Sistemas Distribuídos - Q2/2021

Exercício Programático

27 de julho de 2021

Luan Lima - RA 11047716

1 Servidor

1.1 JOIN

A requisição JOIN é identificada por meio do payload JOIN da classe Mensagem (linha 116). A lista dos peers com seus respectivos arquivos é gerenciada por meio de um ConcurrentHashMap global chamado mapList (linha 21). O estado de cada peer (conectado na rede ou não) é controlado por outro ConcurrentHashMap global chamado mapState (linha 22). Antes de adicionar o novo Peer no HashMap, é verificado se o Peer caiu sem avisar e retornou antes do ALIVE (linha 119), pois logo após enviar um ALIVE o Servidor atualiza o estado do Peer para false (linha 278), significando que este não está mais na rede. Em seguida é criado o objeto da classe Mensagem contendo a resposta JOIN_OK (linha 133) e o Datagrama UDP é enviado (linha 139).

1.2 LEAVE

Ao receber um LEAVE do Peer o servidor exclui o Peer da lista de arquivos e de estados (mapList e mapState) utilizando seu identificador (IP:porta) (linhas 153 e 154). Em seguida é criado o objeto da classe Mensagem contendo a resposta LEAVE_OK (linha 157) e o Datagrama UDP é enviado (linha 163).

1.3 SEARCH

Para realizar o SEARCH cria-se um ArrayList (resultList) que irá armazenar o id (IP:porta) de todos os Peers que contêm o arquivo solicitado (linha 172), identificado por meio do payload do objeto Mensagem (linha 106). Então a lista de arquivos (mapList) é percorrida (linha 176) e o conteúdo (value) de cada entrada é quebrado em forma de String (linhas 180 a 182) e depois transformado em ArrayList (linha 184), o nome do arquivo é buscado neste novo ArrayList do Peer (linha 187) e caso o Peer contenha o arquivo, seu id é adicionado ao resultList (linha 188), a não ser que o id seja o mesmo do Peer que solicitou, evitando que ele baixe dele mesmo. Em seguida é criado o objeto da classe Mensagem contendo a resposta do SEARCH (linha 195) e o Datagrama UDP é enviado (linha 203).

1.4 UPDATE

O UPDATE é feito acessando o mapList, removendo o peer e adicionando novamente (como se fosse um novo peer - linhas 210 e 211). Em seguida é criado o objeto da classe Mensagem contendo a resposta UPDATE_OK (linha 214) e o Datagrama UDP é enviado (linha 220).

1.5 ALIVE

A requisição ALIVE possui sua própria Thread (linha 247). É feito um loop (linha 268) enquanto o peer ainda estiver na lista de arquivos (mapList). Se for uma conexão nova (primeiro JOIN vindo do Peer), a Thread apenas dorme por 30s (linha 273). Caso contrário, ela define o estado do peer como falso, ou seja, fora da rede (linha 278), envia ALIVE ao Peer (linhas 282 e 298), dorme por 1s (309) até que o Peer responda ALIVE_OK e seu estado pela ThreadReply (linha 227), verifica

o estado do Peer novamente (linha 318) e caso seja falso (fora de rede), elimina-o dos hashMaps (linhas 322 e 323). Se o peer ainda estiver na rede, ela apenas dorme por 30s (linha 327).

2 Peer

2.1 JOIN

O JOIN é selecionado no Menu (opção 1 - linha 132) e o objeto Mensagem referente ao JOIN contém o método JOIN, a lista de arquivos do Peer como payload (carga útil), o IP e porta TCP o qual o Peer responderá as solicitações de DOWNLOAD (linha 135) inseridos pelo usuário na console. Em seguida é feita uma transformação de objeto para String JSON (linha 137) e uma Thread para tratar do envio do Datagrama UDP é iniciada (linha 143).

2.2 LEAVE

A requisição LEAVE inicia na linha 295. O objeto da classe Mensagem referente ao LEAVE contém o método LEAVE, payload nulo e o IP e porta do Peer, seus identificadores (linha 298). Em seguida é feita uma transformação de objeto para String JSON (linha 302) e uma Thread para tratar do envio do Datagrama UDP é iniciada (linha 306).

2.3 UPDATE

O UPDATE é feito automaticamente logo após o DOWNLOAD ter sido solicitado pelo usuário na console, começando na linha 274. Primeiramente seus arquivos são atualizados por meio de um método set da classe Peer (linha 278), em seguida é feita uma transformação de objeto para String JSON (linha 285) e uma Thread para tratar do envio do Datagrama UDP é iniciada (linha 289).

2.4 ALIVE_OK

O ALIVE_OK do Peer é tratado dentro da Thread ThreadUDPServer (linha 368), iniciada assim que as informações básicas do Peer são inseridas pelo usuário (linha 118). Por meio do método do objeto Mensagem é identificada a requisição vinda do Servidor (inha 427), então o objeto Mensagem é instanciado (linha 431) contendo o método ALIVE_OK, payload nulo e o IP e porta do peer. Após é feita uma transformação de objeto para String JSON (linha 434) e uma Thread para tratar do envio do Datagrama UDP é iniciada (linha 439).

2.5 SEARCH

Após o usuário selecionar SEARCH no Menu (linha 146), o nome do arquivo é solicitado (linha 153) e a entrada é armazenada na variável fileToDownload de forma global (linha 155), pois será utilizada também na solicitação de DOWNLOAD posteriormente. O objeto Mensagem do SEARCH contém o método SEARCH, o payload é o arquivo a ser baixado e o IP e porta do peer são definidos também (linha 158). Após é feita uma transformação de objeto para String JSON (linha 162) e uma Thread para tratar do envio do Datagrama UDP é iniciada (linha 166). A resposta recebida é tratada na ThreadUDPServer (linha 415), transformando o payload recebido em String em um ArrayList global (linhas 419 a 425) chamado searchList, pois durante o DOWNLOAD serão feitas solicitações sucessivas na lista de Peers que contém o arquivo até que algum aceite.

2.6 DOWNLOAD

A seleção do DOWNLOAD é verifica na linha 168. Então, é feita uma verificação se há algum arquivo para download (linha 174), a lista de arquivos para download (searchList) é percorrida (linha 178) até que algum Peer aceite enviar o arquivo. Cada entrada do searchList contém o IP e porta do Peer visto como servidor (linha 183), então o socket é criado (linha 188), a cadeia de escrita de informações no socket é iniciada (linhas 191 e 192) no socket é escrito o objeto Mensagem transformado em String (linha 202), tendo como método DOWNLOAD e como payload o arquivo requerido. Então, o Peer aguarda a resposta do Peer que contém os arquivos (linha 213). Caso a resposta seja positiva (linha 218), uma nova cadeia de escrita é instanciada para receber o arquivo (linhas 226 e 227). O tamanho do arquivo também é enviado como resposta (linha 230). Dessa

maneira, o Peer pode armazenar os bytes em buffer enquanto houver bytes a serem recebidos (linha 234). Ao finalizar o download, a cadeia de escrita e o socket são fechados (linhas 239 a 241). Se o Peer que contém o arquivo se negar a enviar (linha 246), será feita uma solicitação ao próximo da lista ou ao retornará ao início da lista, tomando cuidado para não solicitar rápido demais (linha 272).

3 Threads

3.1 Servidor

- **ThreadReply:** a ThreadReply é criada APENAS quando um novo Datagrama é recebido pelo Servidor (linha 63 e 88). Dentro da ThreadReply são respondidas as requisições do peer (linha 112) e iniciada outra thread (linha 148), a de tratamento do ALIVE, sendo o JOIN o trigger (evento gerador único) da segunda Thread.
- **ThreadAlive:** a ThreadAlive é criada quando um peer entra na rede por meio da requisição JOIN. Cada Peer terá a sua própria thread para verificação de estado.

3.2 Peer

- **ThreadTCPServer:** tem a função de apenas ficar em loop aguardando novas conexões TCP, criando outra Thread que trata a requisição de cada client separadamente (linha 466).
- **ThreadTCPReqReply:** Responde às requisições de DOWNLOAD vindas de outro Peer. Decide se irá enviar o arquivo ou não de forma randômica (linha 501) e em caso positivo, lê o tamanho do arquivo solicitado (linha 517) e informa ao Peer antes de enviá-lo (linha 523).
- **ThreadUDPClient:** Responsável por realizar a requisição selecionada no Menu. Recebe a mensagem já formatada como String (linha 339) e o socket UDP (linha 340).
- **ThreadUDPServer:** Sua função é receber as respostas vindas do servidor (JOIN_OK, LEAVE_OK, etc) e a requisição ALIVE.

4 Arquivos gigantes

Para realizar o download de arquivos gigantes, foi feita uma adaptação do código disponível em https://gist.github.com/CarlEkerot/2693246. Basicamente, é criada uma cadeia de escrita no socket utilizando um buffer de 4096 bytes (linha 228). O que define o tamanho máximo do arquivo a ser enviado é a variável fileSize (linha 230), que para int suporta até 2.147.483.648 bytes, sendo mudada para long.

5 Referências

As referências utilizadas estão devidamente comentadas no código.