
        [6] Alterar tamanho de chunk
        [9] Sair
> [02/04/2025 12:11:08] [Thread-1] [MessageHandler] Mensagem recebida: 127.0.0.1:9001 1 HELLO
[02/04/2025 12:11:08] [Thread-1] [Peer] => Atualizando relogio para 1
[02/04/2025 12:11:08] [Thread-1] [PeerInfo] Atualizando peer 127.0.0.1:9001 status ONLINE
1
[0] voltar para o menu anterior
[1] 127.0.0.1:9003 OFFLINE
[2] 127.0.0.1:9001 ONLINE

>
```

Funcionalidade 2

```
        case 2:
            peer.getPeers();
            break;
```

getPeers - manda uma mensagem GET_PEERS para todos os nós e o adiciona no HashMap.

```
case "GET_PEERS":
    peer.addNeighborByAddress(source);
    peer.listarPeersConhecidos(clientSocket, source);
    break;
```

listaPeersConhecidos - preenche um ArrayList com objetos do tipo PeerInfo, desconsiderando o próprio endereço. Concatena todos os atributos em um StringBuilder, cria uma mensagem ip, porta, clock, “PEER_LIST”, número de vizinhos e os vizinhos e manda como uma mensagem de resposta pelo handleAnswerMessage.

```
> 2
[02/04/2025 13:09:56] [main] [Peer] => Atualizando relogio para 1
[02/04/2025 13:09:56] [main] [Peer] Encaminhando mensagem "127.0.0.1:9001 1 GET_PEERS" para 127.0.0.1:9003
[02/04/2025 13:09:56] [main] [Peer] Resposta recebida: "127.0.0.1:9003 1 PEER_LIST 0"
[02/04/2025 13:09:56] [main] [Peer] => Atualizando relogio para 2
[02/04/2025 13:09:56] [main] [PeerInfo] Atualizando peer 127.0.0.1:9003 status ONLINE
[02/04/2025 13:09:56] [main] [Peer] => Atualizando relogio para 3
[02/04/2025 13:09:56] [main] [Peer] Encaminhando mensagem "127.0.0.1:9001 3 GET_PEERS" para 127.0.0.1:9002
[02/04/2025 13:09:56] [main] [Peer] Resposta recebida: "127.0.0.1:9002 1 PEER_LIST 1 127.0.0.1:9003:OFFLINE:0"
[02/04/2025 13:09:56] [main] [Peer] => Atualizando relogio para 4
[02/04/2025 13:09:56] [main] [PeerInfo] Atualizando peer 127.0.0.1:9002 status ONLINE
Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       [3] Listar arquivos locais
       [4] Buscar arquivos
       [5] Exibir estatisticas
       [6] Alterar tamanho de chunk
       [9] Sair
> 1
[0] voltar para o menu anterior
[1] 127.0.0.1:9003 ONLINE
[2] 127.0.0.1:9002 ONLINE
```

Funcionalidade 3

```
    case 3:
        peer.showDirectoryShared();
        break;
```

showDirectoryShared - usa uma sequência de elementos do tipo Path e cada elemento é um caminho dos arquivos/diretórios. Com o Files.list() vai listar todos os arquivos de cada elemento. Por meio de um foreach, pegamos apenas o nome de cada arquivo do diretório e imprimimos.
exemplo:

```
Escolha um comando:
        [1] Listar peers
        [2] Obter peers
        [3] Listar arquivos locais
        [4] Buscar arquivos
        [5] Exibir estatisticas
        [6] Alterar tamanho de chunk
        [9] Sair
> 3
icons8-react-48.png
Saitama.png
```

Funcionalidade 4

```
    case 9:
        peer.bye();
        System.exit( status: 0);
        break;
```

bye - vai usar um foreach para ler todos os elementos do HashMap e para cada elemento que estiver com status ONLINE, vai mandar uma mensagem BYE pelo “sendMessage” mostrando que aquele peer vai parar de funcionar.
System.exit(0) - encerra imediatamente a execução da JVM e das threads rodando em segundo plano, para referido peer.

```
case "BYE":
    peer.changeStatusPeer(source);
    handleAnswerMessage(clientSocket, answer: "");
    break;
```

changeStatusPeer - caso o endereço passado esteja no HashMap, será pego o valor associado e por ser um objeto PeerInfo, o status pode ser alterado para OFFLINE.

Saindo do peer:

```
Escolha um comando:
        [1] Listar peers
        [2] Obter peers
        [3] Listar arquivos locais
        [4] Buscar arquivos
        [5] Exibir estatisticas
        [6] Alterar tamanho de chunk
        [9] Sair
> 9
Saindo...
[02/04/2025 12:36:03] [main] [Peer] => Atualizando relogio para 2
[02/04/2025 12:36:03] [main] [Peer] Encaminhando mensagem "127.0.0.1:9001 2 BYE" para 127.0.0.1:9002
```

Vizinho do peer com status ONLINE:

```
Escolha um comando:
        [1] Listar peers
        [2] Obter peers
        [3] Listar arquivos locais
        [4] Buscar arquivos
        [5] Exibir estatisticas
        [6] Alterar tamanho de chunk
        [9] Sair
> [02/04/2025 12:36:03] [Thread-2] [MessageHandler] Mensagem recebida: 127.0.0.1:9001 2 BYE
[02/04/2025 12:36:03] [Thread-2] [Peer] => Atualizando relogio para 2
[02/04/2025 12:36:03] [Thread-2] [PeerInfo] Atualizando peer 127.0.0.1:9001 status OFFLINE
1
[0] voltar para o menu anterior
[1] 127.0.0.1:9003 OFFLINE
[2] 127.0.0.1:9001 OFFLINE
```

Dificuldades enfrentadas

Apesar de não ser um exercício-programa trivial, acreditamos que o desenvolvimento fluiu, em alguns pontos como clock, tratar mensagens, o que deveria ser bloqueante, definir o comportamento de algumas funções, tivemos certas dúvidas, entretanto conseguimos solucionar e trazer um sistema funcional.