UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

ACH2147 – Desenvolvimento de Sistemas de Informação Distribuídos
1º semestre de 2025

Exercício Programa: parte 2

Prof. Dr. Renan Cerqueira Afonso Alves

Sumário

1 Introdução

Nesta segunda parte do exercício, vamos implementar um protocolo simples baseado no relógio de Lamport para prover consistência à informação de status dos peers. Além disso, vamos começar a implementar a funcionalidade de busca e download de arquivos, ou seja, o comando 4 do menu do programa.

```
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
>
```

Figura 1: Menu principal

A inicialização do programa e o formato de mensagem permanecem da mesma forma especificada na parte 1 do exercício. Releia trechos do enunciado da parte 1, caso ache necessário.

2 Alterações do funcionamento do relógio local

O relógio dos peers deve funcionar como um relógio de Lamport, que pode ser descrito da seguinte forma:

- O valor inicial do relógio é 0;
- Antes de enviar qualquer mensagem, o valor do relógio deve ser incrementado em 1 (portanto a mensagem enviada devera possuir o valor já incrementado no cabeçalho);
- **Este item é diferente em relação à parte 1 do EP:**
  Ao receber uma mensagem, deve-se primeiro alterar o valor do relógio local para o maior valor entre o valor atual e o valor contido no cabeçalho da mensagem recebida (ou seja, algo como `clock_local = max(clock_local, clock_mensagem)`. Em seguida, o valor do relógio deve ser incrementado em 1;
- Sempre que o valor do relógio for atualizado, uma mensagem deverá ser exibida na saída padrão com o seguinte formato: `"=> Atualizando relogio para <valor>"`.

3 Alterações no gerenciamento de peers conhecidos

Vamos usar o valor do relógio para implementar um protocolo simples[1] de consistência para gerenciar a informação de status dos peers conhecidos. Em essência, o protocolo funciona da seguinte forma:

- O status de um peer sempre é atualizado ao receber uma mensagem direta daquele peer;
- O status de um peer é atualizado através das mensagens `GET_PEERS` e `PEER_LIST` apenas se for uma informação mais recente do que a armazenada localmente.

Para implementar este protocolo, cada peer deve armazenar, além do status, uma informação adicional para cada um dos peers conhecidos. Esta informação adicional é o valor de relógio naquele peer, sendo o valor mais recente que se conhece. Este valor deve ser atualizado da seguinte forma:

- Ao receber uma mensagem qualquer, o valor de relógio associado ao remetente deve ser atualizado para o valor contido no cabeçalho da mensagem, apenas se este valor for maior[2];
- Lembre-se que se a mensagem for do tipo `BYE` o status deve ser atualizado para OFFLINE, enquanto que para qualquer outra mensagem o status deve ser atualizado para ONLINE.

Por exemplo, suponha que o peer 192.168.0.5:9000 possui a seguinte tabela de peers conhecidos:

Peer	Status	Relógio
192.168.0.6:9000	ONLINE	10
192.168.0.7:9000	ONLINE	15

[1] simples e imperfeito, dependendo das condições de funcionamento

[2] o valor do cabeçalho poderia ser menor no caso do peer ter reiniciado.

Se o peer 192.168.0.5:9000 receber a mensagem `"192.168.0.6:9000 11 HELLO"`, a tabela deve ser atualizada para refletir o novo valor de relógio, da seguinte forma:

Peer	Status	Relógio
192.168.0.6:9000	ONLINE	**11**
192.168.0.7:9000	ONLINE	15

Se, em seguida, a mensagem `"192.168.0.7:9000 13 BYE"` for recebida, o valor do relógio não deve ser atualizado (já que é menor que o valor atual da tabela), mas o status deve ser alterado, já que se trata de uma mensagem do tipo `BYE`. A tabela ficaria da seguinte forma após processar as duas mensagens:

Peer	Status	Relógio
192.168.0.6:9000	ONLINE	11
192.168.0.7:9000	**OFFLINE**	15

O valor de relógio associado a cada um dos peers conhecidos será utilizado para decidir se o status deve ou não ser atualizado ao processar o comando `Obter peers`.

Lembre-se que, ao receber uma mensagem do tipo `GET_PEERS`, deve-se responder com uma mensagem de resposta do tipo `PEERS_LIST`, seguindo o formato padrão de mensagem, contendo os seguintes argumentos:

- A quantidade total de vizinhos descritos na mensagem;
- Cada um dos peers conhecidos, exceto o remetente;
- **Este item é diferente em relação à parte 1 do EP:**
  Cada peer deve ser codificado da seguinte forma: `<endereço>:<porta>:<status>:<relógio>`, onde status pode ser as strings ONLINE ou OFFLINE, e o relógio é o valor de relógio existente na tabela de peers conhecidos.

Ao receber a mensagem do tipo `PEERS_LIST`, o peer que inicialmente enviou a mensagem do tipo `GET_PEERS` deve atualizar o seu conjunto de peers conhecidos, adicionando os peers novos e atualizando o status e o relógio associado aos peers que já eram conhecidos **apenas se o valor de relógio indicado na mensagem for maior que o valor local**.

Vejamos um exemplo. Considere que os peers 10.0.0.5, 10.0.0.6 e 10.0.0.6 possuam as seguintes tabelas de peers conhecidos (omitindo as portas para ficar mais compacto):

Tabela de 10.0.0.5			Tabela de 10.0.0.6			Tabela de 10.0.0.7		
Peer	Status	Relógio	Peer	Status	Relógio	Peer	Status	Relógio
10.0.0.6	ONLINE	10	10.0.0.5	ONLINE	11	10.0.0.5	ONLINE	11
10.0.0.7	ONLINE	15	10.0.0.8	ONLINE	16	10.0.0.8	OFFLINE	17
			10.0.0.9	OFFLINE	13	10.0.0.9	ONLINE	12

Suponha que o peer 10.0.0.5 realize a operação `Obter peers`, enviando uma mensagem `GET_PEERS` primeiro para o peer 10.0.0.6 e depois para o 10.0.0.7.

Após receber a mensagem `PEER_LIST` de 10.0.0.6, a tabela de peers de 10.0.0.5 ganharia duas novas entradas:

Tabela de 10.0.0.5		
Peer	Status	Relógio
10.0.0.6	ONLINE	17
10.0.0.7	ONLINE	15
10.0.0.8	ONLINE	16
10.0.0.9	OFFLINE	13

Após receber a mensagem `PEER_LIST` de 10.0.0.7, devido aos valores do relógio, apenas a entrada referente ao peer 10.0.0.8 seria atualizada, enquanto a entrada referente ao peer 10.0.0.9 permanece a mesma:

Tabela de 10.0.0.5		
Peer	Status	Relógio
10.0.0.6	ONLINE	17
10.0.0.7	ONLINE	18
10.0.0.8	OFFLINE	17
10.0.0.9	OFFLINE	13

4 Comando `Buscar`

O comando `Buscar` primeiramente varre todos os peers conhecidos com status ONLINE em busca de arquivos compartilhados disponíveis. Em seguida, uma lista compilada de todos os arquivos disponíveis em todos os peers é exibida para o usuário, que pode escolher um arquivo para fazer o download.

Para realizar este procedimento, o peer deve enviar uma mensagem do tipo `LS`[3] para cada um dos peers conhecidos com status ONLINE. As mensagens do tipo `LS` não possuem argumentos.

Ao receber uma mensagem do tipo `LS`, o peer deve enviar uma mensagem de resposta do tipo `LS_LIST` contendo os seguintes argumentos:

- A quantidade total arquivos compartilhados;
- Informações dos arquivos compartilhados;
- As informações de cada arquivo devem ser codificadas da seguinte forma: `<nome>:<tamanho>`, onde o nome representa o nome do arquivo e o tamanho é o tamanho do arquivo em bytes. Assuma que o nome do arquivo **não contém** nenhum caractere de espaço.

Após enviar todas as mensagens do tipo `LS` e receber todas as respostas, o programa deve apresentar um menu ao usuário listando todos os arquivos disponíveis.

Por exemplo, suponha que o peer 127.0.0.1:9001 conheça dois peers: o peer 127.0.0.1:9002, que possui os arquivos hello2.txt e loremipsum.txt e o peer 127.0.0.1:9003, que possui os arquivos hello3.txt e loremipsum.txt. Note que o arquivo loremipsum.txt é repetido. Assim, uma possível execução do comando buscar seria a seguinte[4]:

[3] em homenagem ao comando ls presente em sistemas Unix

[4] Não se preocupe em tabular a saída perfeitamente. Mas você ganha pontos de respeito de fizer!

```
   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
>4

       => Atualizando relogio para 10
       Encaminhando mensagem "127.0.0.1:9001 10 LS" para 127.0.0.1:9002
       Resposta recebida: "127.0.0.1:9002 11 LS_LIST 2 hello2.txt:7
loremipsum.txt:5691"
       => Atualizando relogio para 12
       Atualizando peer 127.0.0.1:9002 status ONLINE
       => Atualizando relogio para 13
       Encaminhando mensagem "127.0.0.1:9001 13 LS" para 127.0.0.1:9003
       Resposta recebida: "127.0.0.1:9003 14 LS_LIST 2 hello3.txt:7
loremipsum.txt:5691"
       => Atualizando relogio para 15
       Atualizando peer 127.0.0.1:9003 status ONLINE

Arquivos encontrados na rede:
            Nome            | Tamanho   | Peer
       [ 0] <Cancelar>      |           |
       [ 1] hello2.txt      | 7         | 127.0.0.1:9002
       [ 2] loremipsum.txt  | 5691      | 127.0.0.1:9002
       [ 3] hello3.txt      | 7         | 127.0.0.1:9003
       [ 4] loremipsum.txt  | 5691      | 127.0.0.1:9003

Digite o numero do arquivo para fazer o download:
>
```

Se o usuário escolher um arquivo válido, o programa deve enviar uma mensagem do tipo `DL` para o peer designado. As mensagens do tipo `DL` possuem três argumentos:

- O nome do arquivo escolhido;
- Dois números inteiros, que serão usados apenas na parte 3 do exercícios programa (na parte 2, pode colocar 0 0).

Ao receber uma mensagem do tipo `DL`, o peer deve enviar uma mensagem de resposta do tipo `FILE` contendo os seguintes argumentos:

- O nome do arquivo escolhido;
- Dois números inteiros, que serão usados apenas na parte 3 do exercícios programa;
- O conteúdo do arquivo, codificado em base 64. Vamos usar base 64 pois desta forma é garantido que não haverá nenhum espaço em branco ou quebra de linha que invalidaria o formato de mensagem definido anteriormente (o que poderia acontecer ao enviar o arquivo em formato binário). É recomendável utilizar alguma biblioteca pronta[5]

[5]Em python, por exemplo, basta importar o módulo base64

para fazer a codificação e decodificação para base 64, porém você fazer a sua própria implementação se estiver se sentindo aventureira;

- **Importante:** leia o arquivo em baixo nível (modo binário), não como texto, antes de fazer a codificação para base 64.

Após receber a resposta, o peer que iniciou o download deve decodificar o conteúdo do arquivo para binário e salvar o conteúdo no diretório de compartilhamento, usando o nome do arquivo escolhido. Se o arquivo já existir, apenas sobrescreva. Após finalizar a escrita do arquivo no diretório local, exiba uma mensagem com o formato `"Download do arquivo <nome do arquivo> finalizado."` e retorne ao menu inicial.

Continuando o exemplo, suponha que o usuário escolheu fazer o download do arquivo hello2.txt. A execução seguiria desta forma:

```
Digite o numero do arquivo para fazer o download:
> 1
arquivo escolhido hello2.txt
=> Atualizando relogio para 16
Encaminhando mensagem "127.0.0.1:9001 16 DL hello2.txt 0 0" para 127.0.0.1:9002
Resposta recebida: "127.0.0.1:9002 17 FILE hello2.txt 0 0 aGVsbG8yCg=="
=> Atualizando relogio para 18
Atualizando peer 127.0.0.1:9002 status ONLINE

Download do arquivo hello2.txt finalizado.
```

5 Demais comandos

Os demais comandos (`Exibir estatísticas` e `Alterar tamanho de chunk`) serão implementados na parte 3 do EP.

6 Implementação

A parte 2 do exercício deverá ser realizada com a mesma dupla que a parte 1 foi realizada.

Da mesma forma que a parte 1, considere que os programas serão testados em um ambiente Linux (Ubuntu 22.04). Parte da correção depende da execução da aplicação. Portanto, a nota final do exercício será afetada significativamente se as instruções para a compilação e execução do programa não puderem ser seguidas.

7 Entrega e critérios de avaliação

A entrega deve ser feita no eDisciplinas até o dia 04/05/2025. É suficiente que apenas um integrante da dupla faça a submissão. A entrega deve conter, dentro de um arquivo zip, um relatório em formato PDF, o código fonte e explicações detalhadas de como compilar e executar o código.

O relatório deve conter as decisões de projeto feitas pelos integrantes do grupo. Exemplos de algumas perguntas interessantes a serem respondidas no relatório:

- Como as escolhas feitas na implementação da parte 1 influenciaram as alterações necessárias para a parte 2?
- Foi necessário refatorar parte do código?
- Quais testes foram feitos?
- Quais dificuldades foram enfrentadas?
- Inclua outras informações que achar relevantes.

A nota da parte 2 do EP será atribuída de acordo com o seguinte critério:

Execução do código	6 pontos
Relatório	2 pontos
Aderência à especificação	2 pontos

Se for detectado plágio, todas as duplas envolvidas terão a nota do EP zerada.

Bom exercício!