Sistemas Distribuídos
Projeto de Programação – Napster com Sockets em Python

Revisão 1: 29/05/2023.

IMPORTANTE turmas DA2 e NA2

Para turmas da prática com o professor Rodrigo Tinini:

O projeto deverá ser desenvolvido em Python e as dúvidas do projeto deverão ser encaminhadas unicamente a ele.

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, usando TCP como protocolo de comunicação.

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 implementar os exemplos presentes no documento postado no Sigaa.

Também é recomendado a leitura da documentação de sockets em Python https://docs.python.org/3/library/socket.html,

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 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 conterá a string JOIN_OK.

c) Requisição SEARCH: vinda de um peer que procura 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 conterá uma lista vazia ou com as informações dos peers que possuem o arquivo.

d) 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 conterá a string UPDATE_OK.

e) Inicialização do servidor: o servidor deve capturar inicialmente o IP e a porta do registry. O endereço IP a ser inserido será o 127.0.0.1 se estiver realizando o projeto na mesma máquina. Assuma esse IP quando o Peer quiser comunicar-se com o servidor. A porta default (que permitirá aos peers conectar-se com o registry) será a 1099. Sobre a captura, ela se dará pelo teclado.

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

a) Recebe e responde **[opcional usar threads]** requisições do servidor e de outros peers.
b) Envia uma requisição de JOIN ao servidor. Deve esperar o JOIN_OK.
c) Envia uma requisição de UPDATE ao servidor. Deve esperar o UPDATE_OK.
d) Envia uma requisição de SEARCH ao servidor. Voltará uma lista como resposta (vazia ou com informações).
e) Para os envios acima, considere que não há perdas, nem duplicações.
f) Envia uma requisição de DOWNLOAD a outro peer. Veja abaixo as possíveis respostas obtidas.
g) Requisição DOWNLOAD: vinda de outro peer (PeerY), pedindo por um determinado arquivo. O PeerX deve verificar se possui o arquivo e enviá-lo ao PeerY.
h) Recebimento do arquivo: o arquivo deverá ser armazenado em uma pasta específica do peer.

i) 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 se estiver realizando o projeto na mesma máquina. Ver o JOIN no item 'Menu interativo' na seção 6.

6. Mensagens (prints) apresentadas na console

No 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]".

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

    - 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].

7. Teste realizado pelo professor

O professor executará o código usando o Python 3.8.

O professor abrirá 4 consoles (no Windows, seria o CMD.EXE, também conhecido como prompt). Um deles será o servidor (com o registry) e os outros 3 os peers. A partir das consoles, o professor realizará os testes do funcionamento do sistema.

Cabe destacar que:

- Seu código não deve estar limitado a 3 peers, suportando mais do que 3.
- Inicialmente, o professor executará o servidor. A seguir, executará os peers.
- Você não precisará de 4 computadores para realizar a atividade. Basta abrir 4 consoles.

8. Código fonte

- Deverá criar somente as classes Servidor e Peer.
    - A única exceção é a criação das classes para Threads **[opcional usar 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 será executado usando o Python 3.8, em uma máquina que executa Windows 10/11.

- O uso de bibliotecas que realizam 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 arquivos .py (c) o link para um vídeo, dentro do relatório, que mostre o funcionamento da aplicação. A entrega será 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 Python.
- 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 do screencast deverá conter no máximo 5 minutos, mostrando a compilação, o funcionamento do código nos quatro

consoles e a reprodução do arquivo transferido. **Não envie o vídeo do screencast**, 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 screencast do ponto anterior, deve realizar e mostrar:

   a. Join do peer e aceitação pelo Servidor.
   b. Busca por um arquivo que não existe no sistema.
   c. Busca por um arquivo que existe no sistema.
   d. Download de um arquivo de vídeo. Mostre que pesa mais de 1 GB, mostre que na pasta não existe nenhum arquivo antes do download.
   e. Mostre a visualização do vídeo por um reprodutor de vídeo após o download.

IV. Para cada funcionalidade do servidor, uma breve explicação em "alto nível" de como foi realizado o tratamento da requisição. Na explicação, não se esqueça de mencionar as linhas de código fonte que fazem referência à funcionalidade.

V. Para cada funcionalidade do peer, uma breve explicação em "alto nível" de como foi realizado o tratamento da requisição. Na explicação, não se esqueça de mencionar as linhas de código fonte que fazem referência à funcionalidade.

VI. **[opcional usar threads]** Explicação do uso das threads, mencionando as linhas do código fonte que fazem referência ao uso de threads.

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

VIII. 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 itens que descontarão a nota.

- Código fonte não executa ou usa bibliotecas externas sem aprovação do professor (nota zero).
- Não usa sockets nas comunicações (nota zero).
- Não transfere arquivos maiores a 1GB (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, separado nas consoles, e reprodução (menos 2 pontos)
- Enviou outros arquivos do código fonte além dos citados na Seção 8 (menos 2 pontos).
- Código fonte sem comentários que referenciem as funcionalidade das seções 4 e 5 (menos 2 pontos).

12. Links recomendados

- Download do Python
  https://www.python.org/downloads/

- Informações sobre programação com sockets em Python:
  https://www.tutorialspoint.com/python/python_networking.htm
  https://www.geeksforgeeks.org/socket-programming-python/

- Informações sobre Threads, que permitem que o servidor ou peer receba e envie informações de forma simultânea:
  https://realpython.com/intro-to-python-threading/
  https://docs.python.org/3/library/threading.html

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 implementar os exemplos postados. Com esses exemplos "implementados" e “testados”, você terá um cliente e um servidor (concorrente) funcionando e transmitindo mensagens. Isso é a base para o projeto do Napster.

2. Já implementei os exemplos e eles estão funcionando. E agora?

Na computação temos a frase "dividir para conquistar" e podemos usar esse conceito aqui. Recomendo primeiro implementar o JOIN e validar seu funcionamento. Após isso, implemente as outras funcionalidades, como 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 e obtenha as informações. 2º crie uma estrutura que armazene essas informações (por exemplo, uma lista, um dicionário?). 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).

3. Posso usar alguma biblioteca 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.

4. 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.

5. 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* e semáforos, por exemplo.