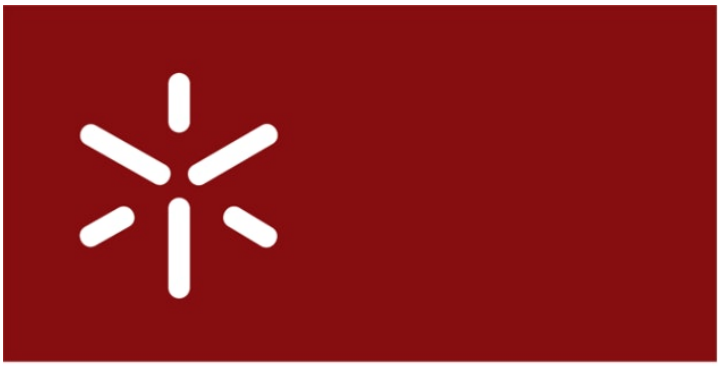


Universidade do Minho

Comunicações por Computador

Trabalho Prático 2

Grupo 43

José João Cardoso Gonçalves a93204
Bernardo Emanuel Magalhães Saraiva a93189
Daniel Torres Azevedo a93324

6/12/2021

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Comunicações por Computador, lecionada no 1º semestre do 3º ano do curso de Licenciatura em Engenharia Informática da Universidade do Minho.

O presente trabalho tem como objetivo implementar uma ferramenta de sincronização rápida, desenvolvendo um protocolo sobre UDP para realizar as transferências.

Para a implementação da aplicação foi utilizado Java como linguagem de programação e a ferramenta Core para emular uma rede para efetuar testes.

Assim, o protocolo desenvolvido cumpre, para além dos requisitos base, com alguns extras, pelo que se descrevem alguns em seguida:

- Controlo de conexão
- Sincronização entre mais que dois sistemas
- Identificação dos ficheiros a transferir entre os diversos sistemas
- Envio e receção ordenada de pacotes
- Envio de ficheiros concorrentemente
- Controlo da transmissão e verificação de sua integridade
- Sincronização de subpastas
- Resposta a pedidos HTTP
- Sistema de logs

Os tópicos referidos anteriormente serão abordados com mais detalhe nas secções seguintes.

2. Arquitetura da solução

O protocolo foi desenvolvido numa arquitetura peer-to-peer, que faz com que cada um dos nós da comunicação funcione tanto como cliente quanto como servidor, permitindo a partilha de serviços e dados sem a necessidade de recorrer a um servidor central.

Assim, o protocolo irá encarregar-se de fazer a verificação de quais os ficheiros que ainda não tem sincronizados e efetuar o pedido. Por outro lado, um dos clientes que tenha esses ficheiros fará o envio diretamente para o cliente que pediu os mesmos.

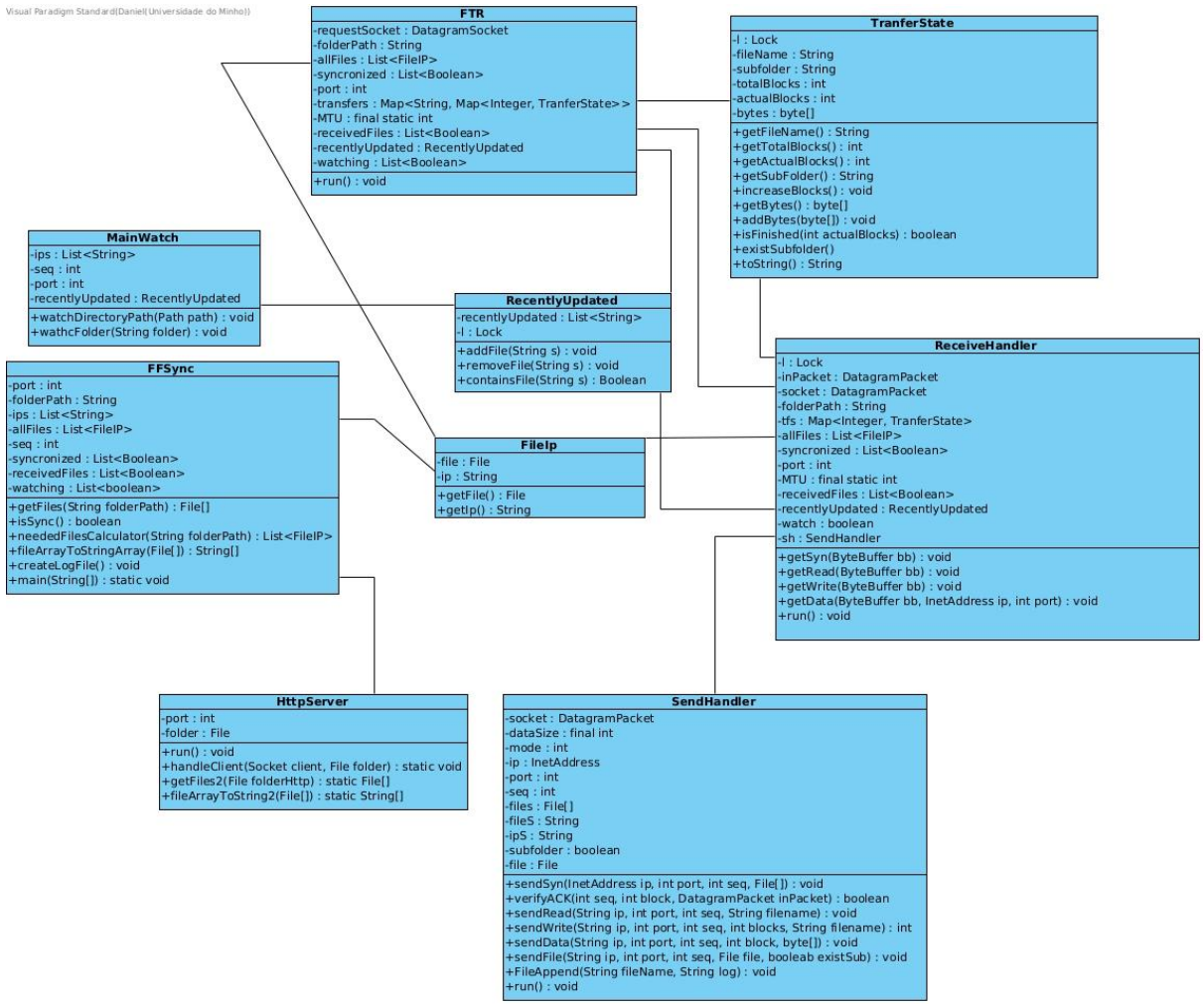


Figura 1 - Representação da arquitetura da aplicação através de diagrama de classes.

3. Especificação do protocolo

Com o objetivo de se realizar transferência de ficheiros entre dois ou mais sistemas foram definidas 2 categorias distintas de mensagens: mensagem de controlo de conexão e mensagens de dados.

No que diz respeito às mensagens de controlo de conexão, existem 4 tipos: Syn, Ack, Read e Write.

Por sua vez, as mensagens de dados denominadas Data são responsáveis pelo envio dos bytes dos diferentes ficheiros.

Para efetuar as transferências e controlar a conexão o protocolo segue uma metodologia Stop and Wait, ou seja após enviar um pacote espera receber um ACK que indica que a transferência do pacote foi bem sucedida, caso o ACK não seja recebido o pacote será reenviado.

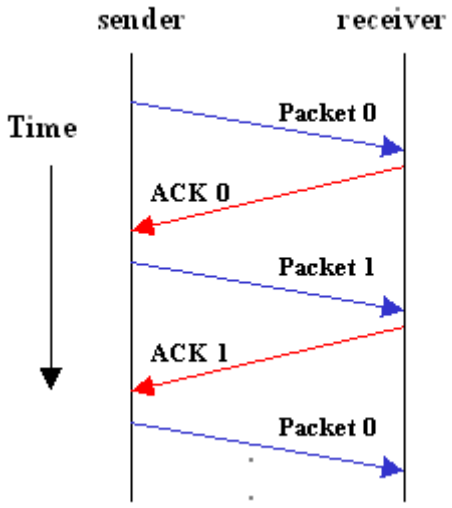


Figura 2 - Representação gráfica de uma transferência stop and wait ao longo do tempo.

3.1 Formato das mensagens

3.1.1 Syn

O pacote **Syn** é a primeira mensagem a ser enviada, esta estabelece a conexão entre os diferentes peers em sincronização. Cada peer após calcular a lista de ficheiros que possui na pasta a sincronizar envia a mesma para todos os outros utilizadores do FFSync possibilitando o cálculo dos ficheiros que necessitam, em todos os nós da conexão.

id	seq	files_length	files	mac

3.1.2 ACK

O pacote **ACK** surge para combater o problema da perda de pacotes. Com o envio desta mensagem é sinalizado o recebimento de determinado pacote através do número de sequência e/ou bloco.

id	seq	block	mac

3.1.3 Read

O pacote **Read** constitui um pedido de leitura de determinado ficheiro, através do seu nome.

id	seq	length	filename	mac

3.1.4 Write

O pacote **Write** constitui um pedido de escrita de determinado ficheiro, fornecendo informações ao receptor como o número de pacotes que terá de receber (blocks) e o nome do ficheiro a ser escrito.

id	seq	blocks	length	filename	mac

3.1.5 Data

O pacote **Data** é responsável pela transmissão dos bytes do ficheiro a transferir adequadamente divididos em pequenas parcelas.

id	seq	block	length	data	mac

4. Implementação

Durante a implementação do protocolo de foram criadas 10 classes:

- *FSync*
- *FTR*
- *ReceiveHandler*
- *SendHandler*
- *TransferState*
- *RecentlyUpdated*
- *FileIP*
- *HttpServer*
- *MainWatch*
- *Hmac*

Descrevem-se em seguida as classes referidas e o seu funcionamento em detalhe.

4.1 *FFSync*

Como classe principal, o **FFSync** tem como função sincronizar os diversos nós de comunicação que se encontram em execução, ou seja identificar quais são os ficheiros que cada peer necessita obter e os que necessita de acrescentar para que todos os nós de comunicação possuam as pastas sincronizadas.

Aqui é implementada a main da aplicação, e por isso é responsável pela criação de várias threads para possibilitar a sincronização.

4.2 *FTR*

A classe ***FTR*** é onde se concentra o protocolo proposto, sendo implementada a interface Runnable, tendo somente o método run, para que seja possível ser executada concorrentemente. Esta classe encontra-se sempre em modo escuta o que possibilita tratar vários pedidos simultaneamente. Sempre que um pacote é recebido é criada uma classe ***ReceiveHandler*** de forma a tratar o pacote adequadamente.

4.3 *ReceiveHandler*

Esta classe é suportada por um método run que possibilita receber vários ficheiros simultaneamente. Nesta classe é efetuado o tratamento dos diferentes pacotes, possuindo métodos de parsing para as diferentes mensagens, bem como métodos de tratamento para as diferentes mensagens.

4.4 *SendHandler*

No que toca ao **SendHandler**, esta é uma classe que implementa a interface Runnable possibilitando assim o envio síncrono de vários ficheiros. Esta classe possui métodos para a criação das diferentes mensagens, bem como para as enviar.

4.5 *TransferState*

A classe ***TransferState*** contém o estado de transferência de um ficheiro, este é usado para armazenar os bytes recebidos dos diferentes ficheiros. Este contém o nome do ficheiro, o nome da subpasta onde se encontra (caso este não se encontre em nenhuma subpasta este atributo será igual a null) ,o total de blocos que são necessários para a realização da transferência, o número de blocos que já foram completamente entregues e um array de bytes que corresponde aos bytes de um ficheiro.

4.6 *RecentlyUpdated*

Classe criada com o objetivo de armazenar, numa lista, o nome dos ficheiros que foram recentemente modificados, evitando ciclos causados pela classe **MainWatch** e implementado locks de modo a impedir alterações indevidas nesta mesma lista.

Quando ocorre uma alteração na pasta a sincronizar o ficheiro alterado é enviado para as os outros peers o que causa uma nova alteração na pasta, sem esta classe o ficheiro seria novamente enviado o originaria um ciclo infinito.

4.9 MainWatch

A classe MainWatch foi desenvolvida como um extra para o projeto, tendo a função de verificar as pastas com os ficheiros e, no caso de alguma ser modificada, alertar o protocolo para que este faça uma nova sincronização, de modo a que todas as pastas se mantenham igualmente atualizadas.

Para alcançar este objetivo, recorre-se à biblioteca WatchEvent de Java.

4.7 FileIP

Esta classe surge como forma de associar os diferentes ficheiros ao Ip onde se encontram.

4.8 HttpServer

Esta classe tem como objetivo colocar o servidor HTTP em funcionamento, sendo no mesmo que a apresentação dos *logs* será realizada. Este processo ocorre na camada 7 do modelo OSI.

Como variáveis de instância, tem-se a porta em que irá correr o servidor e a pasta onde estão os ficheiros (recorrendo à classe File).

Uma vez que este servidor irá correr numa thread associada a cada nó da comunicação (ou seja, para cada peer é criada uma thread com o “seu” servidor HTTP), esta classe implementa um método run para ser executado pela thread.

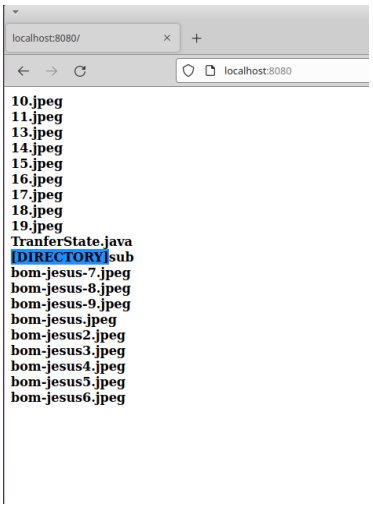
De modo resumido, esta conexão HTTP segue o seguinte plano:

1. Abrir a conexão TCP e ficar em escuta
2. Aceitar o cliente e ler a request

3. Efetuar o parsing da request
4. Mostrar a informação
5. Enviar a resposta

Este método começa por criar uma conexão e aguardar que esta seja estabelecida. Assim que a conexão sucede, entramos no método **handleClient**, que irá fazer a manipulação da conexão.

O **handleClient** começa por ler o input proveniente do socket do cliente, que contém as informações da conexão às quais será feito parsing. Após efetuar o parsing, caso o endereço seja desconhecido (neste caso apresenta um **404 not found**), é enviada a response para o output do cliente, contendo a informação dos logs. Caso tenhamos uma pasta dentro da *folder* a apresentar, esta é representada com uma cor diferente, para que o utilizador possa ter a perceção de que não é um ficheiro.



4.10 Hmac

De modo a conferir robustez, autenticação e proteger a integridade das mensagens, garantindo que estas são recebidas pelo destinatário sem quaisquer alterações acidentais ou maliciosas, foi implantada esta classe. De modo a alcançar este objetivo, no momento do envio é acrescentado, no final, 20 bytes correspondentes à parte criptográfica, esta é criada com o auxílio da biblioteca Mac.

Por sua vez, no momento da sua receção, é retirado, da mensagem, os bytes que foram criados para criptografar a mensagem, assim como a mensagem sem a parte criptografia, e é recalculado o MAC e comparado com o que foi enviado junto da mensagem. Se este for igual o conteúdo das mensagens manteve-se inalterado durante a transferência, caso contrário ocorreu algum erro e o pacote é descartado.

5.Testes

De modo a verificar o desempenho e a qualidade do programa desenvolvido, efetuaram-se alguns testes, demonstrando-se em seguida:

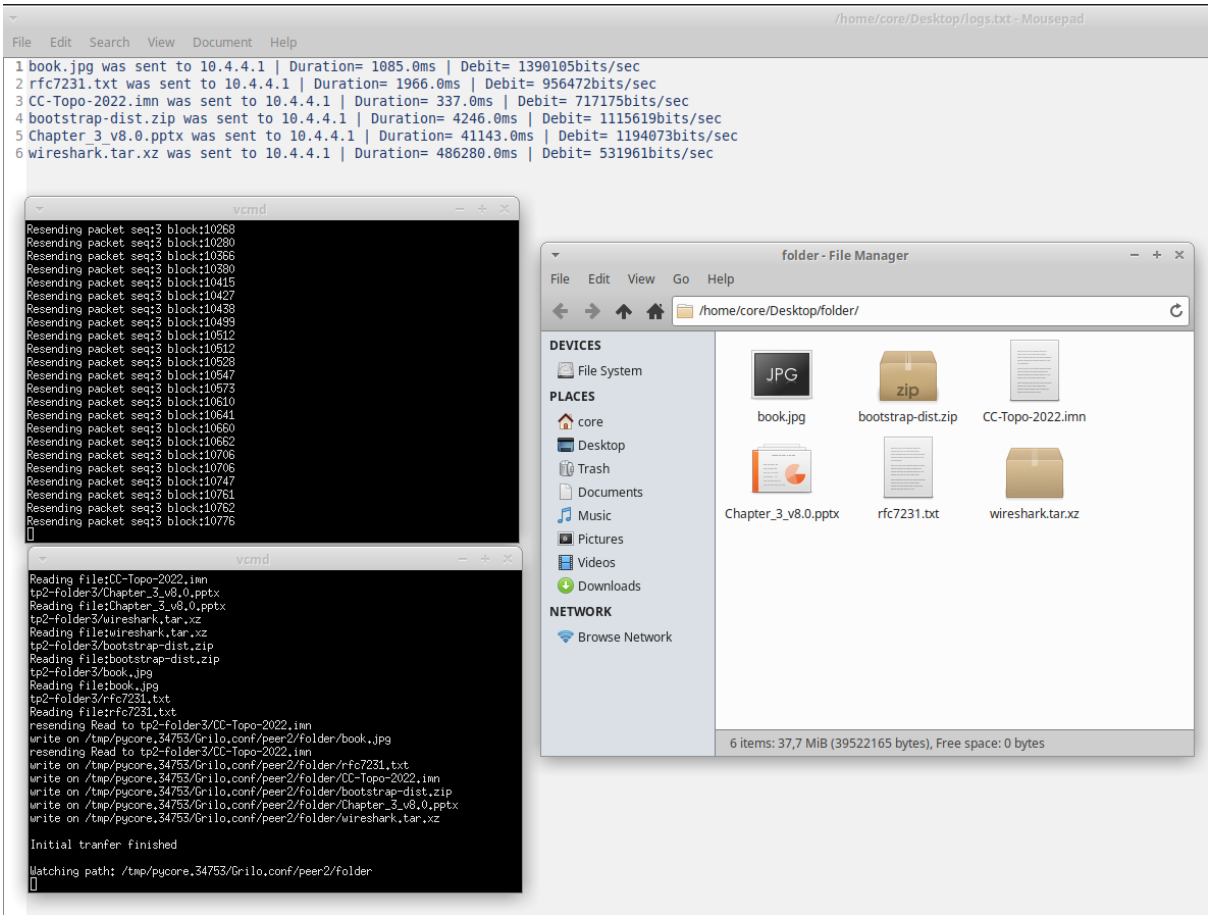


Figura 3 - Teste com apenas dois peers, com transferência de um ficheiro grande (tp2-folder3) e uma comunicação precária

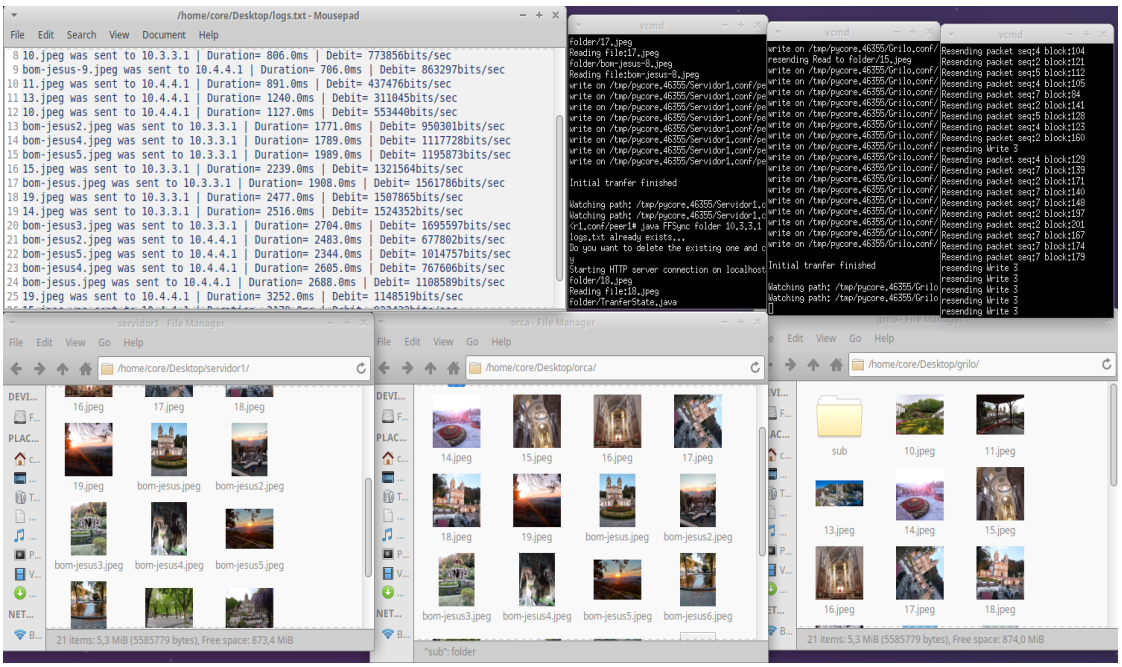


Figura 4 - Teste com 3 peers e com presença de subfolders

O servidor 1 inicialmente contém uma subpasta, que tem 2 ficheiros, e ainda 12 imagens na sua pasta principal.

Por sua vez, a Orca contém 6 imagens e ainda 1 ficheiro.

Finalmente o Grilo contém 1 ficheiro.

No final da execução do protocolo, todas as pastas se encontram sincronizadas, mesmo existindo perdas durante a transferência, devido ao facto de existirem conexões imperfeitas entre os nós.

6. Conclusões e trabalho futuro

Terminado o desenvolvimento do protocolo, o grupo pode constatar que este projeto se tornou um desafio, no sentido de compreender os conteúdos lecionados durante as aulas da unidade curricular e saber aplicá-los num contexto prático. No entanto considera-se que não só foram alcançados os objetivos mínimos, como também se elevou a fasquia para a concretização de alguns dos objetivos adicionais.

Relativamente a trabalho futuro e possíveis alterações que possam ser colocadas em prática de modo a complementar o trabalho desenvolvido, conclui-se que o código implementado ficou suficientemente "legível", podendo ser adicionados, futuramente, mais requisitos, tais como a implantação de uma interface CLI de linha de comando interativo.

Finalmente, face ao protocolo desenvolvido, o grupo encara este projeto como sendo um sucesso, uma vez que todos os requisitos são alcançados com a devida correção e desempenho do programa, avaliando assim seu desempenho de maneira positiva.