Sistemas Distribuídos

Troca de ficheiros

Grupo 43:

Luís Braga (A82088)
João Nunes (A82300)
Luís Martins (A82298)

Braga, Portugal
2 de Janeiro de 2020

Conteúdo

1 Introdução

No âmbito da unidade curricular de Sistemas Distribuídos foi proposta a elaboração de um trabalho prático que visa a criação de uma plataforma de troca troca de ficheiros áudio, tal como outras plataformas mais conhecidas como o *SoundCloud*.

Ora, o sistema deverá permitir que seja possível carregar ficheiros de música, sendo necessário acompanhar esse carregamento com uma panóplia de informação sobre a música tal como o seu título, o género musical, entre outros. As músicas carregadas também encontram-se passíveis de serem descarregadas, tendo de fornecer informação sobre a música de modo a descarregar, e é necessário também ter em atenção questões relacionadas com os recursos consumidos durante a execução e funcionamento da plataforma.

Como tal, e tendo em conta o contexto e o problema apresentado, foi necessário elaborar o projeto em *JAVA* utilizando *threads* de modo a garantir um acesso ordenado aos recursos e *sockets TCP* de modo a tratar da comunicação.

2 Descrição das classes implementadas

De referir que existe um elevado número de classes, uma vez que se tentou sempre que possível manter as variáveis e 'lock' correspondente encapsulado dentro do objeto (classe) com o objetivo de se fazer um locking granular.

2.1 Cliente

As classes que se seguem encontram-se no package Cliente e, portanto, são relativas ao Cliente.

2.1.1 Cliente

Ao iniciar a aplicação, irá ser criado um socket com a porta indicada para o servidor e também uma instância da classe `ClienteState` (onde serão guardadas informações sobre se o cliente está autenticado ou não e também se está à espera de uma resposta).

Em seguida, serão criadas automaticamente duas threads (para a classes `Receiver` e `MenuNavigator`) com os dois parâmetros referidos, iniciando de seguida essas mesmas.

2.1.2 Receiver

A classe `Receiver` é responsável por tratar e analisar os pedidos vindos por parte do servidor, como tal, através de um *BufferedReader* irá ser possíver ler do *Socket*, tendo apenas de seguida de fazer *parse* da mensagem e consoante a diretiva embebida da mensagem isto irá corresponder a uma ação distinta por parte do sistema. Por exemplo, se o cliente receber uma mensagem por parte do servidor do tipo *"OKDOWNLD"* é porque o cliente encontra-se livre para efetuar o *download*, como tal irá ser alocada uma *thread* para encaminhar o pedido para a classe que trata do *download* dos ficheiros de áudio.

2.1.3 MenuNavigator

Esta classe instanciada por uma thread aquando do arranque da aplicação, servirá para apresentar as ações que o utilizador poderá optar aquando da interação que este terá com a interface por linha de comandos, e terá como variáveis de instância:

- `Socket` *cs* : Socket passado por parâmetro;
- `ClienteState` cls : Estado do cliente recebido por parâmetro inicialmente;
- `int` *menu_status*: Inicializada a 0 e irá ser mudada consoante as opções escolhidas pelo utilizador;

- `PrintWriter` *out* : Para passar informações ao servidor, conforme cada ação do utilizador, sendo que cada ação é acompanhada por uma diretiva que facilmente identifica que tipo de ação o servidor irá realizar;
- `String` *extensionRegex*: Irá auxiliar na verificação do tipo de ficheiro que é enviado/recebido.

2.1.4 Uploader

O utilizador, ao pedir para realizar um *upload* de uma dada música para o *SoundCloud*, irá ter que indicar o título, interprete, ano e tags associadas à música (no mínimo, uma).

Após terem sido colocadas as informações conhecidas, irá ser invocada esta classe (através de uma thread e abrindo um socket para o respetivo *upload*), onde será enviado um pedido ao servidor e, de seguida, não permitindo que os pacotes enviados de cada vez sejam superiores a 1500 bytes.

Realizado o upload, são transmitidas as informações recebidas do servidor e é também fechado o socket que foi aberto para o *upload*.

2.1.5 Downloader

Relativamente à classe responsável pelo *download* do lado do cliente são guardadas as variáveis relativas ao *download* de uma música como por exemplo o título da música e o seu *id* bem como o *username* do utilizador responsável pelo pedido de transferência.

Como tal, é necessário um método para iniciar o pedido de *download* onde é passado o id da música juntamente com o *username* do requerido. Para além disso, existe um outro método que trata de criar o ficheiro de áudio com o título da música proveniente da música pretendida, bem como o método cujo intuito é escrever para o ficheiro, de 10000 em 10000 *bytes*, utilizando um array que contém o conteúdo da mensagem vinda do servidor, a leitura é feita através do *input* do *socket*, e é feita a escrita para o ficheiro criado anteriormente utilizando o *BufferedOutputStream*.

2.2 Servidor

De seguida, nesta secção, serão apresentadas as classes relacionadas com o Servidor.

2.2.1 Servidor

A classe `Servidor` possui o intuito de instanciar e começar três threads responsáveis por tratar dos *downloads*, *uploads* e da gestão dos pedidos de *download*, e também é responsável por aceitar a conecção inicial dos clientes.

2.2.2 Musica

Esta classe possui as variáveis relativas à informação de cada música, ou seja, possui o id da música, o título, interprete e o ano da música, uma lista com as várias *tags* associadas à música, como por exemplo *"RAP"*, a extensão do ficheiro da música, *"mp3"*, *"FLAC"*, e depois possui ainda associada a cada música um contador com o número de *downloads* dessa mesma música,

recurso esse que como é partilhado necessita de ser protegido com locks de modo a garantir um acesso exclusivo.

Como tal, para além dos *gets* e dos construtores *standard*, a classe possui um método para construir uma única *String* contendo os vários dados sobre a música, onde tal como seria de esperar, ao adicionar o número de *downloads* à string, é necessário fazer o *lock* antes dessa inserção à *String* uma vez que é necessário proteger o contador porque seria possível uma outra *thread* incrementar o contador ao mesmo tempo que está a ser construída a *String* o que iria causar que esta fosse construída com informação antiga, algo que é de evitar. Um outro método também relevante na classe é o método responsável por incrementar o número de *downloads* de uma dada música, onde tal como no caso anterior, é necessário fazer o *lock* do código responsável por incrementar a variável para garantir a exclusão mútua.

2.2.3 Musicas

A classe `Musicas` contém um *HashMap* com o *id* da música como chave e um objeto da classe música como *values*, portanto este *HashMap* guarda todas as músicas disponivéis no sistema e as suas informações.

Como tal, existe um método cujo intuito é pesquisar por músicas dado uma *tag*, como tal é necessário proteger o acesso à HashMap das músicas com *locks*, sendo então retiradas daí todos os ids das músicas com essa dada *tag* passada como argumento da função, e de seguida através da *tag* é utilizada uma função explicada anteriormente da classe *Musica* que faz o *lock* individual de cada música (ao invés da classe inteira), construíndo dessa maneira um *array* que contém os nomes das músicas com uma dada *tag*. Existe também uma função cujo objetivo é incrementar o *id* da música, onde mais uma vez como é necessário incrementar um contador, o acesso a este terá que ser protegido por um *lock*. De seguida existe também os métodos de *upload* e *download*, onde no *upload* é apenas necessário colocar na *HashMap* das músicas as informações de uma dada música (adicionada nova música ao sistema), o método *download* recebe o *id* da música e o *socket* da comunicação e apenas necessita de incrementar o contador que contém a informação do número de vezes que a música foi transferida, onde mais uma vez é protegido por *locks* e utiliza no final um método cujo principal objetivo é escrever para o *socket* o contéudo da música, para enviar para o cliente.

2.2.4 Utilizador

Classe onde serão guardadas as informações relativos a um dado utilizador do sistema, como o seu *username*, *password*, *cs* (`Socket`) e um booleano que indicará se o cliente se encontra autenticado (caso esteja, encontra-se a verdadeiro). Além disso, existir uma variável *lock* que permitirá um acesso correto.

Além das operações triviais de *login* e *logout*, existe o método *notifyMusic* que servirá para notificar o utilizador. De referir que, aquando do logout, a aréa tem que ser protegida, uma vez que, depois de o cliente se desautenticar, não terá que receber notificações do Servidor.

2.2.5 Clientes

Classe, que na sua essência, guarda um *Map* dos utilizadores, cuja a chave é o *username* e o valor é o respetivo cliente da classe *Utilizador*. Além disso, terá uma variável *Lock* para garantir o acesso partilhado corretamente.

Terá métodos típicos como *login* e *registarUtilizador*. Mas, o mais importante a referir é o método *notifyAllUsers*, em que serve para notificar os clientes, sempre que uma música é introduzida no sistema, sendo protegida a área em que se percorre o *Map* para notificar cada um dos utilizadores.

2.2.6 NotifyUsers

Classe (Thread) que é lançada para notificar todos os utilizadores da existência de uma nova música, executando um método pertencente à classe `Clientes`.

2.2.7 Pedido

Classe que serve de auxílio a um pedido de download, guardando informações relevantes do mesmo. Deverão ser guardados o id da música, o socket e o username do cliente que pediu o download.

2.2.8 PedidosDownload

Nesta classe, serão guardados, numa lista de `Pedido`, os pedidos de download que o servidor receber, sendo que o número de download's a decorrer (*numdownloads*) é limitado pelo número máximo de download's possíveis (definido pela variável *MAXDOWN*).

Além disso, existe a variável *lock*, de maneira a que garanta um acesso sincronizado corretamente nas regiões críticas, como, por exemplo, ao decrementar o número de download's que estão a ser realizados.

Esta classe possui também duas variáveis de condição fundamentais:

- *waitClients* : É utilizada para fazer para esperar caso não haja pedidos para devolver no método getPedidos e, consequentemente, processar no servidor e notifica, caso seja adicionado um novo pedido à lista (método add) e faz com que se possa avançar no método;

- *waitDownload* : Esta variável é usada no sentido em que o número de download's a correr atingiu o limite e, portanto, é necessário esperar e é notificada a thread para que possa avançar e incrementar, quando acabar um download (notifyWaitDownload).

2.2.9 DownloadDispatcher

O `DownloadDispatcher` é responsável pela gestão dos pedidos de *download* de forma justa e equilibrada, como tal implementou-se um mecanismo de *Round Robin* no que toca à gestão dos pedidos de *download*. O `DownloadDispatcher` possuirá como variáveis uma lista com os pedidos de *download*, uma instância da classe apresentada anteriormente a `PedidosDownload` e outra instância da classe Musicas (também já explicada), e possui também o mesmo número de *downloads* contíguos que o mesmo utilizador poderá fazer.

Como tal, existe um método *ronda* que é executado a cada ronda do *RR* onde é iterada a lista de pedidos, sendo retirado a cada iteração do ciclo da função o pedido individual e o utilizador do dado pedido, sendo que existe também um *Map* adicional que guarda o *username* do utilizador e o número de pedidos que já foram atentidos desse utilizador, de seguida é testado se o utilizador do pedido que se está a analisar de momento já foi atendido ou não, se já foi atendido irá incrementar o número de *downloads* que o cliente já fez e vai trocar esse valor

pelo valor ainda presente na *Map* dos pedidos já tratados, continuando com esta execucção até o cliente exceder o número máximo de pedidos contíguos. Após passar as condições anteriores, é inicializada uma nova *Thread* com o inutito de elaborar a transferência propriamente dita da música.

Ainda associados à função anterior *round*, existe um outro método cujo intuito é contar o número de utilizadores da próxima ronda, e um outro método que permite adicionar *escalabilidade* a este processo ao variar o número de *downloads* contíguos que podem ser feitos para o mesmo user consoante o número de utilizadores da próxima ronda.

2.2.10 DownloadServidor

Esta classe (Thread) é lançada no intuito de receber novos pedidos de download, ou seja, sempre que um cliente faz um novo pedido de download, esse pedido (processado corretamente) é adicionado à lista de pedidos, ou seja, à classe `PedidosDownload`.

2.2.11 DownloadMaker

O `DownloadMaker` é chamado aquando da construção da *thread* no método *round* da classe `DownloadDispatcher`, sendo que no `DownloadMaker` é chamado o método de *download* da classe `Musicas`, e uma vez feito o *download* da música, é acordada a próxima *thread* para efetuar o *download* através de um *Conditional Lock*.

2.2.12 UploadServidor

O `UploadServidor` como se trata de uma *thread* trata apenas de chamar a classe *UploadMaker* para tratar do *upload* de uma música proveniente do cliente.

2.2.13 UploadMaker

Nesta classe, serão guardadas informações relevantes de um upload de uma música, a classe Clientes que contem todos os utilizadores do sistema (para se puder notificar os clientes da música introduzida e a classe Musicas.

Como seria de esperar, é necessário tratar da mensagem com o conteúdo da música proveniente da comunicação do cliente para realizar o *upload* de uma música. Como tal, é feito o *parse* da mensagem através do *":"* que é o que delemita os campos referentes aos dados da música, e no final constrói já o ficheiro de música na pasta referente aos *uploads*.

É também necessário receber o contéudo do ficheiro através do *InputStream*, sendo este passado para um *array* de *bytes*.

Aquando da execução da thread, depois de recebida a informação do upload, é chamado o método que inicializa o upload e, de seguida, tratado esse mesmo upload. Após o upload ser realizado, é adicionada a nova música ao sistema e, por último, lança-se uma thread que notifica todos os utilizadores à cerca da existência desta nova música.

2.2.14 ClienteHandler

Esta classe trata da comunicação do servidor com o cliente, sendo que cada ação do cliente, corresponderá a uma mensagem que o seu socket enviará com um determinado significado, acabando por ser interpretada nesta classe essa informação.

3 Resumo do funcionamento do sistema

De modo a sumarizar o relacionamento entre os diversos componentes do sistema, foi elaborado o seguinte diagrama onde se apresenta os diversos elementos do sistema explicados anteriormente, bem como as diversas ações e relacionamentos que estes têm entre si.

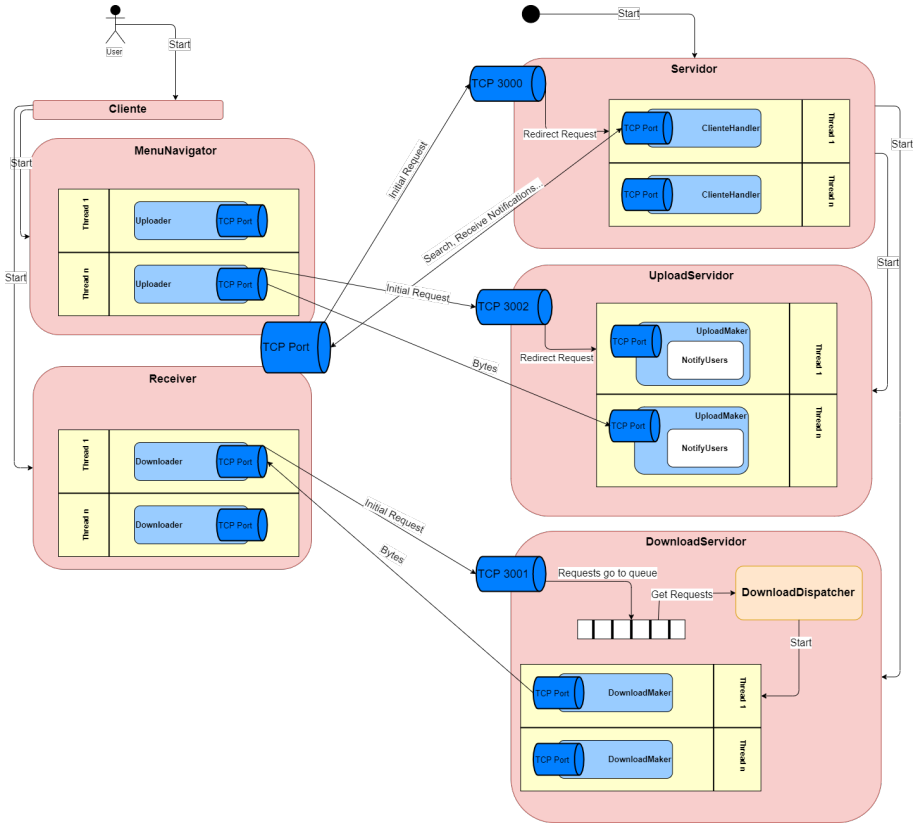

Figura 3.1: Diagrama exemplificativo do funcionamento do sistema.

4 Conclusão

Portanto, este projeto foi útil, na medida em que permitiu consolidar os conhecimentos acerca de unidade curricular de sistemas distríbuidos, tendo sido possível elaborar um projeto robusto e que permite de uma maneira justa, dinâmica e fidedigna a transferência de ficheiros áudio, tendo sempre em consideração o problema de exclusão mútua, de modo a garantir um acesso partilhado aos recursos corretamente, sendo que portanto as regiões críticas são sempre que possível protegidas por *locks*. A solução obtida também é *multithreaded* na medida em que o servidor irá possuir uma *thread* designada para tratar da gestão dos pedidos, outra para o *download* e outra para o *upload*, o que permite melhorar a eficácia da solução final.