Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistemas Distribuídos

Troca de Ficheiros

GRUPO 35
84656 - Hugo Cunha
86268 - Maria Pires
84167 - Susana Marques

3 de janeiro de 2020

Conteúdo

1 Introdução

Este relatório contém a apresentação do projeto desenvolvido no âmbito da disciplina de *Sistemas Distribuídos*. Serão expostos os problemas que o grupo confrontou e as soluções implementadas para a construção de um sistema de troca de ficheiros de música semelhante ao *SoundCloud*.

Para conseguir implementar este projeto recorremos a conceitos apresentados nas aulas da unidade curricular, tais como a criação de threads e controlo de concorrência baseando a implementação na arquitectura *cliente-servidor*, no qual toda a comunicação é feita via *sockets* TCP.

2 Descrição do Problema

A aplicação oferece os seguintes serviços:

- Registo e autenticação de clientes;
- Upload de músicas;
- Download de músicas.
- Pesquisa de músicas com base nas suas etiquetas.

Para um cliente efetuar o registo é necessário fornecer um *username* e uma *password*, para que posteriromente possa fazer *login* no sistema.

Fornecendo um *path* e os metadados é possível efetuar o upload de ficheiros mp3, que ficam de seguida disponíveis para download através do seu id e pesquisa através das suas etiquetas. É também possível consultar a lista de todas as músicas carregadas para a cloud.

3 Arquitetura

3.1 Music

A classe *Music* serve para o sistema gerir os dados relativos a cada musica que este contém. Esta é constituída pelo seu id, que é incrementado sequencialmente através de uma variável global para manter coerência e linearidade na atribuição destes em cada carregamento. É tambem constituido pelo número de downloads e os seus metadados. A classe *Metadados* agrupa os dados da música, sendo estes: Ano, Título, Artista e uma lista de etiquetas(tags).

3.2 Menu

A classe *Menu* implementa a interface com o utilizador. Nesta é feita a leitura da opção do menu selecionada, garantido a sua validade.

3.3 MainClient

A criação da classe *MainClient* implicou a criação de duas threads a *ClientReader* e a *ClientWriter*,é através destas que o Cliente consegue ler a informação do socket através da thread de leitura e enviar através da thread de escrita a sua resposta, de modo a comunicar com o Servidor. Assim, a *ClientReader* recebe o output do socket do Cliente e interpreta o seu conteúdo de modo a invocar os métodos necessários. A thread *ClientWriter* escreve no socket as opções escolhidas pelo Cliente.

3.4 MainServer

Nesta foram criadas duas threads, a *ServerReader* e a *ServerWriter*. É através destas que o Servidor irá ler a informação do *socket* e dar respostas ao Cliente através do socket, utilizando a thread de leitura e de escrita respetivamente. Também a classe *ServerReader* recebe o input do cliente e interpreta o pedido deste, de modo a invocar os métodos que se encontram na classe principal, ou seja, na *SDCloud*. Esta escreve o resultado desses métodos no *MsgBuffer* do cliente, de modo a que *ServerWriter* leia do buffer e envie para o socket do cliente.

3.5 User

Esta classe contém a informação de cada utilizador, nomeadamente, o username, a password e um buffer que permite guardar as mensagens enviadas para este utilizador. Assim, houve a necessidade de criar a classe *MsgBuffer* que suporta concorrência. Esta informação é ser util para a realização do login e registo de novos utilizadores, sendo possível guardar a informação dada.

3.6 SDCloud

A *SDCloud* é a classe fundamental do sistema uma vez que é a responsável pelo armazenamento da informação referente aos utilizadores e biblioteca de música existentes. Nesta encontram-se os métodos essenciais que permitem ao utilizador usufruir de todas as funcionalidades da nossa aplicação.

3.7 SDNetwork

Esta classe é usada para uniformizar o processo de transmissão de dados. Contém uma variável global: 'MAXSIZE' indicativa do tamanho máximo do fragmento do ficheiro que pode ser transmitido. Nesta classe também existem os métodos para transformar um ficheiro em String, fragmentando-o quando necessário para permitir a transmissão.

3.8 Notifier

A classe *Notifier* é fundamental para as notificações de novas músicas serem apresentadas a todos os clientes ligados ao servidor.

4 Testes/Controlo de concorrencia

Para assegurar que as secções criticas do programa funcionavam corretamente e de forma concorrente foram desenvolvidos alguns testes automáticos sobre os *uploads* e *downloads*. Estes consistiram na criação de um grande número de threads que realizam várias operações sobre as classes implementadas verificando assim a sua consistência.

5 Conclusão

Sistemas Distribuídos têm como objectivo melhorar o desempenho de um sistema dividindo responsabilidades e repartindo tarefas. É esperado que este processamento em paralelo leve a uma redução do tempo de execução. Existem várias formas de implementar paralelismo sendo que a arquitectura usada neste trabalho foi a de *cliente-servidor* tendo por base *threads* da *JVM*.

É de notar que o paralelismo tem os seus custos podendo não ser de todo proveitoso em programas com uma escala mais reduzida. A introdução de paralelismo costuma ser acompanhada

pela introdução de concorrência que por sua vez traz problemas de difícil identificação e resolução. Apesar disto, consideramos que os objetivos propostos foram alcançados, tendo conseguido desenvolver o sistema concorrente capaz de efetuar uploads e downloads e aceder aos conteúdos carregados, tendo em atenção as *features* adicionais propostas.