UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

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

Exercício Programa: parte 3

Prof. Dr. Renan Cerqueira Afonso Alves

Sumário

1 Introdução

Nesta segunda parte do exercício, vamos melhorar a funcionalidade de download de arquivos, permitindo fazer o download de forma fragmentada a partir de vários peers. Além disso, vamos implementar os últimos comandos relacionados à alteração do tamanho de chunk e à exibição de estatísticas (itens 5 e 6 do menu).

```
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

Na inicialização do programa, defina o tamanho de chunk padrão como 256. O formato de mensagem permanece da mesma forma especificada na parte 1 do exercício. Releia trechos do enunciado das partes 1 e 2, caso ache necessário.

2 Comando `Alterar tamanho de chunk`

A palavra chunk significa "pedaço", em inglês. Nesta última parte do EP, o download dos arquivos será feito em pedaços, possivelmente a partir de peers diferentes. O propósito deste comando é definir o tamanho do chunk que será usado no momento em que um download for realizado. Este valor é armazenado localmente, portanto nenhuma mensagem é enviada ao executar este comando. Veja um exemplo de uso deste comando:

```
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
> 6

Digite novo tamanho de chunk:
> 128
       Tamanho de chunk alterado: 128
```

Na inicialização do programa, defina o tamanho de chunk padrão como 256.

3 Alterações no funcionamento do comando `Buscar`

Da mesma forma que na parte 2 do EP, o comando `Buscar` primeiramente varre todos os peers conhecidos com status ONLINE em busca de arquivos compartilhados disponíveis. Porém, a lista de todos os arquivos disponíveis em todos os peers deve ser compilada de forma diferente, agrupando arquivos iguais.

A rigor, para ter certeza de que dois arquivos são iguais, seria necessário fazer o download dos dois arquivos e fazer uma comparação bit a bit. Contudo, este procedimento seria muito custoso em uma rede peer-to-peer. Uma forma mais econômica de determinar se dois arquivos são iguais é através da comparação do hash do conteúdo dos arquivos. Neste exercício programa, por simplicidade, vamos considerar que dois arquivos são iguais se tiverem **o mesmo nome e o mesmo tamanho**.

Desta forma, o formato das mensagens `LS` e `LS_LIST` permanece inalterado, apenas o processamento do conteúdo das mensagens `LS_LIST` deve ser alterado.

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 `milbytes.txt` e o peer 127.0.0.1:9003, que possui os arquivos `hello3.txt` e `milbytes.txt`. Note que há um arquivo com nome `milbytes.txt` nos dois peers.

Assim, uma possível execução do comando `buscar`, executado no peer 127.0.0.1:9001, seria a seguinte:

```
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 7
   Encaminhando mensagem "127.0.0.1:9001 7 LS" para 127.0.0.1:9002
   Resposta recebida: "127.0.0.1:9002 9 LS_LIST 2 hello2.txt:7 milbytes.txt:1000"
   => Atualizando relogio para 10
   Atualizando peer 127.0.0.1:9002 status ONLINE
   => Atualizando relogio para 11
   Encaminhando mensagem "127.0.0.1:9001 11 LS" para 127.0.0.1:9003
   Resposta recebida: "127.0.0.1:9003 13 LS_LIST 2 hello3.txt:7 milbytes.txt:1000"
   => Atualizando relogio para 14
   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] hello3.txt    | 7         | 127.0.0.1:9003
       [ 3] milbytes.txt  | 1000      | 127.0.0.1:9002, 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 os peers designados. As mensagens do tipo `DL` possuem três argumentos:

- O nome do arquivo escolhido;
- O tamanho de chunk **local**;
- O índice do chunk que deve ser feito o download. Por exemplo, se o tamanho do chunk for 256 e o arquivo possuir 1000 bytes, o arquivo será dividido em quatro chunks (indexados a partir de 0). O chunk 3 (o último) terá apenas 232 bytes.

Cada chunk precisa ser adquirido apenas uma vez. Cada chunk pode ser obtido de peers diferentes, se houver mais de um disponível. O objetivo é paralelizar o download, aumentando a taxa de transferência. A forma como o download dos chunks é distribuído entre os peers disponíveis é livre (o relatório deve conter uma explicação e justificativa da escolha feita).

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;
- O tamanho de chunk sendo enviado;
- O índice do chunk que está sendo enviado;
- O conteúdo do chunk do arquivo, codificado em base 64;

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

Após receber as respostas, o peer que iniciou o download deve decodificar os chunks para binário, colocá-los na ordem correta 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 milbytes.txt. Uma possível execução seguiria desta forma:

```
Arquivos encontrados na rede:
            Nome           | Tamanho   | Peer
       [ 0] <Cancelar>     |           |
       [ 1] hello2.txt     | 7         | 127.0.0.1:9002
       [ 2] hello3.txt     | 7         | 127.0.0.1:9003
       [ 3] milbytes.txt   | 1000      | 127.0.0.1:9002, 127.0.0.1:9003

Digite o numero do arquivo para fazer o download:
>3
arquivo escolhido milbytes.txt
=> Atualizando relogio para 15
Encaminhando mensagem "127.0.0.1:9001 15 DL milbytes.txt 256 0" para 127.0.0.1:9002
=> Atualizando relogio para 16
Encaminhando mensagem "127.0.0.1:9001 16 DL milbytes.txt 256 1" para 127.0.0.1:9003
Resposta recebida: "127.0.0.1:9002 17 FILE milbytes.txt 256 0 LS0tLS0tLS0tLS0tLS0tLS"
=> Atualizando relogio para 18
Resposta recebida: "127.0.0.1:9003 18 FILE milbytes.txt 256 1 ZnNoZ2RmaHJ0anlqeXQtLS"
Atualizando peer 127.0.0.1:9002 status ONLINE
=> Atualizando relogio para 19
Atualizando peer 127.0.0.1:9003 status ONLINE
=> Atualizando relogio para 20
Encaminhando mensagem "127.0.0.1:9001 20 DL milbytes.txt 256 2" para 127.0.0.1:9002
=> Atualizando relogio para 21
Encaminhando mensagem "127.0.0.1:9001 21 DL milbytes.txt 256 3" para 127.0.0.1:9003
Resposta recebida: "127.0.0.1:9002 22 FILE milbytes.txt 256 2 LS0tLS0tLS0tLS0tLS0tLS"
=> Atualizando relogio para 23
Atualizando peer 127.0.0.1:9002 status ONLINE
Resposta recebida: "127.0.0.1:9003 23 FILE milbytes.txt 232 3 ZWdkIGJnZm55IG11bWl1bC"
=> Atualizando relogio para 24
Atualizando peer 127.0.0.1:9003 status ONLINE

Download do arquivo milbytes.txt finalizado.
```

Note que as mensagens do tipo `FILE` não foram exibidas por completo, para não poluir a saída do programa. Fique à vontade para fazer o mesmo na sua implementação.

4 Comando `Exibir estatísticas`

Vamos coletar algumas estatísticas sobre o programa para analisar o seu desempenho dependendo de algumas variáveis: o tamanho do chunk, a quantidade de peers e o tamanho

do arquivo.

Assim, ao fazer download de um arquivo, o programa deve medir quanto tempo levou para terminar o download. Não inclua o tempo para escrever o arquivo em disco. O método para fazer a medição de tempo deve ser especificado no relatório.

Desta forma, para cada tripla (tamanho do chunk, quantidade de peers e tamanho do arquivo), o programa deve calcular o tempo médio para fazer downloads com estas características, bem como o desvio padrão associado.

Após a escolha da opção `Exibir estatísticas` no menu, devem ser exibidas as estatísticas. Uma possível saída deste comando seria:

```
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
>5

Tam. chunk | N peers  | Tam. arquivo | N | Tempo [s] | Desvio
      256 | 1        | 7            | 2   | 0.00137   | 0.00101
      256 | 2        | 1000         | 3   | 0.00346   | 0.00174
      128 | 2        | 1000         | 1   | 0.00837   | 0
```

Você pode formatar a saída de forma diferente, mas as colunas devem ser: o tamanho do chunk, quantidade de peers, tamanho do arquivo, tamanho da amostra, tempo médio e desvio padrão do tempo.

5 Implementação

A parte 3 do exercício deverá ser realizada com a mesma dupla que as partes 1 e 2 foram realizadas.

Da mesma forma que as partes anteriores, 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.

6 Entrega e critérios de avaliação

A entrega deve ser feita no eDisciplinas na data indicada. 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 das partes anteriores influenciaram as alterações necessárias para a parte 3?
- Quais testes foram feitos?
- Como foi feita a distribuição de chunks entre os peers disponíveis?
- Como foi medido o tempo de download?
- Resultados de experimentos
- Inclua outras informações que achar relevantes.

Nesta parte do exercício programa será requerido a realização de alguns experimentos para investigar como o tempo de download se comporta de acordo com alguns fatores. Sugere-se realizar dois experimentos:

1. Crie arquivos de tamanho 1KB, 10KB e 100KB e coloque-os em dois peers. Um terceiro peer faz o download destes arquivos, variando-se o tamanho do chunk em com os valores de 1 byte, 256 bytes e 512 bytes (faça o download algumas vezes, buscando relevância estatística). Estude a influência do tamanho de chunk no tempo de download dos arquivos, buscando explicar a razão dos dados observados.
2. Crie arquivos de tamanho 1KB, 10KB e 100KB. Variando o número de peers que possuem estes arquivos (sugestão 2, 3 e 4) faça o download a partir de um outro fixando o tamanho do chunk em 256 bytes (faça o download algumas vezes, buscando relevância estatística). Estude a influência do número de peers no tempo de download dos arquivos, buscando explicar a razão dos dados observados.

Estas são apenas sugestões, você pode fazer experimentos diferentes se preferir, justificando o motivo no relatório.

A nota da parte 3 do EP será atribuída de acordo com o seguinte critério:

Execução do código	4 pontos
Relatório	2 pontos
Experimentos e análise	4 pontos

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

Bom exercício!