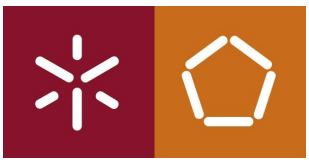

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistemas Distribuídos

SoundCloud

Etienne Costa (a76089)
Pedro Costa (a85700)
Simão Cruz (a57041)
Rui Azevedo (a80789)

5 de Janeiro de 2020

Conteúdo

1 Introdução **3**

2 Arquitectura Cliente/Servidor **4**

2.1 Cliente 4

2.2 Servidor 5

2.3 Comunicação 5

2.4 Serialização de dados 5

3 Componentes do sistema **6**

3.1 Arquitectura do cliente 7

3.1.1 SoundCloud_Stub 7

3.1.2 Notificações 9

3.2 Arquitectura do servidor 10

3.2.1 Task 11

3.2.2 Worker 11

3.2.3 Min-Heap 11

3.2.4 Scheduler 12

3.2.5 SoundCloud 12

3.2.6 Thread Pool 12

4 Conclusão **13**

Lista de Figuras

1 Arquitectura geral Cliente/Servidor 4
2 Arquitectura do Cliente 7
3 Arquitectura do Servidor 10

1 Introdução

O trabalho desenvolvido teve como objectivo aplicar os conceitos leccionados na disciplina de Sistemas Distribuídos da Universidade do Minho e teve como objectivo desenvolver uma plataforma de partilha de ficheiros de música.

Primeiro irá ser abordada, de uma maneira superficial, a arquitectura Cliente/Servidor, bem como a forma de interacção entre os dois. Por fim, é explicado, separada e detalhadamente a arquitectura de cada um dos intervenientes, explicando o comportamento de cada um bem como os seus componentes.

2 Arquitectura Cliente/Servidor

Para o desenvolvimento desta plataforma usou-se uma arquitectura Cliente/Servidor, *i.e*, uma estrutura de aplicação distribuída que permite que hajam vários acessos a um conjunto de dados partilhados entre vários intervenientes (clientes), sendo estes acessos geridos pelas máquinas que detêm esta informação (servidores).

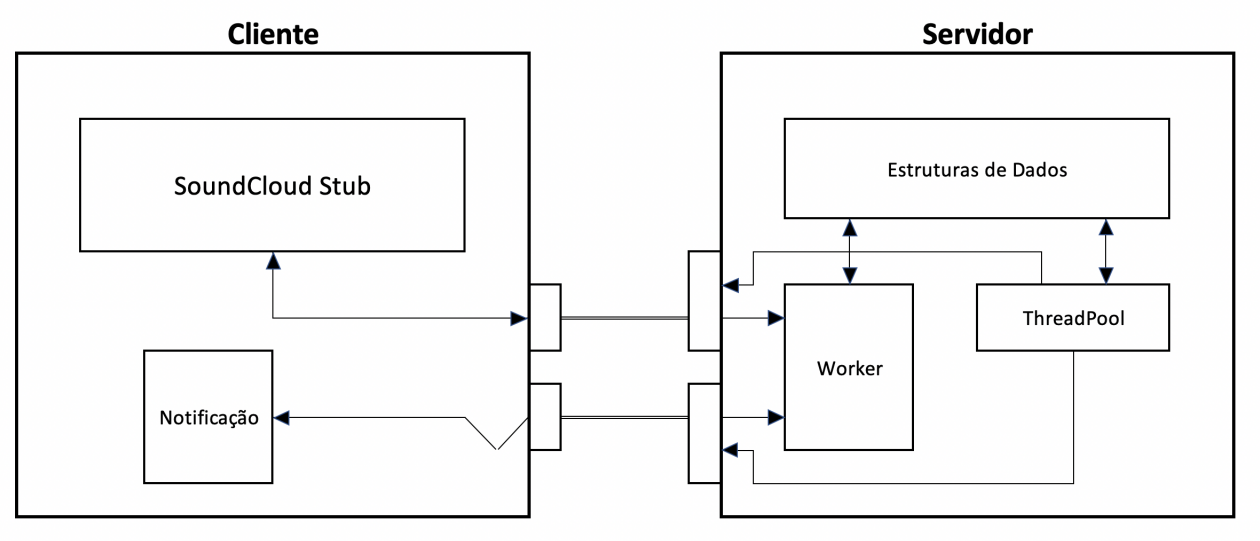

Figura 1: Arquitectura geral Cliente/Servidor

2.1 Cliente

O cliente interage com o servidor através de uma aplicação local que trata de enviar pedidos e receber respostas aos mesmos. Esta aplicação faz com que seja invisível aos utilizadores a comunicação remota com o servidor.

Os clientes, nesta plataforma de troca de ficheiros, têm as seguintes características:

- Inicia a conexão com o servidor
- Faz pedidos ao servidor
- Espera por respostas do servidor

É espectável que o grau de complexidade no desenvolvimento de um cliente com estas características seja baixo comparativamente ao do servidor, dado que o trabalho principal do cliente é tratar de enviar pedidos e, de alguma maneira, trabalhar os dados recebidos.

2.2 Servidor

O objectivo do servidor, neste sistema, é gerir os clientes conectados e responder aos seus pedidos. É imperativo que toda a arquitectura do servidor permita uma boa gestão dos diferentes recursos disponíveis, fazendo com que os recursos consumidos no armazenamento, manipulação e transmissão de ficheiros sejam os mínimos possíveis.

O servidor, neste sistema, tem as seguintes características:

- Fica sempre à escuta de pedidos de conexão de clientes.
- Aceita as conexões.
- Atende os pedidos dos clientes e responde aos clientes com os dados solicitados.
- Garante controlo de concorrência entre os diferentes clientes.

2.3 Comunicação

A comunicação entre os clientes e o servidor é feita através de *Sockets TCP*. O protocolo de rede *TCP/IP* é o mais indicado para este tipo de cenários pois garante as seguintes características:

- Orientado à conexão, ou seja, só há troca de dados após estabelecer uma conexão.
- *Full-Duplex*, *i.e*, garante que tanto o cliente e servidor enviem e recebem dados.
- Garante a entrega dos dados bem como a sua sequência, não duplicação.
- Garante que os dados são entregues sem erros implementando um mecanismo de controlo de erros.
- (...)

2.4 Serialização de dados

Uma vez que a comunicação entre ambas as partes é feita através da rede, é necessário estabelecer uma forma de serializar os dados para que o sistema seja o mais coerente possível. Para isso, estabeleceu-se que as mensagens trocadas entre o cliente e o servidor sejam baseadas em texto e orientadas à linha, há excepção da troca de ficheiros, onde é usada uma serialização de dados adequada à transmissão do conteúdo dos ficheiros. Posteriormente, irá ser definida a formatação das mensagens trocadas.

3 Componentes do sistema

Esta secção tem como objectivo detalhar as arquitecturas de cada das partes da estrutura Cliente/Servidor.

As funcionalidades de um cliente, neste sistema, são as seguintes:

- Autenticação e registo.
- Publicar um ficheiro fornecendo o seu conteúdo e meta-dados.
- Procurar uma música através de diferentes filtros.
- Descarregar uma música, dado o seu identificador.

Para além disso, será possível para um cliente, receber notificações sempre que algum utilizador publica uma música.

Uma vez que o servidor também tem que implementar estas funcionalidades para conseguir responder aos pedidos, foi criada uma interface que ambas as partes terão que implementar. No lado do cliente a ideia é que a implementação desta interface simule a invocação usual de métodos, abstraindo a comunicação remota com o servidor. No lado do servidor, é implementada a lógica de acesso concorrente às estruturas de dados, tanto para armazenamento como para manipulação dos dados.

De seguida, é apresentada a interface desenvolvida para a plataforma.

```
1  public interface SoundCloud {
2      void registration(String username, String password)
3          throws AlreadyRegistedException;
4      void authentication(String username, String password)
5          throws InvalidLoginException;
6      List<Music> consult(int id);
7      List<Music> consult(List<String> tags);
8      Music download(int id) throws InvalidIDException;
9      int upload(Music m);
10 }
```

Para além da interface, foi criado um *package* designado de *Model*, onde esta interface está inserida, e, que para além disso, implementa as classes *Music* e *User*, para mais fácil manipulação dos dados.

3.1 Arquitectura do cliente

O diagrama apresentado de seguida apresentada detalhadamente os componentes do lado do cliente.

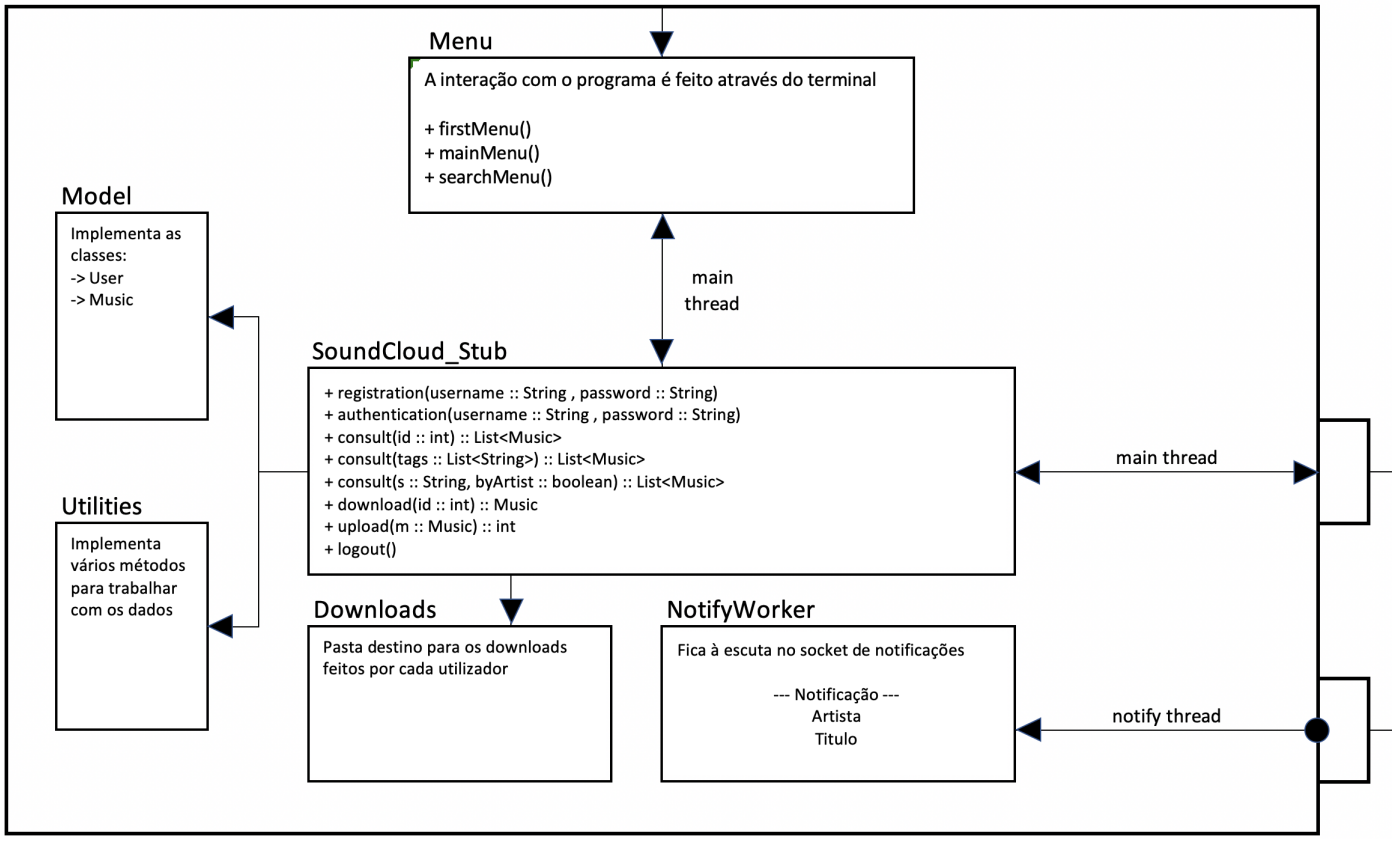

Figura 2: Arquitectura do Cliente

No momento em que um cliente é criado, criam-se dois *sockets* para a comunicação remota com o servidor bem como as suas extremidades de leitura e escrita. Um dos *sockets* tem como objectivo receber notificações de novas músicas no sistema e outro para enviar pedidos e, eventualmente, receber respostas a esses mesmos pedidos.

3.1.1 SoundCloud_Stub

Como foi dito anteriormente, o implementação da interface do *SoundCloud* no lado do cliente, também designada por *Stub*, implementa toda a lógica de envio e recepção de dados. De seguida, é apresentada a lógica da comunicação de cada um dos métodos.

Registo

Formato das mensagens: $R; username; password$

Método: void registration(String username, String password)

Comportamento: O cliente insere os seus dados no terminal e é criada uma mensagem com o pedido de registo e enviada para o servidor. O cliente espera pela resposta do servidor e caso o utilizador não exista, é criado o utilizador no servidor e o utilizador é automaticamente reencaminhado para o menu principal do programa.

Autenticação

Formato das mensagens: $A; username; password$

Método: void authentication(String username, String password)

Comportamento: O cliente insere os seus dados no terminal e é criada uma mensagem com o pedido de autenticação e é enviada para o servidor. O cliente espera pela resposta do servidor e caso o utilizador exista no sistema, é automaticamente reencaminhado para o menu principal do programa.

Consulta

Formato das mensagens: $C; type; [arg_0; ...; arg_n]$

Método: List<Music> consult(String type, List<Args>)

Comportamento: Existem quatro tipos de filtros para as procuras. O utilizador poderá escolher fazer procura de músicas por identificador, artista, título ou lista de etiquetas. Uma vez escolhida a filtragem de pesquisa, é criada uma mensagem com o tipo de procura e a lista de argumentos da mesma e enviada para o servidor. O cliente irá receber uma música de cada vez e vai guardando as músicas recebidas enquanto a mensagem *"Finished"* não é lida. Quando o cliente receber as músicas na sua totalidade devolve a lista colectada.

Download

Formato das mensagens: $D; id$

Método: Music download(int id)

Comportamento: É enviado para o servidor o pedido de *download* de uma música com um determinado ID. Neste momento o cliente cria uma nova extremidade de leitura do tipo *DataInputStream* para começar a receber os dados da música. A primeira mensagem

que recebe contém os diferentes meta-dados da música pretendida. A segunda mensagem é relativa ao número de *bytes* do ficheiro pretendido. Desta maneira, o cliente passa a saber quanto tempo tem que ficar preso no ciclo de leitura do *socket*. De seguida, começa a receber os diferentes fragmentos da música, em que cada segmento tem MAXSIZE *bytes* com o conteúdo. O cliente enquanto vai recebendo as diferentes partes da música vai escrevendo esses fragmentos directamente em disco, numa pasta com o seu nome de usuário, garantindo, deste modo, a capacidade de fazer descargas de ficheiros com grandes dimensões. Quando a transferência acaba, é fechada a extremidade de leitura criada para a transferência.

Upload

Formato das mensagens: $U; size; [m_0; ...; m_n]$

Método: void upload(Music m)

Comportamento: O processo para realizar um *upload* é praticamente o inverso do *download*. Na mensagem com o pedido de *upload* é enviado o tamanho da música em bytes bem como os diferentes meta-dados da música. De seguida é criado uma extremidade de escrita do tipo *DataOutputStream* para o envio da música. O cliente vai lendo MAXSIZE *bytes* diretamente do ficheiro escolhido e vai enviando os fragmentos da música. Quando a música é enviada na sua totalidade, fecha a extremidade de escrita aberta para a transferência do ficheiro.

Logout

Formato das mensagens: L

Método: void logout()

Comportamento: O método *logout* simplesmente envia uma mensagem para o servidor a indicar que o cliente pretende-se desligar do sistema.

3.1.2 Notificações

Como foi referido anteriormente, quando um cliente é iniciado cria automaticamente dois *sockets*, um para pedidos e respostas e outra para receber notificações. Para além disso, é criada uma *thread* para o *socket* das notificações. Desta maneira, é garantido que as notificações são apresentadas mesmo durante processos de consulta, *download* e *uploads*. Quando o *socket* das notificações é criado envia uma mensagem para o servidor com o seguinte formato: $N; username$. Com isto, o servidor sabe que quando um cliente fechar, tem que remover o *socket* de notificações correspondente a esse mesmo cliente. Após o envio da mensagem, a

thread fica à escuta desse *socket* à espera de receber notificações, enquanto não receber uma mensagem para fechar o *socket*. As notificações recebidas têm informação sobre o artista e título da música.

3.2 Arquitectura do servidor

A imagem abaixo apresentada ilustra a arquitetura do servidor, bem como os seus diferentes componentes.

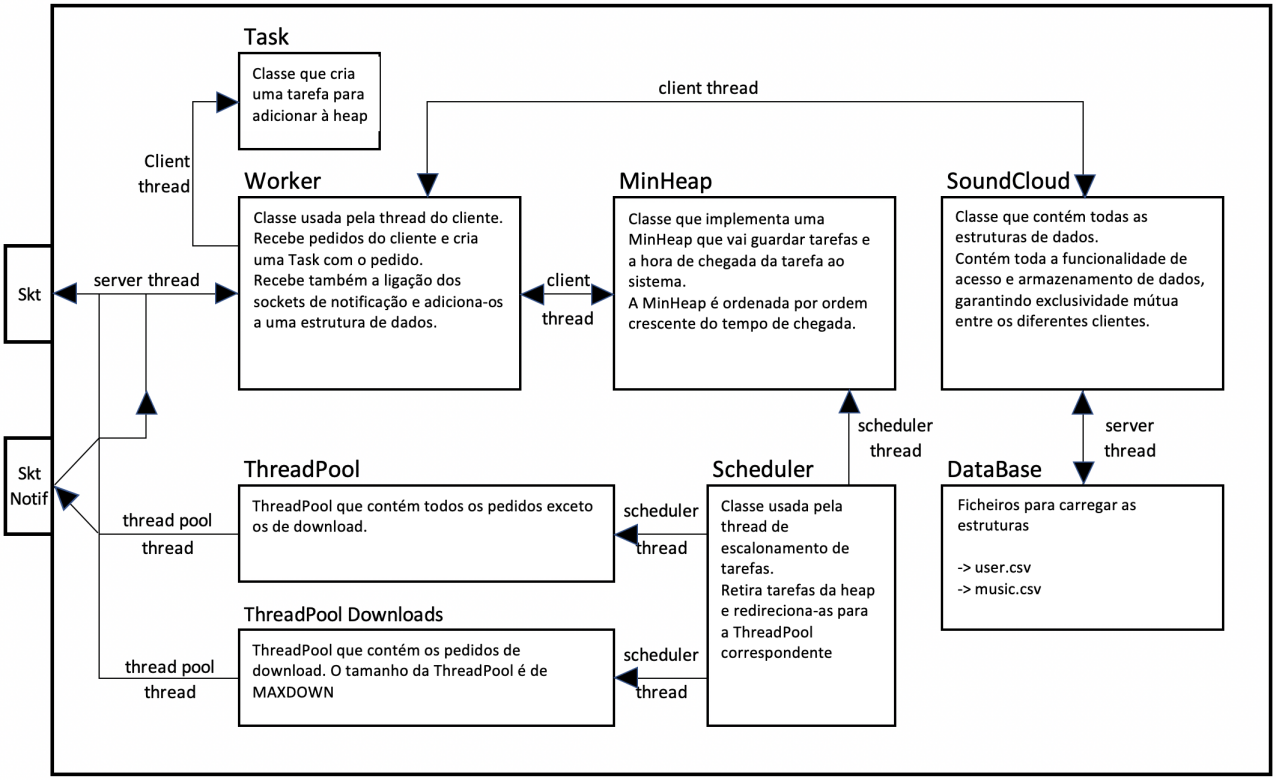

Figura 3: Arquitectura do Servidor

No momento em que o servidor é criado são inicializadas as estruturas de dados definidas para o modulação do sistema. Para além disso, é criado o *socket* que irá aceitar conexões de clientes e criar uma *thread* para cada cliente conectado. De seguida, é apresentado o fluxo de processamento de pedidos do cliente, bem como a política de escalonamento dos mesmos.

3.2.1 Task

A classe *Task* foi definida como uma classe abstracta do tipo *Runnable* que contém duas variáveis de classe, o *SoundCloud*, que contém toda a funcionalidade de acesso e armazenamento das estruturas, e, para além disso, contém a extremidade de escrita do *socket* do cliente para que seja possível enviar/receber uma resposta do cliente.

As sub-classes definidas a partir da classe *Task* dizem respeito a cada uma das tarefas possíveis no sistema, *SearchTask*, *DownloadTask*, *UploadTask*, cada uma delas implementa a lógica de procura, descarga e partilha de ficheiros, respetivamente.

Com esta estrutura, podemos guardar todo o tipo de tarefas numa única estrutura de dados e, quando uma *thread* executar uma tarefa, a mesma simplesmente tem que fazer *run* à tarefa, não sendo necessário fazer *parsing* ao pedido.

3.2.2 Worker

A classe *Worker* é usada pela *thread* reservada para cada cliente. Numa primeira fase, a *thread* fica à espera de pedidos de autenticação ou registo por parte dos clientes, ficando presa num ciclo enquanto os dados não forem correctos. Para além disso, também pode receber um pedido de um *socket* de notificações sendo que, se tal acontecer, adiciona a uma estrutura de dados a extremidade de escrita e o nome de utilizador do cliente para que, quando este se pretende desconectar, retirar o cliente respectivo da estrutura.

Após esta primeira fase, a *thread* fica à escuta do *socket* do cliente a receber pedidos. Sempre que um pedido chega, é feito *parsing* ao pedido e criada a respectiva tarefa e, associada à mesma, a hora de chegada da tarefa ao sistema. Uma vez a tarefa criada, é adcionada à *Heap* de pedidos para, posteriormente, a *thread* de escalonamento de pedidos adicionar a tarefa à *Thread Pool* para ser executada.

3.2.3 Min-Heap

Esta classe foi desenvolvida para garantir uma política de escalonamento de tarefas justa para os clientes.

Sempre que um cliente cria uma tarefa, a mesma é colocada nesta estrutura de dados juntamente com a hora de chegada da mesma. A *heap* é então organizada por ordem crescente da hora de chegada do pedido ao sistema. Desta maneira, conseguesse garantir que, mesmo que uma *thread* de um cliente não consiga obter de imediato o *lock* da estrutura, quando esta conseguir, o seu pedido é priorizado em relação a pedidos posteriores.

3.2.4 Scheduler

O escalonador de tarefas tem como objectivo retirar tarefas da *Heap* e colocá-las na *thread pool* correspondente. O escalonador retira um conjunto fixo de tarefas e liberta a estrutura para que os clientes consigam adicionar tarefas.

3.2.5 SoundCloud

Classe que implementa toda a lógica de acesso e armazenamento de dados nas respectivas estruturas. Para além das funcionalidades de procura, descarga e partilha de ficheiros também contém a funcionalidade de enviar notificações para todos os clientes activos no sistema com as informações do artista e título da música sempre que acontece um *upload*.

3.2.6 Thread Pool

Para que não haja sub-carga no sistema no processo de criação de *threads* foram criadas duas *thread pools* que mantêm um conjunto de *threads* abertas prontas para executar tarefas. Uma *thread pool* é relativa aos pedidos de *download*, uma vez que é necessário limitar este número. Desta maneira, limitando o tamanho da *thread pool* em *MAXDOWN*, é garantido que apenas existem, no máximo, este número das mesmas. A outra *thread pool* contém o resto dos pedidos do cliente.

Todos as *threads* em ambas as *thread pools* respondem aos clientes que fizeram os pedidos.

4 Conclusão

O desenvolvimento de cada uma das partes da arquitectura Cliente/Servidor teve um grau de complexidade bem distinto. Podemos afirmar, com este trabalho, que o desenvolvimento do cliente do sistema foi muito mais rápido e menos complexo equiparando ao servidor. Tal facto era espectável dado que o cliente simplesmente envia pedidos e espera por respostas, enquanto o servidor, para além de armazenar e aceder a dados, tem que garantir controlo de concorrência entre os diferentes clientes, e, para além disso, garantir que os pedidos submetidos sejam atendidos de uma maneira justa entre clientes.

Com este trabalho, pode-se concluir que o desenvolvimento de aplicações distribuídas é um trabalho bastante complexo, principalmente para aplicações que garantem uma grande escalabilidade, pois, é importante para um sistema deste tipo, que faça uma boa gestão dos recursos disponíveis pelos diferentes intervenientes do mesmo.