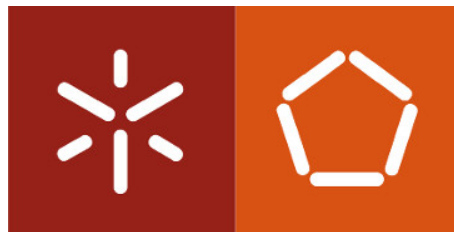


Sistemas Distribuídos

2 de Janeiro de 2019

Grupo nr. 15

a83899	André Moraes
a84577	José Pedro Silva
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1	Implementação	2
2	Organização	3
2.1	Cliente	3
2.1.1	Pasta Upload	3
2.1.2	Pasta Download	3
2.1.3	Classe Cliente	4
2.1.4	Classe Menus	4
2.2	Lib	4
2.2.1	ThreadsNot, Musica, User e Sistema	4
2.2.2	DownLock	5
2.3	Runnable	5
2.3.1	Upload e Download	5
2.3.2	Servidor Runnable	5
2.4	Servidor	6
2.4.1	Pasta files	6
2.4.2	Servidor	6

1 Implementação

Foram criadas duas classes principais executáveis para a implementação da aplicação, tendo assim um servidor e um cliente.

Inicialmente o servidor deve ser corrido antes de qualquer operação e apenas uma vez. Após isso podemos correr o cliente tantas vezes quantos clientes queremos.

Antes da apresentação de qualquer menu, é criada uma thread para cada cliente, onde é aberto um socket e enviada a instrução *notificar* para o servidor. Assim, o servidor sabe agora que este socket serve apenas para enviar notificações acerca de uma nova música. Esta thread será guardada numa lista de threads que são usadas com o propósito de notificar.

O primeiro menu apresentado ao cliente consiste numa escolha entre modo *develop*, *testing* ou *user*, em que o modo *develop* deixa o cliente escrever as intruções para o servidor a partir do *stdin*, o modo *testing* corre duas classes em duas threads difrentes onde dão upload de 5 músicas cada uma em simultâneo, o modo *user* consiste no modo normal de utilização da aplicação.

Apenas o modo de utilização *user* requer autenticação. Desse modo, sempre que o utilizador quer fazer uma operação, terá de a escolher no menu inicial. Seguidamente é apresentado um conjunto de perguntas relativas à operação escolhida, sendo pedido no final o username e a password com o intuito de autenticação antes de enviar a instrução para o servidor. Caso não exista username ou a password for errada, não é enviada a instrução.

Por cada instrução realizada é criado um socket e fechado no fim da mesma, fazendo com que cada *ServidorRunnable* execute apenas uma instrução.

Para efetuar o upload de uma música, é necessário que esta esteja presente na pasta **Cliente/upload** com o formato correto, sendo este, *Titulo - Interprete.mp3*. Sempre que é pedido um upload, o cliente cria uma thread para enviar a instrução, assim, torna possível realizar vários uploads em simultâneo. A execução da instrução download é análoga à do upload.

No servidor, quando recebe a instrução de upload ou download, existe uma requisição do lock de tráfego, criado na classe **DownLock**, onde são bloqueados todos os downloads/uploads que ultrapassem o limite máximo de tráfego em simultâneo no servidor. Para resolver esta situação tentamos implementar uma espécie de Round Robin Scheduling, no entanto, em vez de gerirmos por tempo, gerimos por número de bytes enviados, ou seja, defi-

nimos o número máximo de bytes de envio em **512 kb**. Assim, cada thread, consoante a sua ordem de chegada ao *lock*, pode enviar 512kb cada, permitindo que sejamos justos com todos os clientes e conseguimos ainda impedir a starvation.

No fim de cada upload, a thread responsável pelo mesmo, vai à lista de threads e faz com que estas notifiquem o respetivo cliente de que foi adicionada uma nova música.

2 Organização

No nosso projeto foram criadas 4 pastas distintas (**Cliente**, **Lib**, **Runnable**, **Servidor**) para melhor compreensão e execução do projeto.

2.1 Cliente

Contém as classes e pastas utilizadas pelo cliente.





 download	05/01/2020 16:06	Pasta de ficheiros	
 upload	02/01/2020 15:32	Pasta de ficheiros	
 Cliente	05/01/2020 15:46	Ficheiro JAVA	7 KB
 Menus	04/01/2020 01:44	Ficheiro JAVA	5 KB

Figura 1: Organização pasta Cliente

2.1.1 Pasta Upload

A pasta upload é onde devem estar as músicas que o utilizador quer fazer upload no formato *Título - Interpretete.mp3*.

2.1.2 Pasta Download

A pasta download contém as músicas que foram descarregadas pelo cliente no formato *Título - Interpretete.mp3*.

2.1.3 Classe Cliente

Corresponde ao programa que o cliente corre. São apresentados menus presentes na classe *Menu* e o cliente faz as escolhas a partir do *stdin*. A classe é responsável por compreender o pedido do cliente e enviar a instrução para o servidor através de um *socket TCP*. Ao iniciar o cliente é criada uma *thread* com a classe *Notifications* para ter um ouvinte constante das notificações enviadas pelo servidor. A partir daqui, apenas é aberto o socket quando o cliente quer fazer alguma operação, sendo requisitada primeiro uma autenticação com *username* e *password*.

2.1.4 Classe Menus

Contém os menus necessários para o programa, são chamados depois a partir da classe Cliente.

2.2 Lib

Contém as classes necessárias para organização da aplicação.






 DownLock	02/01/2020 17:56	Ficheiro JAVA	2 KB
 Musica	03/01/2020 19:18	Ficheiro JAVA	2 KB
 Sistema	05/01/2020 01:18	Ficheiro JAVA	3 KB
 ThreadsNot	05/01/2020 02:06	Ficheiro JAVA	1 KB
 User	31/12/2019 12:47	Ficheiro JAVA	1 KB

Figura 2: Organização pasta Lib

2.2.1 ThreadsNot, Musica, User e Sistema

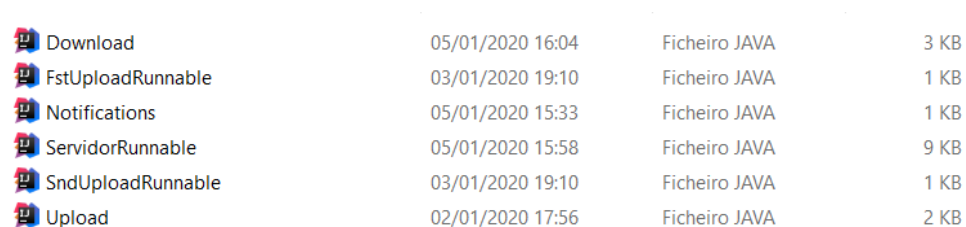
Classes que mantêm a organização da informação presente no servidor, a classe *Sistema* contém um *Map* com as músicas e os users presentes na aplicação. A classe *ThreadsNot* contém a lista de *Threads* que são responsáveis pelas notificações.

2.2.2 DownLock

Esta classe foi implementada por nós com o objetivo de dar *locks* nas descargas sempre que atingir o **MAXDOWN**, ou seja, o número máximo de descargas que ocorrem em simultâneo no servidor.

2.3 Runnable

Contém as classes que implementam *Runnable* utilizadas para várias tarefas da aplicação.









 Download	05/01/2020 16:04	Ficheiro JAVA	3 KB
 FstUploadRunnable	03/01/2020 19:10	Ficheiro JAVA	1 KB
 Notifications	05/01/2020 15:33	Ficheiro JAVA	1 KB
 ServidorRunnable	05/01/2020 15:58	Ficheiro JAVA	9 KB
 SndUploadRunnable	03/01/2020 19:10	Ficheiro JAVA	1 KB
 Upload	02/01/2020 17:56	Ficheiro JAVA	2 KB

Figura 3: Organização pasta Runnable

2.3.1 Upload e Download

São responsáveis pelos processos de upload e download por parte do cliente, visto que o mesmo pode realizar várias instruções de upload e download em simultâneo. Então é necessário a criação de uma *thread* com a classe *Runnable* correspondente.

2.3.2 Servidor Runnable

Classe chamada pelo servidor sempre que aceita um cliente no *socket*, esta classe é responsável por ler uma instrução, corrê-la e fechar o socket correspondente. No caso de receber a instrução *notificar*, esta classe fica em espera até ser necessário enviar uma notificação.

2.4 Servidor

Contém as classes correspondentes ao programa por parte do servidor.


 files	05/01/2020 03:03	Pasta de ficheiros	
 Servidor	05/01/2020 02:07	Ficheiro JAVA	1 KB

Figura 4: Organização pasta Servidor

2.4.1 Pasta files

Contém todos os ficheiros presentes no Servidor, ou seja, todas as músicas que os clientes deram upload no formato *Título - Interpretre.mp3*.

2.4.2 Servidor

Unicamente cria o *ServerSocket* e corre *threads* com o *ServidorRunnable* depois de aceitar o cliente.