Relatório do Trabalho de Sistemas Distribuidos

Grupo 32
Diogo Ribeiro[A84442], Luís Maia[A84241], Ricardo Silva[A84123], and Rui Reis[A84930]

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a84442,a84241,a84123,a84930}@alunos.uminho.pt

1 Introdução: Arquitetura Geral

Implementamos a nossa solução utilizando uma arquitetura principalmente orientada á sessão, isto é, cada utilizador tem no servidor uma thread que corresponde á sua sessão, e essa thread atende os seus pedidos, devido á necessidade de implementar notificações e devido à natureza bloqueante das leituras nos sockets em java, na verdade cada utilizador vai necessitar de ter duas threads no servidor, uma para atender os pedidos, e uma que lhe enviará as notificações de novas músicas no sistema.

Implementamos também o programa sob a pré condição que os requests são independentes das respostas, isto é, se uma thread do cliente faz envia um request, esta não estará responsável por receber nem tratar da resposta, esta condição foi bastante favorável devido á natureza aleatória das respostas por parte do servidor devido ás notificações e á natureza geral do programa, em que o cliente pode emitir vários pedidos simultâneamente e não é funcionalmente necessário que um pedido seja atentido antes de este ser capaz de emitir o próximo (com certas exceções, por exemplo procura->download, o cliente fica á espera do resultado da procura para emitir o pedido de download).

2 Protocolo

Utilizamos um protocolo relativamente simples para a comunicação entre o cliente e o servidor através do socket:

2.1 Login

Pedido: "LOG+Nome+Password"
Resposta: LOG SUCESS || LOG ERROR NAME || LOG ERROR PASSWORD || LOG ERROR UNKNOWN

2.2 Register

Pedido: "REG+Nome+Password"
Resposta: REG SUCESS || REG ERROR NAME || REG ERROR PASSWORD || REG ERROR UNKNOWN

2.3 Download

Pedido: "DOW + NumeroDoFicheiro"
Resposta: "DOW NOME-ARTISTA"+ «Varios»"DOW + Chunk"+ "DOW END") || DOW ERROR NO FILE || DOW ERROR NOT LOGGED || DOW ERROR UNKNOWN

2.4 Publicação

Pedido: "PUB + Titulo + Interprete + Ano + Tags"+ «Varios»"Chunk Codificado"+ "PUB END")
Resposta:PUB SUCESS || PUB ERROR NOT LOGGED || PUB ERROR UNKNOWN

2.5 Procura

Pedido: "SEK+Tag"
Resposta: «Varios»"SEK + Titulo+Interprete+Ano+Tags"+"SEK END"

2.6 Notificação

Resposta: "NEW + Ficheiro.toString"

3 Servidor

3.1 Model

Esta classe contém toda a informação presente no servidor, Lista de Utilizadores, Metadados dos Ficheiros, e repetivos Locks, Upload Log, e o Load Manager, que serão descritos a seguir.

Users É um Map<Nome,User>, o User em si é um classe simples de utilizador que apenas contém um nome e uma password e os metodos getName e checkPassword(string), a classe Model contém também um lock (usersLock), que tem de ser obtido sempre que sejam feitas alterações a este map, por exemplo em addUser(), este funcionamento simples com um lock permite-nos efetuar o controlo de concorrência e evitar os problemas que surgem com esta, sendo as manipulações destes leves, não há a necessidade implementar nenhum algoritmos de fairness nem de limitar operações.

Files É um Map<Int,mediaFile> com funcionamento similar ao Users, um mediaFile corresponde aos metadados de um ficheiro e contém apenas métodos simples.

Upload Log Contém um ArrayList<String> (String é um mediaFile.toString()) com todos os uploads feitos até ao momento, o Número de logs e um lock com uma condition(wakeUpOnUpdate), possui os métodos sleepIfUpdated(int lastLogUpdated) que bloqueia a thread enquanto não receber um wakeUpOnUpdate Signal no caso no lastLogUpdated ser igual ao Numero de Logs Atual, contém também o método addFile que adiciona uma string aos logs, incrementa o número de Logs e faz signal na wakeUpOnUpdate condition. Esta confidition juntamente com estes métodos serão usados para implementar as notificações.

Load Manager Contém as seguintes variáveis:

- public int MAXSIZE;
- private int ticketsOut;
- private int downloadsDone;
- private final int MAXDOWN=2;
- private ReentrantLock lock;
- private Condition canDownload;

MAXSIZE corresponde ao tamanho máximo de um chunk de Download. As restantes variáveis são usadas para implementar o limite de Downloads, a fila para downloads e um algoritmo de Fairness nessa mesma Fila. Contém os métodos getTicket(),waitDownload(int ticket), freeDownload().
getTicket(), obtém o lock , incrementa os ticketsOut,guarda esse valor,desbloqueia o lock e retorna o valor.
waitDownload(int ticket), bloqueia a thread enquanto o ticket não for válido para download(ticket>(this.downloadsDone+this.MAXDOWN)), consegue isto através de um await na canDownload condition. freeDownload(), incrementa o downloadsDone, e faz signal na canDownload condition, acordando as threads bloqueadas em waitDownloade permitindo que uma delas seja besbloqueada.

Estes três métodos permitem-nos implementar um limite para os downloads com uma queue Fair e não permite que qualquer user faça mais de um download enquanto qualquer outro espera (Similar a um algoritmo de Round Robin), tal é obtido porque como fizemos o programa orientado á sessão, casa user não pode ter dois downloads a correr ao mesmo tempo e por isso só pode tirar um ticket de cada vez, se um outro chegar enquanto ele faz o download este segundo terá prioridade para o downloads sobre este (ticket inferior), independentemente do número de downloads que o primeiro pretenda fazer.

Métodos Os métodos do model são todos simples e nenhum merece consideração especial.

3.2 ServerWorker

Variáveis Contém as seguintes variáveis:

- private String loggedUserName;
- private String mediaFolderPath;
- private Socket socket;
- private model serverInfo;

Run O funcionamento geral do serverWorker é o seguinte:

```
while ((s = socketReader.readLine()) != null) {
    System.out.println("Client sent " + s);
    this.processInput(s, socketReader, socketWriter);
}
```

Lê uma linha do socket, invoca o método processInput, e no fim deste recomeça o ciclo.

Process Input Este método seleciona o método a ser invocado dependendo do tipo de request, verificando os três primeiros digitos do input:
(DOW,PUB,SEK,REG,LOG)

Register,Login Estes métodos não requerem atenção, recebem uma string de input, tratam do pedido, e enviam uma resposta dependendo do resultado do processo.

Requerimento do login Os próximos métodos requerem que o utilizador esteja logado, no caso de não estar, Search e Download simplesmente enviam a mensagem de erro apropriada, Publish, para além disso, corre o seguinte ciclo:

```
while (socketReader.readLine().equals("PUB END") == false) {}
```

Para "Limpar", o socketReader, já que o cliente faz o envio completo do ficheiro ao enviar o pedido, é uma consequência da independência dos pedidos/respostas, é um aspeto que podia vir a ser melhorado estabelecendo comunicação entre as Threads(Reader/Interface) do cliente.

Publish Este método cria um mediaFile que é adicionado ao model e faz a leitura dos Chunks do ficheiro até receber uma string "PUB END"que marca o fim do ficheiro, o ficheiro é guardado com o seu id como nome.

Download Este método, depois de verificar se o ficheiro existe e lidar com o caso de não existir, coloca a thread na Fila para download:

```
this.serverInfo.ldManager.waitDownload(
this.serverInfo.ldManager.getTicket());
```

Depois da thread obter a vez, o download é executado, isto é depois de escrever todos os Chunks na socket, respeitando o limite de tamanho(Neste caso é necessário identificar que a string correspondem a um download devido á natureza aleatória das notificações ->"DOW Chunk"). // Por fim é adicionado um download ao ficheiro e é libertada a vez na Fila dos downloads.

```
this.serverInfo.addDownloadToFile(fileNumber);
this.serverInfo.ldManager.freeDownload();
```

Search Este método recebe o input, faz a procura no model dos ficheiros que contenham a Tag (retorna ArrayList<String>[String=mediaFile.toString()]) procurada e escreve uma a um no socket("SEK String"), "SEK END"marca o fim da resposta á procura;

Send No Input Este método é chamado sempre que o serverWorker recebe um Request impróprio, escreve no socket uma mensagem de erro.

3.3 NotificationWorker

Esta classe serve para enviar a qualquer momento as notificações ao User, para que as notificações sejam em Real-Time, tem de correr num thread própria que é lançada pelo serverWorker no inicio da sua execução (Podiamos ter criado apenas apenas os método de notificação e chama lo sempre que processássemos um request, mas iamos perder a experiência Real-Time), faz uso do método sleepIfUpdated do uploadLog e pede a esse todos os novas músicas e escreve-as no socket quando acorda antes de voltar a chamar o método, guarda o número do ultimo log que escreveu no socket, com o qual chama o método referido do uploadLog. O serverWorker informa a Thread no notificationWorker quando o cliente terminar a conecção para que esta também termine a sua execução.

3.4 mainServer

Corresponde ao servidor do programa, apenas aceita connection requests de clientes e lança um server worker para servi-lo.

4 Cliente

Implementamo o cliente através de três classes, clienteInterface,serverRemoto e clienteReader, clienteInterface trata de receber o input do cliente, serverRemoto de processar esse input e clienteReader de ler as respostas do servidor e precessálas/informar o User destas.

4.1 ClienteInterface

Esta classe lida com a interação com o utilizador, informa-o dos inputs disponiveis e após receber um input e determinar o tipo de pedido a que esse corresponde, invoca o método apropriado do ServerRemoto para o processar.

Os inputs disponiveis são:

- "Help"
- "Register + Nome + Password"
- "Login + Nome + Password"
- "Publish+NomeDoFicheiro+NomeDaMusica+NomeArtista +Ano+Tags"
- "Search + Tag
- "Download + NomeDoFicheiro"
- "Quit"

4.2 ServerRemoto

Esta classe processa os inputs recebidos do clienteInterface, contém métodos semelhantes ao serverWorker, já que da perspetiva do clienteInterface, é como se fosse este que executa os métodos.

Register,Login,Download,Search Estes métodos são semelhantes, o input é recebido e é enviado o Pedido apropriado através do socket.

Publish Este método começa por verificar se o ficheiro a publicar existe, e lidar com o caso de não existir, depois disso inicia o publish escrevendo "PUB + MetaDados"no socket, seguido dos Chunks do ficheiros codificados (neste caso não precisam se ser identificados com PUB porque o envio de input por parte do cliente é determinístico, isto é, o servidor sabe que estas linhas correspondem á publicação). Por fim envia "PUB END"para marcar o fim da publicação.

4.3 ClienteReader

Esta classe que corre numa thread lançada pelo clienteInterface, mais uma vez devido á necessidade de implementar as notificações em Real-Time é responsável por ler as respostas do servidor e processar essas mesmas respostas.

Quando verifica que o socker fechou, termina a sua execução e informa o clienteInterface de tal para que este encerra também a sua execução.

Variáveis A classe contém as seguintes variáveis:

- private boolean isOpen;
- private String MediaPath;

- private String onGoingDownload;
- private ArrayList<String> searchList;
- private BufferedReader socketReader;

isOpen: É usado para informar o clienteInterface do estado da Thread do clienteReader.
onGoingDownload: Corresponde ao nome do ficheiro para o qual o download atual está a escrever
searchList: Corresponde ás musicas que satisfazem a procura que o servidor enviou até ao momento.

Process Input Este método serve apenas para apurar qual o tipo do input recebido e excecutar o método apropriado.

Register,Login,Publish Estes métodos são semelhantes, o input é recebido e informam o User através de um System.out.println com a mensagem apropriada.

Download Este método começa por verificar se o input é "DOW END", nesse caso coloca o onGoingDownload a null e informa o User que o download acabou e o nome do ficheiro.
De seguida verifica se a mensagem é de erro: Se o ficheiro não existe, se o utilizador não está logado, ou se ocorreu algum erro desconhecido e informa o utilizador de tal.
Por fim, já sabendo neste ponto que se trata de informação sobre download, verifica se o onGoingDownload atual é nulo (Isto é, a mensagem tem de ser para um novo download então) e nesse caso coloca o valor no input na variável, iniciando assim o download, no caso da variável não ser nula, significa que a mensagem corresponde à continuação de um Download, o método abre o ficheiro de nome contido no onGoingDownload e adiciona a este informação contida na mensagem.

Search Este método começa por verificar se a mensagem é de erro ("SEK ERROR NOT LOGGED"), e informa o utilizador se tal.
De seguida verifica se a mensagem marca o fim de um search ("SEK END") e nesse caso informa o User sobre todos os metadados de músicas que adquiriu até ao momento e esvazia a lista acumulada.
Por fim, se nenhuma das condições acima foi verificada significa que a mensagem contém uma música correspondente á procura por isso esta é adicionada à lista acumulada até ao momento.

Notification Este método simplesmente recebe o input e informa o utilizador da nova música presente no sistema.

5 Conclusão

Para concluir, achamos que conseguimos implementar um programa capaz de cumprir todas as funcionalidades básica e de bónus de um forma simples e relativamente eficaz, tivemos a oportunidade de implementar os conhecimentos adquiridos na disciplina sobre controlo de concorrência, Filas de espera, Await/Notify, e comunicação entre Cliente/Servidor.