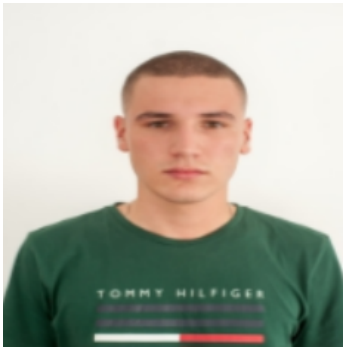
Comunicações por Computador (2023/2024) - PL6

Universidade do Minho - Campus de Gualtar, R. da Universidade, 4710-057 Braga Portugal

Trabalho Prático Nº2 - Grupo 62
Transferência rápida e fiável de múltiplos servidores em simultâneo

Duarte Leitão[a100550]



Hugo Ramos[a100644]

Diogo Araújo[a100544]


Packet lost
Lost reference ??
CC = GOATED

1 Introdução

A partilha de ficheiros é um dos serviços essenciais de uma rede. Permite que um programa cliente identifique o ficheiro e o descarregue de um servidor que o disponibiliza. O serviço tem como requisito base que a transferência seja fiável. Os dados devem ser descarregados e guardados sem erros. Os protocolos desenhados para a transferência de ficheiros procuram garantir os requisitos base, mas também assegurar um bom desempenho na transferência.

Este trabalho tem como objetivo criar um serviço de transferência de ficheiros numa rede *peer-to-peer*, com múltiplos servidores, que são também clientes do mesmo serviço. O serviço é levemente inspirado nas redes do tipo "*BitTorrent*"e tentamos recriá-lo com recurso à linguagem Java.

2 FS_Tracker

A classe FS_Tracker é usada para criar o servidor do sistema que sabe quais são os nodos que o constituem e que ficheiros tem cada um desses nodos. No tracker encontram-se em execução duas *threads*. A primeira tem como função estar constantemente à espera de comandos para executar no servidor. Encontram-se disponíveis o comando "connections", que tem como objetivo listar todos os nodos conectados ao servidor, e o comando "exit", que tem como objetivo desligar corretamente o servidor. A segunda *thread* (principal) é usada para aceitar *sockets* de nodos que se querem conectar ao servidor, e de seguida criar para cada nodo uma *thread* para gerir a conceção TCP com o servidor.

2.1 connectionsHandlerTracker

A classe connectionsHandlerTracker é responsável por gerir as conecções entre um dado nodo e o servidor. Nesta classe constam dois objetos: o *socket* do nodo e os dados do servidor.

Quando é estabelecida uma conceção TCP entre nodo e servidor, este *handler* começa por obter o endereço IP do nodo que se acabou de conectar e criar uma entrada para guardar os dados que esse endereço tem na sua pasta de entrada na base de dados do servidor. Antes de poder tratar de qualquer comando, o *handler* cria ainda os canais para o *input* e *output* de objetos e notifica o nodo conectado do seu endereço IP, visto que o mesmo não o conhece quando é inicializado. De seguida recebe os dados de todos os ficheiro que o nodo que se esta a conectar a ele tem armazenados e insere essa informação na entrada que criou anteriormente.

Enquanto o *socket* do nodo estiver aberto, o *handler* irá processar comandos vindos do nodo, separando o *input* em comando e argumentos. Caso a conceção entre o nodo e o servidor termine, os dados sobre o nodo em questão serão eliminados do servidor, visto que o mesmo já não se encontram disponíveis. O nodo poderá enviar os seguintes comandos: "get <filename>- perguntar ao servidor em que outros nodos se encontram os chunks de um certo ficheiro; "insert- inserir na entrada da base de dados do nodo novos chunks que o mesmo tenha baixado e "exit- para sinalizar o seu retiro.

2.2 TrackerData

A classe TackerData contém todos os dados relevantes para o servidor, como é o caso do endereço IP, números de portas, número de nodos conectados e um *hashmap* que contém os endereços IP de todos os nodos conectados (chave) e como valor tem outro *hashmap* que guarda na chave o nome de um ficheiro e no valor um FileInfo (informação sobre o ficheiro).

Esta classe assume um papel importante na procura e partilha de ficheiros. Sempre que um ficheiro é requisitado por um dos nodos, será devolvida uma lista com os endereços IP de todos os nodos que contém o ficheiro em questão. Sempre que é inserido um novo nodo ou um ficheiro num dado nodo, é ainda feita a atualização da base de dados do servidor. Para garantir a consistências dos dados em operações concorrentes, foram usados *locks* nos métodos usados para estes fins.

Esta classe faz o controlo de eventuais corridas que possam acontecer devido a leitura e mudança de dados por parte de varias threads utilizando locks e unlocks.

3 FS_Node

A classe FS_Node é usada para representar todos os nodos do sistema de partilha de ficheiros. Para que o nodo seja inicializado com sucesso, é necessário que o mesmo receba como argumento a diretoria onde se encontram os ficheiros partilhados.

Após a inicialização do nodo será estabelecida uma conexão entre o nodo e o servidor através de um *socket* TCP. São de seguida criados os *pipes* de escrita e leitura para que o nodo possa ser notificado do seu endereço IP e para que o mesmo possa informar o servidor quais são os ficheiros que tem disponíveis para partilha.

No nodo encontramos duas *threads* em execução. A primeira, que é a principal, tem como objetivo estar constantemente à espera de comandos e os seus argumentos para executar. Estão disponíveis o comando "get", que é usado para pedir o ficheiro introduzido como argumento, o comando "printFiles", usado para consultar os ficheiros disponíveis no nodo, e o comando "exit", usado para fechar o nodo corretamente. A segunda *thread* assume um papel importante na partilha de ficheiros. Esta tem como objetivo estar constantemente à espera de pedidos UDP enviados por outros nodos e processar os datagramas recebidos.

3.1 Fazer pedidos de ficheiros

A realização de um pedido de um ficheiro começa pela utilização do comando "get"seguido pelo nome do ficheiro pretendido. Este pedido é primeiro enviado para o servidor através da conecção TCP estabelecida no início, que irá verificar a existência do ficheiro pretendido na rede. O mesmo irá depois enviar os nodos que possuem *chuncks* do ficheiro. Caso nenhum nodo possua *chuncks* do ficheiro ou o nodo já possua o ficheiro localmente, esta operação chega ao fim. Caso o número seja maior que zero, o nodo irá ler a lista de nodos que contém *chunks* do ficheiro pretendido.

Através do método **getFileLocations**, será criada uma lista de objetos da classe **FileInfo**, que será preenchida com informações sobre a localização do ficheiro nos nodos (endereço IP do host, nome do ficheiro, tamanho e último *chunck*). Serão pedidos *chuncks* do ficheiro pretendido a vários nodos contidos na lista construida anteriormente. Serão priorizados nodos com com menor tráfego para realizar pedidos de *chunks* de forma a evitar sobrecarga de nodos ou congestionamentos. Quando um nodo é selecionado para fornecer um dado *chunck*, este é colocado num *array*. Neste *array*, os nodos que têm um índice maior, não são solicitados à mais tempo, logo o algoritmo de seleção de nodos vai preferir nodos com maiores índices no array para realizar pedidos. É relevante mencionar que são realizados vários pedidos de *chunks* em simultâneo através de múltiplas *threads*. Estas threads fazem parte de uma *threadpool* que conforme são necessários chunks a função que pede os mesmos vai submeter uma *task* as *threads* que se encontram disponíveis nesta *threadpool*.

Para pedir *chunks* ao nodo que possui o ficheiro desejado, o nodo irá recorrer ao método **createRequestDatagram** para criar um datagrama que contém toda a informação relativa ao ficheiro.

3.2 Request

A classe Request recebe informações relevantes, como detalhes do nodo atual, nome do arquivo, tamanho do arquivo, posição inicial e final, porta do destinatário e um arquivo a transferir, para criar um pedido que irá ser enviado para os nodos que contém um ficheiro desejado por outro. Para enviar este pedido, é criado um novo *socket* UDP numa porta disponível no nodo que realiza o pedido. Depois de enviado o pedido, espera-se um *acknowledgement* do outro nodo. Caso passe um tempo de espera pré-definido, o pedido será reenviado. Caso contrário, o nodo avança para a receção de dados.

3.3 Receber pedidos de ficheiros

Como referido anteriormente, cada nodo contém uma *thread* responsável por processar pedidos vindos de outros nodos. Após ser recebido o primeiro pacote, será criado um novo *socket* UDP. É

alocada dinamicamente no nodo que processa o pedido uma porta para a comunicação. É ainda aberta uma nova *thread* para lidar com a comunicação entre os dois nodos.

O nodo que recebe o pedido começa a processar os dados contidos no primeiro datagrama recebido, com recurso ao método **processRequests**. Este datagrama contém informações importantes, como o nome do arquivo pedido, a posição de início e fim desejadas do arquivo e um ***checksum*** para garantir a integridade dos dados.

Após extrair as informações do datagrama, o nodo verifica a integridade dos dados usando o checksum. Este *checksum* é comparado com um *checksum* recalculado localmente dos dados recebidos para garantir que não houve alteração dos dados durante a transmissão.

Se a integridade dos dados for confirmada, o nodo local passa para o envio dos *chuncks* do arquivo solicitado para o nodo remoto. Esta transferência é feita com recurso ao *socket* UDP criado anteriormente. O nodo começa a enviar pacotes contendo partes do arquivo, conforme solicitado pelo nodo remoto.

Durante o envio dos pacotes com partes do arquivo, o nodo espera por ***acknowledgements*** do nodo remoto para cada pacote enviado. Estas confirmações são essenciais para garantir que a transmissão ocorra sem perda de dados. Quando o nodo remoto recebe o *acknowledgment*, confirma a receção de um pacote e sabe que já pode enviar o seguinte.

Quando a troca de pacotes entre nodos acaba, serão trocadas as confirmações finais para garantir a conclusão da transmissão de pacotes. Será posteriormente fechado o *socket* UDP utilizado, dando assim a interação entre os nodos como terminada.

3.4 Transferência de ficheiros

Para garantir que todos os ficheiros são transmitidos de forma fiável e eficiente, a transferência dos mesmos é feita através de sockets. o nome do ficheiro é convertido para bytes e os dados são lidos a partir da startPosition até à endPosition divididos em chuncks. Para cada chunck é criada uma mensagem que contém o número de sequência, o nome do ficheiro, um sinal de EOF, os dados que constituem o ficheiro e um checksum CRC32 para garantir a integridade de todos os dados. Após o envio, aguarda-se pela receção do ack correspondeste. Se este ack nunca chegar a ser recebido ou não estiver correto, a mensagem volta a ser reenviada. É definido um limite de 10 tentativas para a retransmissão do pacote e aguarda-se durante um período de 50 milissegundos até o Tracker enviar um ack.

4 Domain Name System

O DNS permite que o FS_Tracker e o FS_Node se identifiquem com os seus nomes ao invés dos IP's. Este era um objetivo importante do trabalho mas o nosso grupo não conseguiu implementar por falta de tempo ao tentar resolver problemas com o resto do trabalho, nomeadamente na área da fragmentação. A ideia que tínhamos para implementar este domínio era criar uma nova classe denominada de Domain Name System que ia ter um HashMap em que a chave seria o IP do Node e o valor seria o nome do Node. Cada vez que um novo Node fosse iniciado e quisesse registrar-se no Tracker, ele ia enviar ao Tracker o seu nome e o seu Ip, criando este uma nova entrada no seu HashMap. Se um Node mudasse de Ip, enviava uma mensagem ao Tracker a informar qual o seu novo Ip e o Tracker atualizava a entrada. Se um Node fosse desconectado, o Tracker ia receber uma mensagem e removia a entrada correspondente do seu HashMap.

5 Testes e Resultados

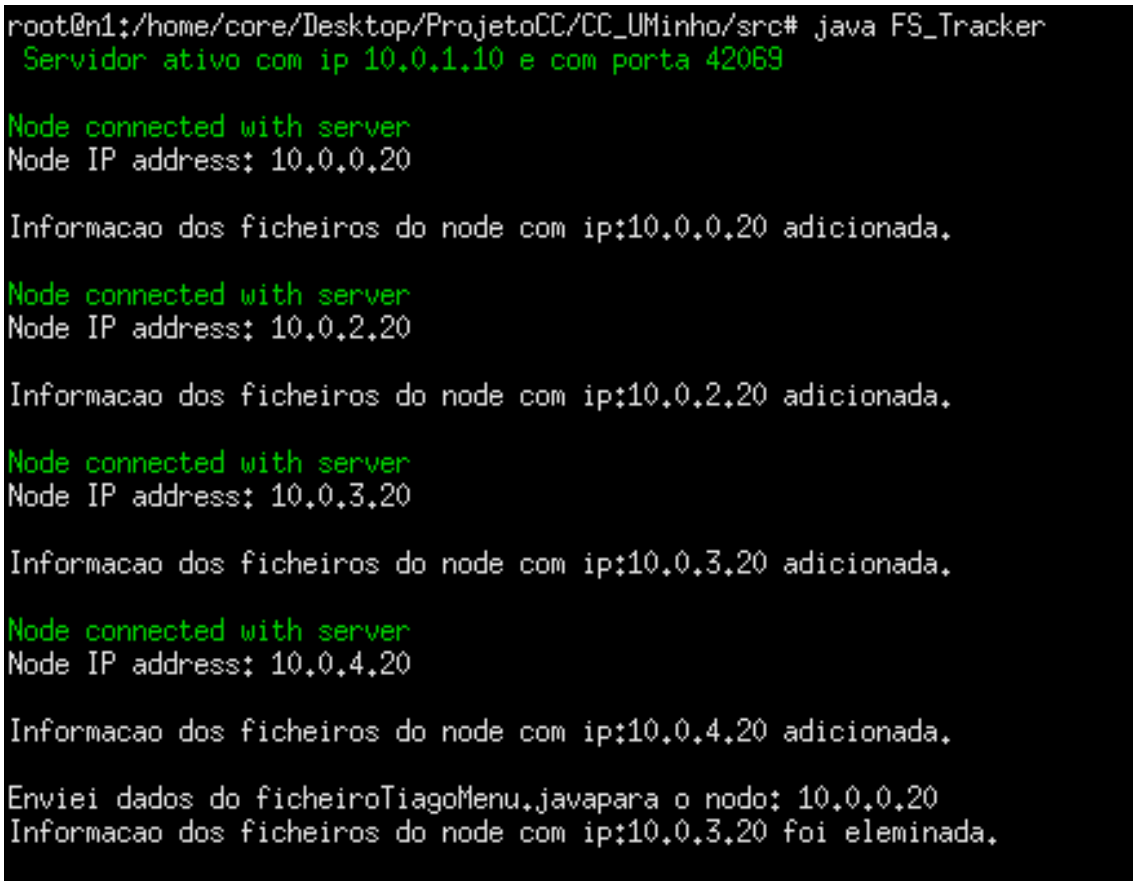


Figura 1: Conexões dos Nodes com o Tracker

Como podemos ver pela figura 1, quando o Tracker é iniciado este envia uma mensagem a dizer qual o servidor e a porta onde está ativo. Cada vez que um novo Node se conecta com o Tracker, é enviada uma mensagem a dizer qual o Ip do Node que se conectou e que a informação que o mesmo contém é adicionada com sucesso. Também quando um Node faz **get** de um ficheiro, é informado que o Node já possui agora esse ficheiro e se um Node faz **exit**, é notificado que as informações que este continha foram eliminadas.

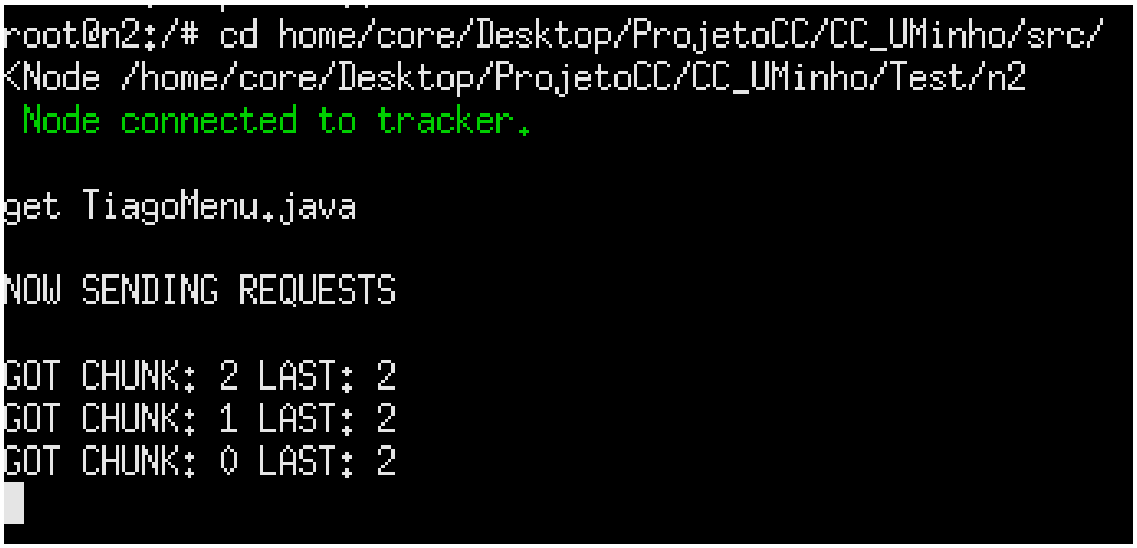


Figura 2: Execução do comando get

Relativamente à figura 2, quando um Node se conecta com sucesso ao Tracker, é enviada uma mensagem a informar do facto e quando se executa o comando **get**, o Node vai sendo informado dos chunks que já recebeu de um determinado Node que tem esse chunk do ficheiro e no Tracker é informado que o Node recebeu todos os chuncks de um determinado ficheiro.

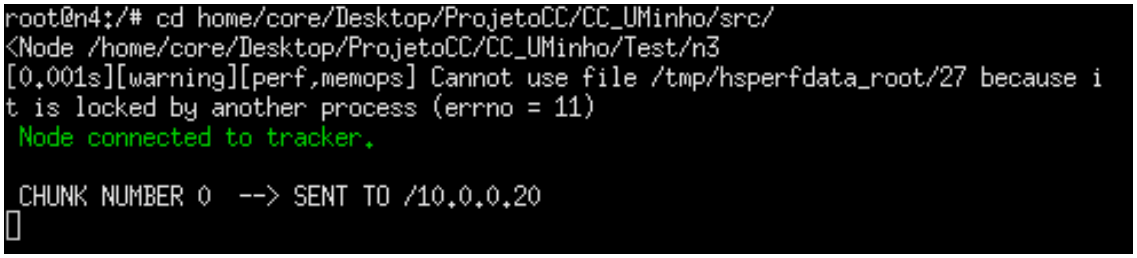


Figura 3: Envio de chuncks

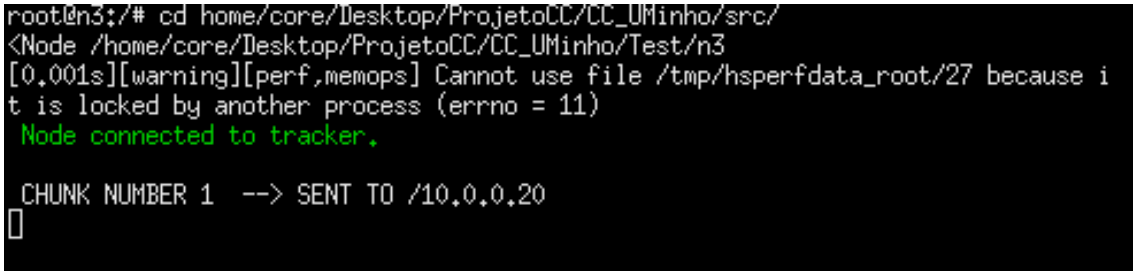


Figura 4: Envio de chuncks

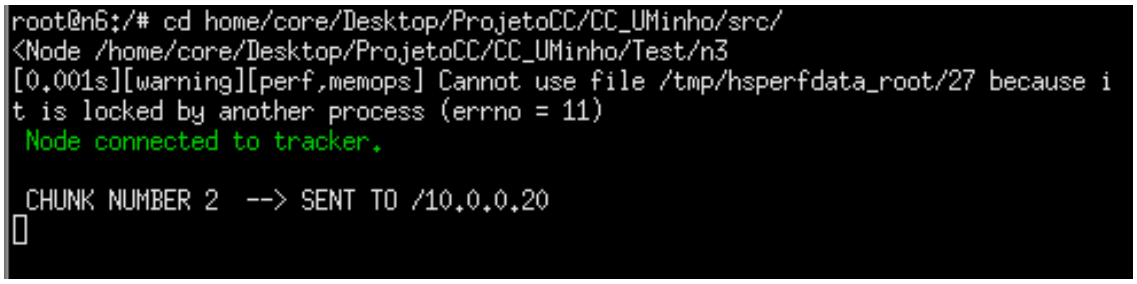


Figura 5: Envio de chuncks

Através da análise da figura 3,4 e 5, podemos verificar que os Nodes que contém chuncks do ficheiro pedido, vão enviar o/os chunck/s que cada um tem informando qual o número daquele chunck e para qual o Node que foi enviado.

Figura 6: Execução do comando get

Como observado na figura 6, sempre que um Node se desconecta através do comando **exit**, é enviada uma mensagem a informar que o mesmo foi desconectado e no Tracker é enviada uma mensagem a referir que as informações que o Node continha foram eliminadas como se vê na figura 1.

6 Conclusão

Através do desenvolvimento deste projeto, conseguimos consolidar e aplicar conceitos abordados durante as aulas teóricas. A existência de várias entidades no projeto que comunicam entre si exigiu uma boa compreensão de protocolos da camada de transporte (TCP e UDP) e de *sockets*. A partilha de ficheiros entre nodos implicou a implementação de um método de fragmentação de ficheiros.

O desenvolvimento do serviço *peer-to-peer* permitiu-nos reconhecer a transferência de ficheiros como um elemento vital das redes. Para além de garantir a transmissão de dados entre os vários elementos da rede, garante também a integridade e disponibilidade dos arquivos.

Este projeto permitiu-nos ainda evoluir o nosso conhecimento sobre a linguagem de programação Java. A utilização desta linguagem num contexto fora do habitual (contexto das redes) mostrou-nos o quão versátil esta pode ser.