Nome: João Augusto Zanardo de Lima **RA:** 11201920195

Link para o Vídeo do funcionamento: https://www.youtube.com/watch?v=O_siwng_GX0

Funcionalidades do Servidor:

Para a tratativa da requisição do *JOIN* foi criado um método remoto chamado "*join*" onde um peer se conecta ao servidor. Esse método recebe o endereço IP do *peer*, a porta e uma lista de arquivos que o mesmo possui. Se o *peer* já estiver cadastrado no servidor, uma mensagem de erro será retornada. Caso contrário, as informações do par são adicionadas ao *map peers* e uma mensagem de sucesso é retornada.

Para a tratativa da requisição *SEARCH* foi criado um método remoto chamado "*search*". Este método é usado para pesquisar *peers* que possuem um determinado arquivo. Ele recebe o endereço do peer solicitante e o nome do arquivo a ser pesquisado, em seguida, percorre todos os *peers* registrados no *map peers* e verifica se eles possuem o arquivo desejado. Os *peers* que possuem o arquivo são adicionados a uma lista e retornados como resultado da pesquisa.

Para a tratativa da requisição UPDATE foi criado um método remoto chamado "update". Este método é usado para atualizar as informações de um *peer*. Ele recebe o nome do *peer* e o nome do arquivo a ser adicionado ao respectivo *peer*. Caso esse *peer* existe e esteja presente no map *peers*, o arquivo é adicionado à sua lista de arquivos e em seguida, uma mensagem de sucesso é retornada. Caso contrário, uma mensagem de falha é retornada.

A conexão RMI ocorre na classe "main" a qual é executada assim que o programa é iniciado. Ao executar, uma instância da classe ServerImpl é criada e em seguida, um registro RMI é criado na porta 1099 e o objeto remoto (*server*) é registrado no *registry*.

Funcionalidades do Peer:

Temos o método *start* que é responsável por iniciar a execução do *peer*. Ele estabelece a conexão com o servidor e também obtém a referência para a interface *ServerInterface* e inicializa um *loop* de comandos que permite que o usuário interaja com o *peer*. Dependendo do comando digitado, as funções *join*(), *search*(), *download*() ou uma opção para sair serão executadas. Essas funções funcionam como as requisições que o *peer* solicita ao servidor para que haja toda a interação.

Após a execução da função *join()* é iniciado uma nova instância da classe *ThreadForDownload*. Essa classe interna estende a classe *Thread* e é responsável por lidar com o processo de *download* em um *peer*. Ela cria um *socket* de servidor na porta especificada pelo *peer* e aguarda a conexão de outro *peer*. Uma vez conectado, o peer envia o arquivo solicitado para o *peer* de destino em pacotes de 4096 *bytes*.

Após a inicialização e o aceite do socket do servidor, foi utilizado o *DataOutputStream* e o *DataInputStream* para a criação dos canais de fluxo de entrada e saída entre os *peers.* Para o canal de transmissão de envio do arquivo foi utilizado o *FileInputStream.*

Após a requisição do *search* ser realizada com sucesso, é salvo em uma variável estática, o nome do arquivo que foi procurado dentre os *peers* registrados. Essa variável é utilizada na requisição de *download* onde ao informar o IP e porta do *peer* que contém o arquivo solicitado, será feito a transferência de arquivo para o *peer* solicitante.

Referências:

https://stackoverflow.com/questions/5694385/getting-the-filenames-of-all-files-in-a-folder

https://www.geeksforgeeks.org/transfer-the-file-client-socket-to-server-socket-in-java/

https://stackoverflow.com/questions/67430428/problem-when-sending-messages-from-client-socket-to-server-socket

https://codedamn.com/news/java/transferring-files-from-client-to-server-socket-in-java

Java.io.FileInputStream.read() Method (tutorialspoint.com)

https://www.geeksforgeeks.org/java-fileinputstream-read-method-with-examples/

Java Examples & Tutorials of FileInputStream.read (java.io) | Tabnine