Sistemas Distribuídos
Projeto de Programação - Gossip

Revisão 1: 07/09/2022.

1. Definição do Sistema

Crie um sistema P2P não-estruturado que permita encontrar o peer que possui uma determinada informação a ser procurada, utilizando UDP como protocolo da camada de transporte e *Gossip* como protocolo de busca.

2. Recomendação Inicial

Se nunca programou com TCP ou UDP, recomendo assistir o vídeo do link https://www.youtube.com/watch?v=nysfXweTI7o e implementar os exemplos mostrados. Só assistir o vídeo não lhe será de utilidade quando tiver que implementar funcionalidades mais complexas.

3. Visão geral do sistema

O sistema será composto por muitos peers (com IP e porta conhecidas). Cada peer atua tanto como provedor de informações (neste caso dos nomes dos arquivos que possui) quanto como buscador destas.

Considere que quatro peers *peer1*, *peer2*, *peer3* e *peer4* fazem parte do sistema P2P. Um peer, por exemplo o *peer1*, poderá procurar pelo nome de um arquivo e enviar essa requisição a outro (e.g., *peer4*), escolhido de forma aleatória. Caso o *peer4* tenha o arquivo, responderá ao *peer1* diretamente com seus dados (e.g., IP:porta de *peer4*). Caso o *peer4* não tenha o arquivo, escolherá outro peer (e.g., *peer3*) e repetirá o processo.

4. Funcionalidades do Peer X

a) Recebe e responde simultaneamente (obrigatório com threads) requisições dos peers. Por 'simultaneamente' entenda-se que o peer deverá poder realizar outras funcionalidades enquanto está fazendo requisições ou esperando por elas.

b) Inicialização: captura do teclado o IP e porta do peer X, a pasta onde estão localizados seus arquivos, e o IP e porta de outros dois peers.

c) Monitoramento da pasta: cada 30 segundos o peer verificará se na pasta (capturada na inicialização) houveram modificações, ou seja se foram inseridos ou removidos arquivos. A lista de arquivos deverá estar armazenada em alguma estrutura na memória, por exemplo, uma lista ou um hash.

d) Envio de uma mensagem SEARCH: escolhe de forma aleatória um peer (obtidos na inicialização) a quem enviar a requisição, contendo o nome do arquivo procurado.

e) Recebe uma mensagem SEARCH: 1º importante, descarte requisições já processadas (i.e., já respondidas ou encaminhadas). 2º Caso o peer X tenha o arquivo, responderá diretamente a quem **inicialmente** realizou o SEARCH com uma mensagem RESPONSE (que contém o IP:porta do peer X). 3º Caso X não tenha o arquivo, encaminhará a requisição, repetindo o processo de envio do SEARCH (item d).

f) Recebe uma mensagem RESPONSE: descarte requisições já processadas.

g) Monitoramento do envio do SEARCH. No item d) quando o peer realiza um envio, pode que nunca receba a mensagem RESPONSE (pois ninguém possui o arquivo). Assim, o peer deve ter algum mecanismo para tratar esse caso (por exemplo, usando *timeouts*).

Observações:

- Toda comunicação entre peer ←→ peer será por UDP e deverá obrigatoriamente transferir uma classe Mensagem criada por você.

5. Mensagens (prints) apresentadas na console

Na console do peer deverão ser apresentadas “exatamente” (nem mais nem menos) as seguintes informações

- Menu interativo (por console) que permita realizar a escolha somente das funções INICIALIZA e SEARCH.
    - No caso do INICIALIZA, deve capturar do teclado o IP:porta, a pasta onde se encontram os arquivos (e.g., c:\temp\peer1\, c:\temp\peer2\, etc.) e mais outros dois IP:porta. Faça print “arquivos da pasta: [só nomes dos arquivos]”.
    - No caso do SEARCH, deve capturar do teclado só o nome do arquivo com sua extensão (e.g., aula.mp4). A busca será exatamente por esse nome. Note que não deve capturar a pasta.
- Cada 30 segundos, print “Sou peer [IP]:[porta] com arquivos [só nomes dos arquivos]”. Substitua a informação entre os parênteses com as reais. Por exemplo: Sou peer 127.0.0.1:8776 com arquivos aula1.mp4 aula2.mp4
- Quando receber a mensagem RESPONSE (caso não tenha sido processada antes), print “peer com arquivo procurado: [IP:porta] [arquivo_procurado]”.
- Se nunca receber a mensagem RESPONSE, print “ninguém no sistema possui o arquivo [arquivo_procurado]”.

- Quando receber uma mensagem SEARCH:
    - Caso já tenha processado a requisição, print “requisição já processada para [arquivo_procurado]”.
    - Caso não tenha processado a requisição:
        - Se tiver o arquivo, print “tenho [arquivo_procurado] respondendo para [IP:porta]. Note que esse IP:porta é de quem inicialmente fez a busca.
        - Se não tiver o arquivo, print “não tenho [arquivo_procurado], encaminhando para [IP:porta]. Note que esse IP:porta é aleatório.

6. Teste realizado pelo professor

O professor compilará o código usando o javac da JDK 1.8.

Após a compilação, o professor abrirá 4 consoles (no Windows, seria o CMD.EXE, também conhecido como prompt) correspondendo a 4 peers. A partir das consoles, o professor realizará os testes do funcionamento do sistema. Exemplo das consoles pode ser observado no link: https://www.youtube.com/watch?v=FOwKxw9VYqI

Cabe destacar que:

- Seu código não deve estar limitado a 4 peers, suportando mais do que 4.
- Você não precisará de 4 computadores para realizar a atividade. Basta abrir as 4 consoles.

7. Código fonte

- Deverá criar somente as classes Peer e Mensagem. A última deverá ser utilizada **obrigatoriamente** para o envio e recebimento de informações (nas requisições e respostas).
    - Caso não use, envie e receba a classe Mensagem, serão descontados 3 pontos da nota final.
    - Caso envie novas classes, serão descontados 3 pontos da nota final.
    - A única exceção é a criação das classes para Threads, mas elas deverão ser criadas dentro da classe Peer (e.g, classes aninhadas).

- O código fonte deverá apresentar claramente (usando comentários) os trechos de código que realizam as funcionalidades mencionadas na Seção 4.

- O código fonte deverá ser compilado e executado por uma JDK 1.8 (também denominada JDK 8). Você poderá utilizar todas as classes e pacotes existentes nela. O link destas é https://docs.oracle.com/javase/8/docs/api/

- O uso de bibliotecas que realizem parte das funcionalidades pedidas não será aceito. Caso tenha dúvidas de alguma específica, pergunte ao professor.

8. Entrega Final

A entrega é individual e consistirá em: (a) um relatório [SeuRA].pdf; (b) o código fonte do programa com as pastas e os .java respectivos (c) o link para um vídeo, dentro do relatório, que mostre o funcionamento. A entrega será realizada como definida no plano de ensino, não sendo aceita por outras formas.

- Caso queira utilizar outra linguagem de programação, deve enviar um email ao professor até a terceira semana de aula e esperar a aceitação do professor. Após essa data, será obrigatório o envio em Java.
- Caso tenha utilizado uma biblioteca permitida pelo professor, envie-a também.

O relatório deverá ter as seguintes seções:

I. Nome e RA do participante

II. Link do vídeo do funcionamento (*screencast*). O vídeo deverá conter no máximo 5 minutos, mostrando a compilação e o funcionamento do código nas quatro consoles. **Não envie o vídeo**, envie só o link do vídeo, o qual pode disponibilizar no Youtube ou em outro lugar semelhante (como vimeo). Lembre-se de dar permissão para visualizá-lo.

III. No vídeo do ponto anterior, deve realizar e mostrar:

   a. Funcionamento de uma busca por um arquivo que existe (em outro peer).
   b. Funcionamento de uma busca por um arquivo que não existe no sistema.
   c. Funcionamento do monitoramento da pasta quando se insere um novo arquivo nela e a busca desse novo arquivo por outro peer.

IV. Para cada funcionalidade do peer, uma breve explicação em “alto nível” de como foi realizado o tratamento da requisição. Na explicação DEVE mencionar as linhas do código fonte que fazem referência.

V. Explicação do uso das threads, mencionado as linhas do código fonte que fazem referência.

VI. Links dos lugares de onde baseou seu código (caso aplicável). Prefiro que insira os lugares a encontrar na Internet algo similar.

9. Observações importantes sobre a avaliação

A seguir mencionam-se alguns assuntos que descontarão a nota.

- Código fonte não compila ou usa bibliotecas externas à JDK sem aprovação do professor (nota zero).
- Não transfere a classe Mensagem no envio/recebimento (menos 3 pontos)
- Não usou Threads no Peer (menos 4 pontos).
- Não enviou o relatório (menos 2 pontos).
- Não enviou o link do vídeo, o link não está disponível ou o vídeo não mostra o funcionamento pedido (menos 2 pontos)
- Enviou outros arquivos do código fonte além dos citados na Seção 7, por exemplo os .class ou outras classes (menos 2 pontos).
- Código fonte sem comentários que referenciem as funcionalidade da Seção 4 (menos 2 pontos).

10. Links recomendados

- Para baixar a JDK 8
  https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html

- Vídeo explicativo sobre programação UDP, TCP e Threads:
  https://www.youtube.com/watch?v=nysfXweTI7o

- Informações sobre programação com UDP podem ser encontradas em:
  https://www.baeldung.com/udp-in-java
  https://www.geeksforgeeks.org/working-udp-datagramsockets-java/
  https://docs.oracle.com/javase/tutorial/networking/datagrams/index.html

- Informações sobre programação com TCP podem ser encontradas em:
  https://www.baeldung.com/a-guide-to-java-sockets

- Informações sobre Threads, que permitem que o servidor ou peer receba e envie informações de forma simultânea:
  https://www.baeldung.com/a-guide-to-java-sockets (Seção 6: TCP com muitos clientes)
  https://www.tutorialspoint.com/java/java_multithreading.htm

11. Ética

Cola, fraude, ou plágio implicará na nota zero a todos os envolvidos em todas as avaliações e exercícios programáticos da disciplina.

Perguntas Frequentes (FAQ)

1. Por onde começo?

Como mencionado na Seção 2, recomendo assistir e implementar os exemplos mostrados no vídeo. Com esses exemplos "implementados" e “testados”, você terá um cliente e um servidor (concorrente) funcionando e transmitindo mensagens por UDP e TCP. Isso é a base para o projeto.

2. Já implementei os exemplos de UDP e TCP do vídeo e estão funcionando. E agora?

Na computação temos a frase "dividir para conquistar" e podemos usar esse conceito aqui. Recomendo primeiro implementar o INICIALIZA e ver que está funcionando. Só depois implementará as outras funcionalidades, como SEARCH e a concorrência.

Por exemplo, vamos pensar só no INICIALIZA do lado do Peer. 1º implemente um método que leia as informações do teclado. 2º implemente um método que leia as informações da pasta obtida no ponto anterior e armazene em uma estrutura. 3º implemente um método que cada certo tempo monitora se houveram mudanças na pasta e imprima as informações.

3. Como faço para enviar o objeto Mensagem em vez do String?

Existem algumas formas. Você pode serializar o objeto (não recomendo) ou usar o formato JSON (recomendo). JSON permite transformar um objeto Java a uma String e vice-versa. Caso queira usar o JSON, deve ser através da biblioteca https://github.com/google/gson. Um link interessante: http://tutorials.jenkov.com/java-json/gson.html (só até "Generating JSON From Java Objects").

6. Tenho a estrutura da lista ou hash e criei as Threads que atendem os Peers. Estou tendo problemas de concorrência ao atualizar a estrutura. Como resolvo isso?

Existem diversas formas, como o uso de *locks*, semáforos, *synchronized*, etc. A mais simples é usar uma estrutura que já trate/implemente a concorrência. Para isso, o Java 1.8 possui o pacote *concurrent*. Por exemplo, pode usar o ConcurrentHashMap ou o ArrayBlockingQueue. Um link interessante: https://www.baeldung.com/java-concurrent-map