UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

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

Exercício Programa: parte 1

Prof. Dr. Renan Cerqueira Afonso Alves

Sumário

1 Introdução

O objetivo deste exercício programa é implementar um sistema de compartilhamento de arquivos peer-to-peer simplificado chamado **EACHare**. Cada peer participante da rede disponibiliza um conjunto de arquivos e mantém uma lista de outros peers conhecidos. Qualquer peer da rede pode perguntar aos seus peers conhecidos quais arquivos eles possuem e selecionar um dos arquivos para fazer o download.

Nesta primeira parte do exercício programa, a principal funcionalidade que vamos começar a implementar é o gerenciamento de peers conhecidos.

Note que o EP é cumulativo, ou seja, as próximas partes dependerão da boa implementação desta parte.

Dicas

1. Implemente o exercícios aos poucos, não deixe para fazer tudo de uma vez ao final do prazo
2. Planeje a implementação e os testes do programa começando pelas funcionalidades mais simples.
3. Testar o programa através do menu pode ser trabalhoso pois requer a digitação repetida de valores. Portanto, recomenda-se criar testes com valores fixos para agilizar o processo. Não esqueça de remover estes testes antes de entregar a versão final, deixando apenas o menu textual.
4. Dúvidas sobre o enunciado podem ser colocadas no fórum do eDisciplinas específico do EP ou discutidas ao final das aulas. Lembre-se de **não colocar** trechos de seu código no fórum!

2 Especificação

Cada peer pertencente à rede de compartilhamento deverá ser identificado através de um endereço e de uma porta, funcionando como um servidor TCP. O programa deve receber como parâmetro a sua identificação na forma `<endereço>:<porta>`. Como todos os integrantes do sistema peer-to-peer executam o mesmo código, este parâmetro permite diferenciar os peers entre si, permitindo até mesmo executar diversos peers na mesma máquina, apenas trocando a porta de cada um. O endereço deve ser o endereço IP (vamos considerar apenas IPv4) atribuído a alguma interface de rede disponível na máquina. É possível usar o endereço de loopback (127.0.0.1), que está sempre disponível.

Além disso, há dois outros parâmetros: um arquivo de texto contendo uma lista inicial de peers conhecidos e o nome do diretório que contém os arquivos compartilhados pelo peer em questão. A invocação do programa na linha de comando é feita desta forma:

```
$ ./eachare <endereco>:<porta> <vizinhos.txt> <diretorio_compartilhado>
```

O arquivo com a lista de peers deve conter endereços de outros peers do sistema na forma `<endereço>:<porta>`, um por linha. Estes serão os peers que podem ser contatados inicialmente, mas outros poderão ser descobertos durante a execução do programa, como veremos adiante. Um exemplo de conteúdo deste arquivo é mostrado abaixo:

```
192.168.100.50:5000
192.168.100.50:5001
192.168.100.60:5000
192.168.100.70:5001
```

2.1 Formato de mensagem

Por simplicidade, vamos utilizar um formato de mensagem codificado em texto puro (apesar de não ser muito eficiente). Toda mensagem será composta de um cabeçalho com três itens, separados por um espaço em branco:

- A identificação do servidor de origem da mensagem (no formato `<endereço>:<porta>`);
- Um número que representa o valor atual do relógio (clock) local ; e
- Um código que indica o tipo de mensagem enviada.

Dependendo do tipo da mensagem, haverá uma lista de argumentos após o cabeçalho, cada argumento também separado por espaço. Em todo caso, a mensagem deve ser sempre terminada por um caractere de nova linha (`\n`). Veja o formato geral da mensagem na Figura 1. Note que estamos especificando um protocolo camada de aplicação.

```
<ORIGEM> <CLOCK> <TIPO>[ ARGUMENTO1 ARGUMENTO2...]\n
```

Figura 1: Formato geral de mensagem

Toda tentativa de envio de mensagem deve ser exibida na saída padrão no formato `"Encaminhando mensagem <mensagem> para <endereço:porta destino>"`. No exemplo abaixo, a mensagem foi enviada por um peer operando na porta 9002 no endereço 127.0.0.1, o valor de relógio é igual a 1 e o tipo de mensagem é HELLO.

```
Encaminhando mensagem "127.0.0.1:9002 1 HELLO" para 127.0.0.1:9001
```

2.2 Funcionamento do relógio local

O relógio local de cada peer deve ser inicializado com zero. Nesta parte 1 do exercício programa[1], o relógio deve funcionar da seguinte forma:

- **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);
- Ao receber uma mensagem, 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>"`

2.3 Inicialização do programa

Ao inicializar, o programa deve processar os parâmetros passados:

- Deve ser criado um socket TCP com o endereço e a porta indicados (primeiro parâmetro);
- Para cada peer no arquivo (segundo parâmetro):
  - Cada peer deve possuir um status associado, que pode ser `ONLINE` ou `OFFLINE`;
  - Inicialmente, considere que os vizinhos estão no estado `OFFLINE`;
  - A seguinte mensagem deve ser exibida na saída padrão: `"Adicionando novo peer <endereço>:<porta> status OFFLINE"`; e
- Verificar se o diretório de compartilhamento (terceiro parâmetro) é um diretório válido e pode ser lido. Se essas condições não forem verdadeiras, termine a execução do programa.

Após realizar estes passos, o peer deve escutar por conexões no seu endereço e porta designados e exibir um menu conforme mostrado na Figura 2.

```
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 2: Menu principal

[1] na parte 2 usaremos um relógio de Lamport

2.4 Comando `Listar peers`

Ao executar este comando, o programa deve exibir na tela uma lista com todos os outros peers conhecidos, bem como o status atual de cada um. O usuário poderá escolher um dos peers para enviar uma mensagem do tipo `HELLO`, ou apenas retornar para o menu inicial.

As mensagens do tipo `HELLO` não possuem argumentos. Se a mensagem for enviada com sucesso, atualize o status do peer destino escolhido para ONLINE, caso contrário, atualize o status do peer destino para OFFLINE..

Ao receber uma mensagem do tipo `HELLO`, o peer deve adicionar o remetente ao seu conjunto de peers conhecidos com status ONLINE, ou, caso o remetente já esteja na lista, apenas atualizar o status para ONLINE.

Sempre que o status de um peer for atualizado, exiba uma mensagem com o seguinte formato: `"Atualizando peer <endereço>:<porta> status <ONLINE ou OFFLINE>"`.

No exemplo abaixo, o peer 127.0.0.1:9001 foi inicializado com o peer 127.0.0.1:9002 em sua lista de peers conhecidos:

```
Adicionando novo peer 127.0.0.1:9002 status OFFLINE

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

Lista de peers:
       [0] voltar para o menu anterior
       [1] 127.0.0.1:9002 OFFLINE (clock: 0)
> 1
   => Atualizando relogio para 1
   Encaminhando mensagem "127.0.0.1:9001 1 HELLO" para 127.0.0.1:9002
   Atualizando peer 127.0.0.1:9002 status ONLINE
```

O peer 127.0.0.1:9002 ao receber a mensagem de `HELLO` exibe as seguintes mensagens:

```
   Mensagem recebida: "127.0.0.1:9001 1 HELLO"
   => Atualizando relogio para 1
   Atualizando peer 127.0.0.1:9001 status ONLINE
```

2.5 Comando `Obter peers`

Ao executar este comando, o peer deve percorrer a sua lista de peers conhecidos e enviar uma mensagem do tipo `GET_PEERS` para cada um dos peers conhecidos, independentemente de seu status. As mensagens do tipo `GET_PEERS` não possuem argumentos. Se a mensagem for enviada com sucesso, atualize o status do peer destino para ONLINE, caso contrário, atualize o status do peer destino para OFFLINE.

Ao receber uma mensagem do tipo `GET_PEERS`, o peer deve enviar uma mensagem de resposta, seguindo o formato definido na Figura 1 do tipo `PEER_LIST` contendo os seguintes argumentos:

- A quantidade total de vizinhos descritos na mensagem;
- Cada um dos peers conhecidos, exceto o remetente;
- Cada peer deve ser codificado da seguinte forma: `<endereço>:<porta>:<status>:0`, onde status pode ser as strings ONLINE ou OFFLINE, e, nesta versão do EP, o número no final será sempre zero e não será usado.

Ao receber a resposta, o peer deve atualizar o seu conjunto de peers conhecidos, adicionando os peers novos e atualizando o status dos peers que já eram conhecidos.

Para o exemplo abaixo, vamos assumir o seguinte cenário:

- Há quatro peers no total: 127.0.0.1:9001, 127.0.0.1:9002, 127.0.0.1:9003, e 127.0.0.1:9004
- Inicialmente, o peer 127.0.0.1:9001 conhece apenas o peer 127.0.0.1:9002
- Inicialmente, o peer 127.0.0.1:9002 conhece apenas os peers 127.0.0.1:9001 e 127.0.0.1:9003
- Inicialmente, o peer 127.0.0.1:9003 conhece apenas o peer 127.0.0.1:9004
- Todos os peers estão com status ONLINE

Após o peer 127.0.0.1:9001 executar o comando `Obter peers` duas vezes, ele obtém conhecimento de todos os peers disponíveis. Veja a saída esperada (menu parcialmente omitido):

```
Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       (...)
>1

Lista de peers:
       [0] voltar para o menu anterior
       [1] 127.0.0.1:9002 ONLINE
>0

Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       (...)
>2

       => Atualizando relogio para 2
       Encaminhando mensagem "127.0.0.1:9001 2 GET_PEERS" para 127.0.0.1:9002
       Resposta recebida: "127.0.0.1:9002 4 PEER_LIST 1 127.0.0.1:9003:ONLINE:0"
       => Atualizando relogio para 5
       Atualizando peer 127.0.0.1:9002 status ONLINE
```

```
       Adicionando novo peer 127.0.0.1:9003 status ONLINE

Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       (...)
>1

Lista de peers:
       [0] voltar para o menu anterior
       [1] 127.0.0.1:9002 ONLINE
       [2] 127.0.0.1:9003 ONLINE
>0

Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       (...)
>2

       => Atualizando relogio para 6
       Encaminhando mensagem "127.0.0.1:9001 6 GET_PEERS" para 127.0.0.1:9002
       Resposta recebida: "127.0.0.1:9002 7 PEER_LIST 1 127.0.0.1:9003:ONLINE:0"
       => Atualizando relogio para 8
       Atualizando peer 127.0.0.1:9002 status ONLINE
       Atualizando peer 127.0.0.1:9003 status ONLINE
       => Atualizando relogio para 9
       Encaminhando mensagem "127.0.0.1:9001 9 GET_PEERS" para 127.0.0.1:9003
       Resposta recebida: "127.0.0.1:9003 10 PEERS_LIST 2 127.0.0.1:9002:ONLINE:3
                                                          127.0.0.1:9004:ONLINE:0"
       => Atualizando relogio para 11
       Atualizando peer 127.0.0.1:9003 status ONLINE
       Atualizando peer 127.0.0.1:9002 status ONLINE
       Adicionando novo peer 127.0.0.1:9004 status ONLINE

Escolha um comando:
       [1] Listar peers
       [2] Obter peers
       (...)
>1

Lista de peers:
       [0] voltar para o menu anterior
       [1] 127.0.0.1:9002 ONLINE
       [2] 127.0.0.1:9003 ONLINE
       [3] 127.0.0.1:9004 ONLINE
```

2.6 Comando `Listar arquivos locais`

Este comando é executado localmente, ou seja, não envolve envio de mensagens. Ao executar este comando, o conteúdo do diretório de arquivos compartilhados deve ser exibido na saída padrão.

Por exemplo, se o diretório compartilhado possuir os arquivos `hello_world.txt`, `relatorio.doc` e `musica.mp3`, a saída esperada deste comando seria conforme indicado abaixo:

```
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

   hello_world.txt
   relatorio.doc
   musica.mp3
```

2.7 Comando `Sair`

Ao executar este comando, o peer deve percorrer a sua lista de peers conhecidos e enviar uma mensagem do tipo `BYE` para cada um dos peers que estiverem com status ONLINE. As mensagens do tipo `BYE` não possuem argumentos. Após enviar as mensagens, o peer deve parar de esperar por conexões e o programa deve terminar a sua execução.

Ao receber uma mensagem do tipo `BYE`, o peer deve atualizar o status do remetente para OFFLINE.

Lembre-se de, sempre que o status de um peer for atualizado, exibir uma mensagem com o seguinte formato na saída padrão: `"Atualizando peer <endereço>:<porta> status <ONLINE ou OFFLINE>"`.

No exemplo abaixo, o peer 127.0.0.1:9001 possui apenas o peer 127.0.0.1:9002 em sua lista de peers conhecidos (os valores de relógio são ilustrativos):

```
   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...
       => Atualizando relogio para 20
       Encaminhando mensagem "127.0.0.1:9001 20 BYE" para 127.0.0.1:9002
```

```
$
```

Ao processar esta mensagem, o peer 127.0.0.1:9002 exibiria a seguinte mensagem:

```
Mensagem recebida: "127.0.0.1:9001 20 BYE"
=> Atualizando relogio para 21
Atualizando peer 127.0.0.1:9001 status OFFLINE
```

2.8 Demais comandos

Os demais comandos serão implementados nas partes 2 e 3 do EP.

3 Implementação

O exercício deverá ser realizado em duplas, a serem informadas no eDisciplinas na atividade Escolha de dupla EP. Os integrantes do grupo podem escolher a linguagem de programação que desejarem para implementar o exercício programa, desde que a especificação fornecida seja seguida. Como consequência, será necessário incluir instruções detalhadas de como compilar e executar o código fonte entregue.

A comunicação entre os nós deve ser feita através de sockets TCP. Decisões a respeito do paradigma de programação, organização de código e decisões de projeto em geral ficam a cargo dos integrantes do grupo, e devem ser justificadas no relatório.

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.

4 Entrega e critérios de avaliação

A entrega deve ser feita no eDisciplinas até o dia 30/03/2025. 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:

- Quais foram as maiores dificuldades enfrentadas?
- Qual foi o paradigma de programação escolhido e por quê?
- Como foi feita a divisão do programa em threads?
- Foi escolhido utilizar operações bloqueantes ou não bloqueantes para enviar e receber dados?
- Quais as estruturas de dados usadas e por quê?
- Se utilizar orientação objetos, quais as classes escolhidas?
- Quais testes foram feitos?
- Inclua outras informações que achar relevantes.

A nota da parte 1 do EP será atribuída de acordo com o seguinte critério:

Execução do código	4 pontos
Relatório	4 pontos
Aderência à especificação	2 pontos

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

Bom exercício!