Sistemas Distribuídos
Projeto de Programação - Napster

Revisão 1: 21/05/2022.

1. Definição do Sistema

Crie um sistema P2P que permita a transferência de arquivos de vídeo gigantes (mais de 1 GB) entre peers, intermediados por um servidor centralizado, utilizando TCP e UDP como protocolo da camada de transporte.

O sistema funcionará de forma similar (porém muito reduzida) ao sistema **Napster**, um dos primeiros sistemas P2P, criado por Shawn Fanning aos 18 anos.

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 1 servidor (com IP e porta conhecidas) e muitos peers. O peer atua tanto como provedor de informação (neste caso de arquivos) quanto como receptor deles.

Inicialmente, o servidor estará disponível para receber requisições dos peers. Quando um PeerX entra no sistema, deve comunicar ao servidor suas informações. O servidor receberá as informações e as armazenará para futuras consultas. Quando um PeerY quiser baixar um vídeo, deverá enviar para o servidor uma requisição com o nome do arquivo. O servidor procurará pelo nome e responderá ao PeerY com uma lista de peers que o contém. O PeerY receberá a lista do servidor e escolherá um dos peers da lista (vamos supor que o escolhido é o PeerZ). A seguir, o PeerY requisitará o arquivo para o PeerZ, quem poderá aceitar o pedido, enviando o arquivo, ou rejeitar o pedido. Finalmente, quando o PeerY baixar o arquivo em uma pasta, a pessoa poderá ir na pasta e visualizá-lo usando um software externo de reprodução, como VLC.

4. Funcionalidades do Servidor

a) Recebe e responde simultaneamente (com threads) requisições dos peers.

b) Requisição JOIN: vinda de um peer que quer entrar na rede. A requisição deve conter as informações mínimas do peer (e.g., nome dos arquivos que possui), as quais devem

ser armazenadas em alguma estrutura de dados no servidor. A resposta do servidor enviada ao peer será a string JOIN_OK.

c) Requisição LEAVE: vinda de um peer que quer sair da rede. As informações desse peer armazenadas no servidor devem ser eliminadas. A resposta do servidor enviada ao peer será a string LEAVE_OK.

d) Requisição SEARCH: vinda de um peer que quer baixar um determinado arquivo. A requisição deve conter somente o nome do arquivo com sua extensão (e.g o string Aula.mp4). A resposta do servidor enviada ao peer será uma lista vazia ou com as informações dos peers que possuem o arquivo.

e) Requisição UPDATE: vinda de um peer que baixou um arquivo. A requisição deve conter o nome do arquivo baixado, o qual será inserido na estrutura de dados que mantém as informações dos peers. A resposta do servidor enviada ao peer será a string UPDATE_OK.

f) Requisição ALIVE: enviada cada 30 segundos pelo servidor aos peers para saber se estão vivos ou se saíram da rede de forma abrupta (i.e., sem avisar com o LEAVE). O servidor receberá a string “ALIVE_OK” se o peer estiver vivo. Caso o servidor não receba a resposta de um determinado peer, deverá eliminar as informações deste da estrutura de dados.

g) Inicialização do servidor: o servidor deve capturar inicialmente o IP. O endereço IP a ser inserido será o 127.0.0.1. Assuma esse IP quando o Peer quiser comunicar-se com o servidor. A porta default (que permitirá aos peers conectar-se com ele) será a 10098. Sobre a captura, ela se dará pelo teclado.

5. Funcionalidades do Peer (assuma o ponto de vista de um PeerX)

a) Recebe e responde simultaneamente (com threads) requisições do servidor e de outros peers.
b) Envia por UDP uma requisição de JOIN ao servidor. Deve esperar o JOIN_OK.
c) Envia por UDP uma requisição de LEAVE ao servidor. Deve esperar o LEAVE_OK.
d) Envia por UDP uma requisição de UPDATE ao servidor. Deve esperar o UPDATE_OK.
e) Envia por UDP uma resposta ALIVE_OK ao servidor.
f) Envia por UDP uma requisição de SEARCH ao servidor. Voltará uma lista como resposta (vazia ou com informações).
g) Para os envios por UDP acima, caso não receba a resposta, deverá realizar novamente a requisição, cuidando de possíveis duplicações (e.g., atrasos na resposta do servidor).
h) Envia por **TCP** um requisição de DOWNLOAD a outro peer. Ver abaixo as possíveis respostas obtidas.

i) Requisição DOWNLOAD: vinda de outro peer (PeerY), pedindo por um determinado arquivo. O PeerX deve verificar se possui o arquivo e, de forma aleatória, aceitar ou rejeitar o pedido. Se aceito o pedido, enviará por **TCP** o arquivo ao PeerY. Se rejeitado, enviará por **TCP** a string DOWNLOAD_NEGADO ao PeerY.

j) Recebimento do arquivo: o arquivo deverá ser armazenado em uma pasta específica do peer.

k) Requisição DOWNLOAD_NEGADO: vinda do peer que rejeitou a requisição de DOWNLOAD. O PeerX deverá realizar o mesmo pedido a outro peer da lista obtida no SEARCH. Caso não haja, realizar o pedido (depois de um determinado período de tempo) ao mesmo peer que o rejeitou.

l) Inicialização do peer: o peer deve capturar inicialmente o IP, porta, e a pasta onde estão (e serão) armazenados seus arquivos. Sobre a captura, ela se dará pelo teclado. Sobre as pastas, cada peer terá sua própria. Por exemplo, se houverem 3 peers, haverá 3 pastas diferentes. Ver o JOIN no item 'Menu interativo' na seção 6.

6. Mensagens (prints) apresentadas na console

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

- Quando receber o JOIN_OK, 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 resposta do SEARCH, print "peers com arquivo solicitado: [IP:porta de cada peer da lista]"

- Quando receber o arquivo, print "Arquivo [só nome do arquivo] baixado com sucesso na pasta [nome da pasta]".

- Quando receber o DOWNLOAD_NEGADO, print "peer [IP]:[porta] negou o download, pedindo agora para o peer [IP]:[porta].

- Menu interativo (por console) que permita realizar a escolha somente das funções JOIN, SEARCH, DOWNLOAD e LEAVE.

    - No caso do JOIN, deve capturar só o IP, porta e a pasta onde se encontram os arquivos (e.g., c:\temp\peer1\, c:\temp\peer2\, etc.). A partir da pasta, seu código procurará nela o nome dos arquivos a serem enviados ao servidor.
    - No caso do SEARCH, deve capturar só o nome do arquivo com sua extensão (e.g., aula.mp4). A busca por ele será exatamente por esse nome. Note que não deve capturar a pasta.
    - No caso do DOWNLOAD, deve capturar só o IP e porta do peer onde se encontra o arquivo a ser baixado. Note que não deve capturar a pasta.

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

- Quando receber o JOIN, print “Peer [IP]:[porta] adicionado com arquivos [só nomes dos arquivos].
- Quando receber o SEARCH, print “Peer [IP]:[porta] solicitou arquivo [só nome do arquivo].
- Se não receber o ALIVE_OK, print “Peer [IP]:[porta] morto. Eliminando seus arquivos [só nome dos arquivos]”.

7. 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). Um deles será o servidor e os outros 3 os 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 3 peers, suportando mais do que 3.
- Inicialmente, o professor levantará (executará) o servidor. A seguir, levantará os peers.
- Você não precisará de 4 computadores para realizar a atividade. Basta abrir as 4 consoles.

8. Código fonte

- Deverá criar somente as classes Servidor, Peer e Mensagem. A última deverá ser utilizada **obrigatoriamente** para o envio e recebimento de informações (nas requisições e respostas). 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 ou Servidor (e.g, classes aninhadas).

- O código fonte deverá apresentar claramente (usando comentários) os trechos de código que realizam as funcionalidades mencionadas nas Seções 4 e 5.

- 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.

9. 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 pelo Moodle, não sendo aceita por outras formas.

- Caso queira utilizar outra linguagem de programação, deve enviar um email ao professor até o dia 05/07 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. O vídeo deverá conter no máximo 3 minutos, mostrando a compilação, o funcionamento do código nas quatro consoles e a reprodução do arquivo transferido. **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. Para cada funcionalidade do servidor, 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.

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. Separe a explicação de servidor e do peer, mencionado as linhas do código fonte que fazem referência.

VI. Explicação de como implementou a transferência de arquivos gigantes.

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

11. 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 arquivos maiores a 1GB (menos 2 pontos).

- Não usou Threads tanto no Peer quanto no Servidor, i.e., ambos (menos 2 pontos).
- Não é possível reproduzir o arquivo transferido com um reprodutor externo (menos 1 ponto)
- 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 e reprodução (menos 2 pontos)
- Enviou outros arquivos do código fonte além dos citados na Seção 8, por exemplo os .class ou outras classes (menos 2 pontos).
- Código fonte sem comentários que referenciem as funcionalidade das seções 4 e 5 (menos 2 pontos)

12. 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

13. 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 do Napster.

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 JOIN e ver que está funcionando. Só depois implementará as outras funcionalidades, como LEAVE e SEARCH e a concorrência.

Por exemplo, vamos pensar só no JOIN 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. 3º implemente um método que envie essas informações, como String, ao servidor. Agora, vamos pensar só no JOIN do lado do Servidor. 1º implemente um método que recebe a String do socket e obtenha as informações. 2º crie uma estrutura que armazene essas informações (por exemplo, uma Hashtable? ou um List?). 3º envie a resposta para o Peer. Agora, voltemos ao Peer. 4º Crie um método para receber JOIN_OK. 5º. crie um ciclo while para que realize novamente alguma operação (veja que agora poderá realizar outras, como SEARCH ou LEAVE). Recomendo fazer o LEAVE após o JOIN, o que lhe permitirá realizar testes.

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").

4. Posso usar alguma biblioteca (como a gson da questão 3) para enviar o arquivo de 1GB?

Como mencionado no documento, não pode usar bibliotecas que implementem as funcionalidades pedidas. Por outro lado, existem centenas de páginas Web que mostram como transmitir um arquivo gigante.

5. Após o envio do arquivo de um *Peer A* a um *Peer B*, o *Peer A* envia mais alguma mensagem?

Não. O peer A envia só o arquivo e o peer B o recebe. Não precisa enviar nenhuma outra mensagem após esse envio. Lembrando que o DOWNLOAD_NEGADO corresponde ao caso do Peer A não enviar o arquivo ao Peer B.

6. Tenho a estrutura no servidor 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