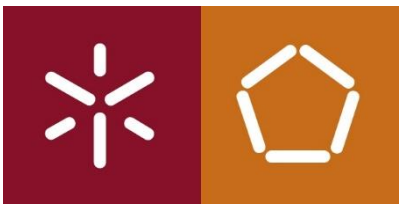
Sistemas Distribuídos



Trabalho prático – Troca de Ficheiros

5 de janeiro de 2020

Grupo nº 40

Filipa Alves dos Santos (A83631)

Hugo André Coelho Cardoso (A85006)

João da Cunha e Costa (A84775)

Válter Ferreira Picas Carvalho (A84464)




Mestrado Integrado em Engenharia Informática

Universidade do Minho

Índice de conteúdos

1. Introdução **3**

2. Desenvolvimento **3**

2.1. Estrutura da Aplicação 3

2.1.1. Cliente 3

2.1.2. Servidor 4

2.1.3. Cloud 4

2.2. Funcionalidades 5

2.2.1. Básicas 5

2.2.2. Adicionais 6

3. Conclusão **6**

1. Introdução

Neste projeto de Sistemas Distribuídos, foi-nos solicitado fazer um programa que implementasse uma partilha de ficheiros, para que músicos possam fazer upload do seu conteúdo e outros utilizadores possam fazer download deste mesmo.

Para tal, foi necessário criar um modelo Cliente / Servidor, utilizando sockets e threads. Deste modo, os utilizadores podem realizar pedidos simultâneos ao servidor e este está preparado para responder prontamente a todos os utilizadores ao mesmo tempo. Porém, para não sobrecarregarmos o sistema com diferentes pedidos, foi também pedido para implementar certas limitações relativas aos tamanhos dos ficheiros bem como o número de downloads simultâneos.

As funcionalidades básicas pedidas e implementadas por nós consistem, inicialmente, em registar um utilizador (sign up) de modo a que depois este consiga entrar na plataforma com esses dados (log in). Já dentro do sistema, os utilizadores conseguem dar publicar ficheiros de música (upload), bem como descarregar outros que tenham sido previamente carregados (download) e pesquisar por músicas com determinadas etiquetas (search). Para além disso também conseguimos efetivar as funcionalidades propostas de notificar os utilizadores de músicas novas, limitar o número de descargas simultâneas e o tamanho dos ficheiros.

Neste documento, vamos como explicar estruturamos a nossa aplicação em diferentes classes e packages e o nosso raciocínio para cada uma das funcionalidades pedidas.

2. Desenvolvimento

2.1. Estrutura da Aplicação

2.1.1. Cliente

Este package contém todas as classes relacionadas com o utilizador/cliente. Estas classes recebem o input e enviam-no para o servidor que executa a funcionalidade correspondente. Mais tarde, recebem a resposta deste e, dependendo dela, envia um output correspondente ao cliente.

- **Client:** inicia um cliente novo, isto é, cria um socket e conecta o novo utilizador com o servidor através deste. Esta conexão é usada nas threads das novas instâncias das classes Writer, Reader e Notifier, que vão servir para respetivamente enviar e receber informação do servidor. Também é criado o Menu deste utilizador específico.

- **Menu:** esta classe funciona como a interface da nossa aplicação. Apresenta ao utilizador o menu correspondente ao estado atual do programa e regista a escolha (input) do utilizador. Além disso, também contém outros métodos para pedir/perguntar ao cliente certas informações.

- **Reader:** responsável por ler as respostas do server (BufferedReader) e transmitir alguma mensagem ao utilizador e/ou mudar o estado do menu.
- **Writer:** responsável por ler o input do cliente e transmitir mensagens correspondentes ao servidor (BufferedWriter).
- **Notifier:** recebe mensagens do SendNotification do package Servidor e notifica o cliente correspondente.

2.1.2. Servidor

Este package contém todas as classes relacionadas com o servidor. Estas classes recebem as mensagens da parte cliente, atualizam corretamente a nossa "base de dados" e respondem de volta ao utilizador.

- **Server:** inicia o servidor, com uma serverSocket para se ligar aos clientes. Para cada cliente, cria um Worker que se conecta com o cliente pelo socket criado no package Cliente. Para além disto, esta classe inicia a nossa "base de dados", uma instância da classe Soundcloud.
- **Worker:** é a classe do Servidor que realmente realiza as funcionalidades pedidas pelo Cliente, que muitas vezes envolvem comunicar com a base de dados para obter informação bem como atualizar os dados que ela contém.
- **SendNotification:** é criada uma thread desta classe, uma por cada Cliente, para enviar notificações a todos os utilizadores logged in de que uma música foi carregada.

2.1.3. Cloud

Este package contém todas as classes relacionadas com armazenamento. Estas classes recebem ordens da parte servidor, e atualizam as estruturas de armazenamento de dados, quer sobre a música, quer sobre os utilizadores.

- **Musica:** define o que é uma música, tendo como variáveis instância todas as características associadas a uma música.
- **Soundcloud:** esta classe funciona como a nossa base de dados temporária, sendo que só armazena os dados do programa (músicas e utilizadores) durante uma iteração da aplicação. Também contém todos os métodos associados às estruturas de dados que contém.

2.2. Funcionalidades

2.2.1. Básicas

- **Autenticação e registo de utilizador:** primeiro o cliente tem se registar - o sistema envia "REGISTER-[username],[password]" para o servidor que, depois de verificar se algum utilizador com um username igual já existe, regista-o na base de dados (mensagem "SIGNEDUP-") ou, no caso de já existir, informa o cliente que já existe um utilizador com o username inserido (mensagem "DENIED_2-"). Agora registado, o cliente pode entrar no sistema inserindo os dados com que se registou. A mensagem "LOGIN-[username],[password]" é enviada para o servidor e este verifica se os dados inseridos correspondem a algum dos utilizadores registados. Caso esteja, manda a mensagem "LOGGEDIN-[username]" ao cliente, que muda o estado para "LOGGED" e guarda o username em questão numa variável do Menu. Se não foi autorizado, informa o cliente que os dados inseridos são inválidos (mensagem "DENIED_1-"). Por último, ele pode dar logout que funciona da maneira inversa ao login: retira-o da lista de utilizadores logged in (menciona no tópico das notificações) e altera o estado para "NOTLOGGED".

- **Publicar um ficheiro de música:** é enviada mensagem "UPLOAD-[nome],[artista],[ano],[tag_1%...%tag_n]", seguidos pelos seguintes dados: tamanho, meta-dados e conteúdo binário. O servidor recebe a mensagem e atualiza o estado do cliente para "UPLOADING" através da mensagem "UPLOADING-" e, quando o ficheiro é reconstruído no servidor, é enviada a mensagem "UPLOADED" e o estado do cliente passa para "LOGGED", visto que o upload terminou. O ficheiro fica com o nome [identificador].mp3 no armazenamento interno do servidor.

- **Efetuar uma procura de música:** a mensagem "SEARCH-[tag_1,...,tag_n]" é enviada, o servidor recebe a mensagem e vai buscar as músicas existentes na base de dados com as tag dadas. Depois, o servidor envia a mensagem "SEARCHING-" e dá-se display das músicas encontradas.

- **Descarregar um ficheiro de música:** o cliente autenticado fornece o identificador único do ficheiro pretendido e de seguida, é enviada ao servidor a mensagem serializada "DOWNLOAD-[identificador]" pelo extremo de escrita do cliente. No servidor, a mensagem é tratada e após identificar o ficheiro, este é decomposto nas suas vertentes: tamanho (em bytes), meta-dados (nome, nº downloads, artista, etc) e o seu próprio conteúdo binário, que são enviados ao extremo de leitura do cliente com a mensagem "DOWNLOADING-", que indica que este deve esperar que sejam enviados todos os meta-dados e bytes do ficheiro, atualizando o seu estado para "DOWNLOADING". Quando os bytes são todos enviados, o download é dado como terminado e o servidor envia a mensagem "DOWNLOADED-", atualizando agora o cliente para o estado "LOGGED", visto que já recebeu o ficheiro. O ficheiro fica com o nome "[identificador] - [nome], [artista] ([ano]) [[tag1,...,tagN]].mp3" no armazenamento interno do cliente.

2.2.2. Adicionais

- **Limite de descargas:** De forma a implementarmos este limite, a nossa "base de dados" possui uma variável de instância MAXDOWNLOADS, que indica o nº máximo de downloads que se podem ter simultâneos. Sempre que um download excede este limite, é colocado em espera e é notificado, mal o nº de downloads atuais seja inferior a MAXDOWNLOADS, que pode proceder ao efetivo download.

- **Notificação de novas músicas:** De modo a conseguirmos mandar uma notificação a todos os clientes autenticados no momento do upload, tivemos que criar uma estrutura de dados extra (Hashmap) na classe Souncloud que guarda o nome dos utilizadores e a instância do Worker respetiva a cada. Deste modo, cada vez que se adicionar uma música na nossa base de dados, vai ser dado uma string com o nome e artista da música nova. Cada um desses Workers vai mandar a mensagem "[notificação]" por uma nova thread da classe SendNotification, através do socket de notificações do respetivo cliente. A mensagem é recebida pelo Notifier, que alerta com um print da notificação enquanto o resto do programa pode fazer qualquer outra operação, garantindo assim que uma thread só escreve em apenas um socket.

- **Tamanho dos ficheiros limitado:** Para conseguirmos implementar tamanho limitado de envio, criamos um buffer de bytes de tamanho estático MAXSIZE, preenchendo progressivamente com conteúdo do ficheiro de música até ser enviado todo o conteúdo binário deste (usando um ciclo que detete o fim do ficheiro). Para podermos reconstruir do lado do servidor (caso seja upload) ou do lado do cliente (caso seja download), enviamos também o tamanho original do ficheiro e, usando ciclos cuja condição de paragem é: "o nº de bytes lidos é igual ao tamanho original", obtemos o ficheiro na sua totalidade, não excedendo envios superiores a MAXSIZE.

3. Conclusão

Neste trabalho prático, pusemos em prática muito do que estudámos durante este semestre ao utilizarmos diversas técnicas de controlo de concorrência, como as threads e a própria implementação de um servidor com sockets. Deste modo consideramos que fomos bem-sucedidos em criar uma plataforma de partilha de música, garantido um bom funcionamento de ações simultâneas sem sobrecarregar o sistema e evitando situações não desejadas como "deadlock".

Tivemos algumas dificuldades ao corrigir erros, sendo que cada ação implica passar por várias classes até acabar, e também ao implementar o limite de descargas. Apesar disto, estamos satisfeitos com o resultado final do projeto. Este trabalho foi essencial para o desenvolvimento da nossa compreensão de sistemas distribuídos e uma base para que no futuro façamos outros programas com estruturas semelhantes.