Sistemas Distribuídos

Prof. Dr. Vladimir Moreira Rocha

Aluno João Vitor Arruda de Bartolo (RA 11201812168)

Relatório do Projeto Napster

1- Link do vídeo demonstrando o funcionamento: https://youtu.be/JC5LsCtVfdc

2.1- Funcionalidades do servidor:

a) Método MAIN (linha 15): no método MAIN o servidor cria um socket UDP(linha 20) para receber requisições dos peers (JOIN, SEARCH, DOWNLOAD, LEAVE), quatro estruturas de dados do tipo Map(linhas 23-32), sendo elas: *mapaPeers* (onde guarda a porta TCP dos peers juntamente com um file representante do local onde os arquivos serão encontrados para download), *peersAlive* (onde guarda uma chave do tipo *InetSocketAddress* que representa o ip e porta do cliente que permanecem vivos no sistema, e o valor *Integer* que guarda a porta UDP ALIVE do cliente), *respAlive* (onde guarda uma chave do tipo *InetSocketAddress* que representa ip e porta dos clientes que responderam a requisição do ALIVE, e um *Integer* que representa a porta UDP ALIVE do peer) e por fim *peersPortas* (onde guarda a relação entre porta UDP ALIVE e porta TCP de cada peer para consultas). Das quatro estruturas de dados usadas, as três últimas respectivamente são usadas para o método ALIVE do servidor. Logo após de definidas é chamada a seguinte thread (linha 38) : *ThreadLoopAlive*, onde inicia o processo de requisição para os peers que ainda permanecem conectados na rede ou tiveram uma saída brusca (sem o método Leave).

Em seguida o servidor entra num laço (linha 43) onde fica aguardando receber pacotes de requisição de peers que se conectaram no sistema. Quando ocorre o recebimento do pacote o servidor roda o a thread (linha 55) *ThreadRequisicao* que responde a requisição do peer enquanto aguarda novas requisições.

b) Método JOIN (linha 184): no método JOIN o servidor define um File (linha 190) referente a string local passada pelo objeto mensagem e em seguida define um array de strings que guardará o nome dos arquivos dentro do path local. Em seguida adiciona à estrutura de dados *mapaPeers* (linha 194) a porta TCP do peer e o path referente ao local que se encontram os arquivos disponibilizados para download. Enquanto que adiciona à estrutura *peersAlive* (linha 201) Ip:porta do peer que até o momento está vivo no sistema e sua porta. Em seguida nas linhas 204-209 o servidor printa “Peer [IP]:[porta] adicionado com arquivos [nome dos arquivos]” e por fim envia de volta ao

peer (linhas 214-217) o objeto mensagem com a string “JOIN_OK” para que o peer saiba que sua inclusão foi aceita.

c) Método SEARCH (linha 133): inicia o método com o print “Peer [IP]:[porta] solicitou arquivo [nome do arquivo]” referente ao peer que solicitou a busca pelo arquivo descrito. Em seguida define um ArrayList<String> (linha 142) que receberá e guardará as portas TCP dos peers que possuem o arquivo solicitado. Entre as linhas 144-160 é criado dois laços: o primeiro que percorre a estrutura *mapaPeers* capturando seu local de acesso e guardando os arquivos existentes do path em um vetor de string, e o segundo que percorre esse array de string definido anterioramente comparando com o nome do arquivo solicitado pelo peer. Caso encontre no mínimo um peer que possui o arquivo buscado, altera o tipo string mensagem para SEARCH_OK (caso não, altera para SEARCH_NOT_OK) e envia a mensagem para o cliente juntamente com o array de peers que possuem o arquivo.

d)Método LEAVE (linha 227): remove (linha 233) o peer que fez a solicitação com base na sua porta TCP da estrutura de dados *mapaPeers*. Logo em seguida remove(linhas 237 e 238) o peer das estruturas *peersAlive* e *peersPortas.* Na linha 244 há uma condição para mandar de volta ao peer que sua requisição LEAVE foi aceita, ela foi feita pro caso do ALIVE, quando o servidor percebe que o peer morreu, ela usa também a função LEAVE para retirar suas informações de base de dados e nesse caso não é necessário mandar a resposta de volta ao peer.

e) Método DOWNLOAD (linha 60): Similarmente ao SEARCH o método DOWNLOAD considera que o peer fez a requisição de download sem ter feito o Search antes, por isso inicialmente busca pelos peers que possuem o arquivo solicitado (linhas 70-89). Em seguida faz a busca (linhas 92-110) deste arquivo dentro dos peers que possuem o mesmo através do IP e Porta definidos pelo peer na requisição de download. Caso realmente encontra, devolve o objeto mensagem com o tipo definido em “DOWNLOAD_OK” (caso não devolve com a string “DOWNLOAD_NEGADO”).

f) ThreadRequisicao (linha 260): após receber a requisição de um peer no método main, o servidor roda essa thread para responder o cliente com base em sua necessidade. Avalia qual é tipo de requisição que o cliente pede (seja JOIN, SEARCH, DOWNLOAD ou LEAVE) e encaminha para a respectiva função necessária para continuar o processo. Em particular, o servidor também receberá pelo main a resposta dos peers que estiverem vivos com a string ALIVE_OK, dessa forma entra no else if (linha 306) para que o servidor atualize seu Map de peers que permanecem vivos.

- Essa thread foi implementada para que o servidor possa responder requisições de múltiplos clientes ao mesmo tempo sem precisar travar com métodos bloqueantes.

g)ThreadLoopAlive (linha 322): fica dentro de um laço que rodará eternamente: sempre espera por 30 segundos e assim limpa a estrutura de dados *respAlive* para monta-la novamente na ThreadVefAlive que será chamada logo em seguida.

- Essa thread foi implementada para que o servidor não precise ficar preso à um laço de requisições alive sendo que precisa responder requisições de outros tipos.

h)ThreadVefAlive (linha 367): nessa thread inicialmente percorremos toda a estrutura *peersAlive* e enviamos por meio da thread *ThreadEnvioAlive* para cada peer uma mensagem do tipo "ALIVE?". O que aconterá em segundo plano no cliente: ele receberá essa requisição e irá responder ou não. Caso responda, a *ThreadRequisicao* irá fazer seu papel de adicionar o peer em sua lista de peers que deram resposta como vivos. Logo em seguida a Thread de verificar alive irá fazer comparações entre as estruturas de dados *peersAlive* e *respAlive* e irá encontrar os peers que estavam vivos inicialmente em *peersAlive* e não deram a resposta em *respAlive.*

- Essa thread foi implementada pro caso de termos um número muito alto de clientes, assim a *ThreadLoopAlive* não precisa se dar ao trabalho de ficar parada aguardando todo esse processo.

i) ThreadEnvioAlive (linha 454): nessa thread é feito apenas o envio, a "cutucada" no peer para que ele responda a mensagem conferindo por sua vida.

- Essa thread foi implementada para que a *ThreadVefAlive* não precise aguardar cada processo de envio para cada peer.

2.2- Funcionalidades do Peer:

a)Método MAIN (linha 25): Inicializa pedindo IP e Porta UDP do peer pra troca de mensagens básicas para as requisições JOIN, SEARCH, LEAVE e DOWNLOAD. Define a porta UDP menciona acima, um socket TCP para envio dos arquivos no caso de outro peer pedir o download e um socket UDP que serve apenas para a requisição ALIVE do servidor. Em seguida roda o menu(linhas 60-130) para o cliente dizer o que quer fazer. No caso do pedido de JOIN, chamará a função JOIN que conversa com o servidor e ao receber a resposta irá iniciar duas thread: ThreadLoopDown onde ficará disponível para receber requisições de download, e a ThreadClienteLoopAlive que ficará respondendo o servidor dizendo que está vivo. No caso do pedido de SEARCH, apenas monta o objeto mensagem e envia pra função SEARCH. No caso de pedido de DOWNLOAD o cliente deverá inserir as informações referente ao peer de onde quer baixar e o nome do arquivo, logo em seguida roda a ThreadReqDownload para dar andamento no processo de download. Por fim, no caso do pedido do LEAVE, o cliente chama a função LEAVE e recebe uma flag para finalizar a execução do menu.

b)Método JOIN(linha 220): envia pelo objeto mensagem a requisição JOIN ao servidor e aguarda de volta o JOIN_OK. Após o recebimento, printa "Sou peer[IP]:[porta] com arquivos [nomes dos arquivos]"

c)Método SEARCH(linha 205): envia pelo objeto mensagem a requisição SEARCH ao servidor e aguarda de volta o objeto mensagem que vai possuir o array de peers que

possuem o arquivo solicitado. Em seguida printa: “peers com arquivo solicitado:[IP:porta de cada peer da lista]”.

d) Método LEAVE(linha 196): envia pelo objeto mensagem a requisição LEAVE e aguarda de volta o LEAVE_OK.

e) Método DOWNLOAD (linha 139): inicialmente cria um socket que se conecta com o peer que possui o arquivo e envia pelo objeto mensagem o arquivo que deseja. Logo em seguida cria o arquivo no diretório local que possui e lê em tamanho de 4K bytes os envios vindo do peer provedor do arquivo. Após o envio é printado: “Arquivo [nome do arquivo] baixado com sucesso na pasta [nome da pasta]”

f) Método devolveArquivos (linha 258): recebe um vetor de string e devolve a transformação para um arraylist de string.

g) ThreadLoopDownload (linha 390): cria um laço e uma porta TCP para receber a requisição de download de algum arquivo que possui. Após isso, inicia a *ThreadDownload.*

- Essa thread foi feita para que o peer possa responder a requisição de download de outros clientes enquanto ainda manda outras requisições ao servidor ou outros peers pelo menu principal.

h)ThreadDownload (linha 290): como o peer pode fazer a requisição de download antes do Search, antes ele aguarda mensagens vindo do servidor dizendo se o download pode ser feito, caso não possa, ele atualiza a porta de pedido dentro dos peers que possuem o arquivo enviado pelo servidor. Caso possa, ele continua o código e lê o envio de 4k em 4k bytes para enviar ao outro peer.

- Essa thread foi criada para poder responder múltiplas requisições de download ao mesmo tempo sem ter que esperar finalizar um download.

i)ThreadReqDownload (linha 424): Apenas chama a funcaoDownload.

- Essa thread foi criada para que o cliente possa pedir um download sem ter que ficar aguardando a finalização do mesmo, podendo assim continuar mandando outras requisições ao servidor e a outros peers.

j)ThreadClienteLoopAlive (linha 445): recebe uma mensagem do servidor perguntando se está vivo e envia de volta caso esteja.

- Essa thread foi feita para poder ficar respondendo requisições alive enquanto o peer continua a solicitar novas requisições para o servidor.

3- Para o envio de arquivos grandes foi divido o arquivo solicitado em pacotes de no máximo 4K bytes e foi lido em pacotes de no máximo 4K bytes por parte do peer que solicitou o arquivo.

4- Referências:

https://www.daniweb.com/programming/software-development/threads/383707/how-to-send-file-via-socket