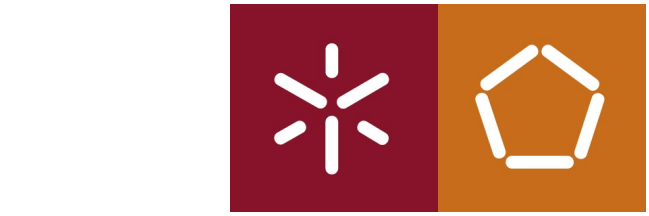


Universidade do Minho
Mestrado Integrado em Engenharia Informática
3ºano - 1º Semestre

Sistemas Distribuídos

Relatório do Trabalho Prático

Grupo 20




a83732 – Gonçalo Rodrigues Pinto
a84197 – João Pedro Araújo Parente
a84829 – José Nuno Martins da Costa
a85059 – Diogo Paulo Lopes de Vasconcelos

3 de Janeiro de 2020

Conteúdo

1 Introdução

No 1º semestre do 3º ano do Curso de Engenharia Informática da Universidade do Minho, existe uma Unidade Curricular denominada por Sistemas Distribuídos, que tem como objectivo ajudar os estudantes consolidar a formação do desenvolvimento de software concorrente, presente em todas as formas de sistemas informáticos actuais, e proporcionar a formação inicial, com uma visão abrangente, em sistemas informáticos distribuídos.

O presente trabalho pretendeu-se implementar uma plataforma para partilha de ficheiros de música sob a forma de cliente/servidor em Java utilizando *sockets* e *threads*.

2 Projecto

As plataformas de troca de ficheiros como o SoundCloud permitem que músicos partilhem as suas criações directamente com os seus fãs. Para o efeito, podem carregar ficheiros de música acompanhados de meta-informação variada (título, autor, interprete, género, ...). A meta-informação serve para que os ouvintes tomem conhecimento dos ficheiros partilhados e possam efectuar pesquisas. Tendo encontrado os ficheiros desejados, podem então descarregá-los para uso posterior.

Tendo em conta que os ficheiros a serem trocados são de dimensão considerável, normalmente com vários MB, a concretização destes sistemas tem que prestar particular atenção aos recursos consumidos com o armazenamento, manipulação e transmissão destes ficheiros. Em particular, é importante a limitação do número de operações simultâneas que podem ser efectuadas para não sobrecarregar o sistema e a manutenção de uma justiça relativa entre os diferentes utilizadores.

3 Abordagem à transferência de ficheiros

Começou-se a abordagem ao projecto proposto reflectindo-se sobre como se iria abordar a comunicação entre o servidor e o cliente, especialmente em relação à transferência de ficheiros.

Após a leitura dos requisitos, foi evidente que **cada cliente iria ter o seu próprio canal de comunicação com o servidor**, caso contrário iria-se obter um grande *bottleneck*, para além de não se ter a capacidade de identificar de quem envia a informação.

Posteriormente, foi solicitado a transferência simultânea de ficheiros, até ao momento múltiplas transferências seriam todas efectuadas no mesmo canal (apenas seriam dividas por cliente) o que levaria a uma mistura de bytes, ou seja, haveria corrupção de dados, concluiu-se que **cada transferência iria ter um próprio canal.**

Neste momento foi decidido como é que estes canais são criados/renovados, como estes **são sockets e as suas sockets têm dois lados quando criados, um que fica em espera activa até que alguém entre do outro lado e o outro lado que entra na socket, mas que se ninguém estiver presente ele ignora**, foi decidido quem fica de que lado.

Para o primeiro problema pensou-se em duas possíveis soluções , uma de **cada vez que alguém quer efectuar uma transferência criar uma socket para essa transferência e quando acabar o processo eliminar-se a socket**, a segunda solução é semelhante à primeira contudo em vez de eliminar a socket guarda-se, e se mais tarde esse cliente necessitar de efectuar uma transferência pode reutilizar essa socket, minimizando assim o custo de abrir e fechar pois iria-se efectuar esses procedimentos menos vezes. Considerando um cenário onde os ficheiros a transferir são de dimensão razoável, os benefícios da segunda solução tornam-se menos relevantes, desta forma e também dando prioridade à forma mais simples optou-se pela primeira solução referida.

Em relação ao segundo problema, pensou-se em 4 diferentes abordagens:

- A entidade a Enviar é a que espera;
- A entidade a Receber é a que espera;
- **O Cliente é sempre o que espera;**
- O Servidor é sempre o que espera;

Considerando que o servidor deve ser o mais eficiente possível, não deve depender de ninguém apenas de si mesmo, logo não se pode permitir que se coloque recursos do servidor à espera de alguma acção do cliente que não se sabe se este irá demorar 1 segundo ou 10 minutos, logo decidiu-se seguir a 3 abordagem apresentada.

Notas:

Transferência de dados- uma entidade envia dados e a outra recebe esses mesmos dados, sendo as entidades um cliente e o servidor.

Tipos de Transferência de dados

- Download - Cliente recebe dados, Servidor envia dados;
- Upload - Servidor recebe dados, Cliente envia dados;

4 Cliente

O cliente usando *sockets TCP* efectua a ligação ao servidor e procede lendo linhas de texto do standard input. Cada linha de texto lida é processada pelo método 'handle' da classe "ClientCommands", este método é responsável por analisar o que recebeu, isto é, se é um comando legível que pode ser tratado e reconhecido pelo servidor, por outro lado se não for um comando válido é apresentada uma mensagem de volta ao cliente a indicar que o comando que ele inseriu não existe e apresenta a lista de todos os comandos possíveis que pode inserir.

Do lado do cliente existe também uma classe denominada de "WriterThread", esta classe tem o *InputStream* do servidor e vai estar em ciclo contínuo a ler tudo o que o servidor responder e a escrever isso no standard output.

5 Servidor

O servidor cria uma *ServerSocket*, uma biblioteca de músicas (classe onde ficam armazenadas as informações sobre todas as músicas carregadas para servidor, de realçar que dado o facto de que esta classe é acedida por várias threads em simultâneo, para esse efeito criou-se uma classe "RWLock"idêntica à que se realizou nas aulas práticas da unidade curricular em que este trabalho está inserido, de forma a garantir que os acessos às regiões críticas sejam efectuados de forma segura (exemplo de zona crítica: Ao adicionar-se uma música, ela recebe o maior id disponível, logo este recurso tem de ser protegido, para não haver músicas com id's iguais), um catálogo de utilizadores (classe onde estão guardadas as informações de todos os utilizadores registados no servidor) e um gestor de threads (classe que gere o número máximo de threads a efectuar downloads e uploads em simultâneo, esta classe irá ser aprofundada posteriormente), após estas classes serem criadas o servidor fica parado no *accept* do *ServerSocket* aguardando que os clientes se conectem, depois de algum cliente ser conectado é lançada uma thread "ClientAuthenticator"que efectua a autenticação dos utilizadores já existentes no catálogo de utilizadores, como também regista novos utilizadores.

Depois de efectuada a autenticação a thread "ClientAuthenticator"lança uma nova thread "ServerCommands", que irá responder a todos os outros comandos disponíveis, tais como efectuar uma pesquisa por músicas, uploads e downloads. Se a determinada altura o utilizador enviar o comando *logout* a thread "ServerCommands"é terminada e a thread "ClientAuthenticator"'que estava à espera desta terminasse a sua execução, para poder autenticar outro utilizador diferente.

6 Funcionalidades Básicas

6.1 Autenticação e registo de utilizador

```
register exemplo@sd.pt password123
Please enter the path where downloads are made:
user/downloads/nova pasta/
Registration successful!!!
```

```
login exemplo@sd.pt password123
Successfully Authenticated!!!
```

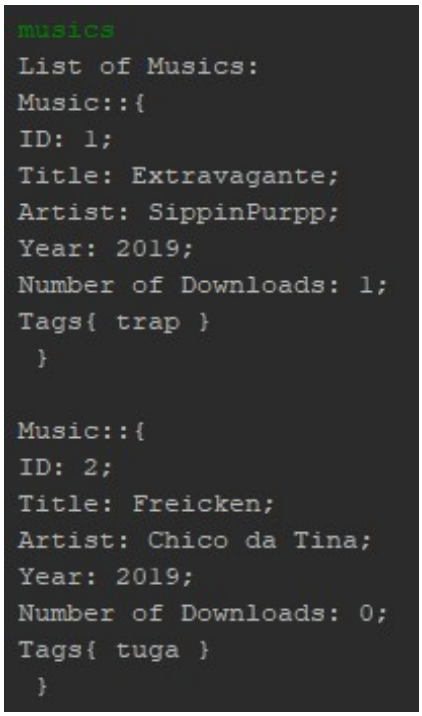
O método 'handle' da classe "ClientCommands", como foi dito previamente, verifica se o cliente inseriu um comando válido, se o for é enviado para o servidor a mesma mensagem e o servidor trata de verificar se essa conta existe no seu catálogo de utilizadores, se existir é enviada uma mensagem de volta a dizer que o utilizador foi autenticado com sucesso e depois disso o cliente fica com acesso a todos os outros comandos disponíveis. No entanto, se a conta não existir é enviada uma mensagem a dizer que o email ou password estão errados ou não existem. O comando *register* funciona de forma idêntica ao de *login* excepto que neste caso também é pedido ao cliente o *path* para onde quer que o download dos ficheiros sejam efectuados além disto o servidor também verifica se o email inserido já existe no catálogo de utilizadores caso exista informa o cliente a dizer que já existe um utilizador com esse email.

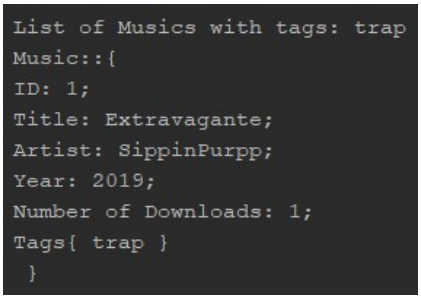
6.2 Publicar um ficheiro de música

```
upload
Please insert as follows path file|Music Name|Artist Name|Year|tag1,tag2,tag3,...
C:\Users\Gonçalo Pinto\Music\Musica\Extravagante.mp3|Extravagante|SippinPurpp|2019|trap
A NEW MUSIC has been added to the system, ID: 1, Title: Extravagante, Artist: SippinPurpp
```

Depois de receber as informações da música separadas por uma barra vertical como se pode ver na imagem acima, a classe "ClientCommands"envia para o servidor as informações da música (excepto a localização desta) e também a porta da socket de onde uma thread irá enviar os bytes do ficheiro, desta forma o servidor lança uma thread que irá receber os bytes da socket que se criou, posteriormente de receber toda a informação da música é adicionada à biblioteca de músicas e é enviada uma notificação a todos os utilizadores que estejam conectados.

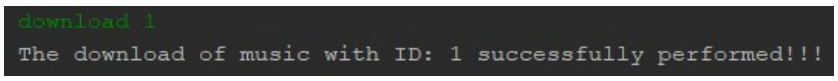
6.3 Efectuar uma procura de música





Para efectuar a procura de uma música o cliente escreve o comando *musics* seguido de um número variável de etiquetas separadas por espaços, aparecendo no ecrã as músicas que satisfazem uma qualquer das etiquetas, se o cliente não inserir nenhuma etiqueta o servidor mostra todas as músicas guardadas na sua biblioteca.

6.4 Descarregar um ficheiro de música



O "ClientCommands"recebe um pedido de *download* juntamente com o ID da música que o cliente quer descarregar, de seguida é enviado para o servidor o pedido de *download* juntamente com o ID da música e o número da porta da socket de onde a thread da parte do cliente vai receber os bytes do ficheiro enviados pela thread do servidor para a socket, depois de recebido todos os bytes do ficheiro, o cliente é então notificado a dizer que o download foi efectuado com sucesso e o contador do número de downloads da música em questão é incrementado.

7 Funcionalidades Adicionais

7.1 Limite de Descargas

Para o servidor garantir que não há mais do que *MAXDOWN* descargas a ocorrer em simultâneo, fizemos uma classe denominada de "ManagerThreads"em que o cliente e o servidor têm acesso, quando o cliente efectua um download é invocado o método 'requestReceive' desta classe, este método vai ver se o cliente que está a pedir o download já esgotou o seu número máximo de downloads em simultâneo, se for o caso e se existirem outros clientes a pretender efectuar downloads a thread fica em *await*, após passar a verificação das descargas em simultâneo daquele cliente, o método verifica se existem *MAXDOWN* descargas a ocorrerem em simultâneo, se a situação verificar-se a thread entra novamente em *await* e aguarda pela sua vez para efectuar a descarga.

7.2 Notificação de novas músicas

A classe "ManagerThreads"além de gerir as descargas em simultâneo de vários clientes guarda também como variável de instância o *PrintWriter* de escrita para cada cliente, para quando ser efectuado um novo upload, a thread que recebe os bytes do lado do servidor chama o método 'notifyClients' com o id da música, nome da música e o artista. Este método vai tratar de enviar uma mensagem formatada com o id, nome e artista da música para todos os clientes conectados ao servidor.

7.3 Tamanho dos ficheiros ilimitado

As classes "Send"e "Receive", possuem como variáveis de instância uma *socket* e uma string que corresponde a localização do ficheiro, pressupondo do princípio que os ficheiros podem ter tamanho ilimitado e assim podem não caber todos na memória, colocamos então na classe "Send"e "Receive"um limite de *MAXBYTES* de bytes em memória num determinado instante, desta forma caso o cliente peça um download, a classe "Receive"do lado cliente vai ler da socket *MAXBYTES* de cada vez até que não haja mais nada para ler e a classe "Send"do lado do servidor vai escrever *MAXBYTES* de cada vez na socket até que não haja mais bytes para escrever.

8 Conclusão

O presente relatório descreveu, de forma sucinta, a resolução do projecto proposto sob a forma de cliente/servidor em Java utilizando sockets e threads.

Consideramos que os principais objectivos foram cumpridos.

Sentimos que a realização deste projecto consolidou os nossos conhecimentos na medida em que nos permitiu entender melhor as características, virtudes, limitações e aplicabilidade do modelo de memória partilhada; permitir analisar os vários problemas decorrentes da programação com processos concorrentes; aplicar vários tipos de primitivas de controlo de concorrência em sistemas de memória partilhada; explorar as características, virtudes, limitações e aplicabilidade dos modelos e arquitecturas de sistemas distribuídos; como também proporcionou-nos resolver problemas de coordenação em sistemas distribuídos.