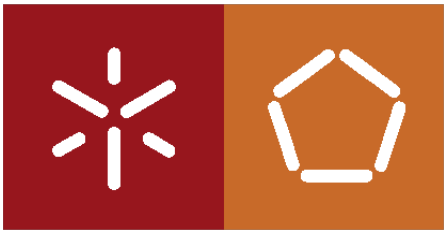


Sistemas Distribuídos

MIEI - 3º ano - 1º semestre
Universidade do Minho

Trabalho Prático


Fernando Pereira
A75496


Joaquim Simões
A77653


Marco Gonçalves
A75480


Paulo Gameiro
A72067

9 de Janeiro de 2019

Conteúdo

1 Introdução

No âmbito da unidade curricular de Sistemas Distribuídos foi-nos proposto a elaboração de um trabalho prático cuja função é implementar um serviço de alocação de servidores na nuvem e de contabilização do custo incorrido pelos utilizadores. O objetivo é então o desenvolvimento de uma aplicação distribuída capaz de fazer a gestão dos servidores.

Os utilizadores para reservar um servidor têm a possibilidade de o fazer através de um pedido ou em leilão. Sendo um pedido, um servidor fica atribuído até o utilizador decidir libertá-lo (pelo preço nominal horário correspondente ao tempo utilizado). Se reservá-lo por leilão, o utilizador indica o preço que está disposto a gastar num servidor de determinado tipo. Caso haja uma reserva a pedido em que não hajam servidores disponíveis do tipo pretendido, a nuvem tem a possibilidade de cancelar a reserva de um servidor em leilão para satisfazer a reserva.

A aplicação foi toda escrita em Java usando threads e sockets TCP, como pedido no enunciado.

2 Desenvolvimento

2.1 Account

Esta classe representa cada conta usada neste serviço, sendo identificada por um email e password.

2.2 AccountsMap

Esta classe é o aglomerado das contas, como o próprio nome indica, um HashMap com as Account existentes. A key da hash é uma String que é o email.

2.3 EmailList

Esta classe possui como atributos um ArrayList com todos os emails que têm a sessão iniciada no serviço.

2.4 Auction

Esta classe representa toda a informação de um leilão, desde o nome do maior leiloeiro e o dinheiro que ofereceu, o limite máximo, o tipo do servidor e uma lista com todos os leiloeiros presentes nesse leilão.

2.5 AuctionManager

Esta classe é responsável por gerir os leilões.

2.6 WriteMap

Esta classe é um HashMap com os BufferedWritters dos clientes, que permitem estabelecer a comunicação entre o servidor e o cliente.

2.7 Aluguer de Servidores

Esta classe representa um executável que permite aos utilizadores usufruírem do serviço.

2.8 Client

Esta classe permite que o Cliente interaja com o sistema, dando um adress e port, criando assim o socket que permite ao cliente trocar mensagens com a aplicação.

2.9 Catalogue

Esta classe representa o conjunto dos servers, os preços nominais de cada um, o montantes oferecidos em leilão e tipo de cada servidor.

2.10 MyQueue

Esta classe é usada quando um cliente quer reservar um servidor mas não o servidor não está disponível, ficando então na queue. Quando o servidor for libertado, a queue é notificada e distribui o servidor (caso haja algum cliente à espera).

2.11 Server

Esta classe possui a server Socket que posteriormente, é usada pelas sockets do cliente, ou seja, é a classe servidor. Dentro desta classe existe a classe ServerThread que serve para cada cliente ter uma thread servidor, tendo assim acesso às variáveis da classe Server.

2.12 Servers

Esta classe representa os servidores que estão a ser leiloados. Tem os vários atributos como o id, preço nominal, informação se está ocupada ou não, etc.

3 Arquitetura da aplicação

3.1 Comunicação Cliente - Servidor

De modo a cumprir com os requisitos do enunciado, nós implementados um servidor e um cliente que comunicam via sockts TCP. Quando um Cliente se conecta, a comunicação é mediada, por uma thread associada ao cliente, thread essa que comunica com a thread principal do servidor, obtendo informação e despoletando processos no mecanismo central do programa.

3.2 Funcionalidades

- **Listar Catálogo**

  O utilizador pode listar servidores livres, ocupados, para leilão, alugados por ele, todos os servidores do tipo desejado.

- **Reservar Servidor**

  O utilizador pode reservar servidores pelo preço nominal ou fazer proposta oferecendo um preço no leilão.

- **Meus Servidores**

  O utilizador pode visualizar os seus servidores.

- **Libertar Servidor**

  O utilizador pode libertar um server, deixando o desocupado.

- **Log Out**

4 Controlo de Concorrência

O objetivo deste trabalho é conseguir controlar o acesso ao conjunto de leilões e o envio de mensagens para cada utilizador, de maneira que o programa consiga lidar com vários utilizadores em simultâneo.

Como a utilização de threads permite que os mesmos dados sejam acedidos/alterados, para evitar que hajam dados incoerentes, utilizamos o mecanismo de exclusão mútua: **synchronized**. Poderiamos ter optado por utilizar o **ReentrantLocks** mas como isso nos levaria a uma implementação mais complicada e com medo de dar mau resultado, decidimos não correr riscos optando pela sua não utilização.

5 Interface

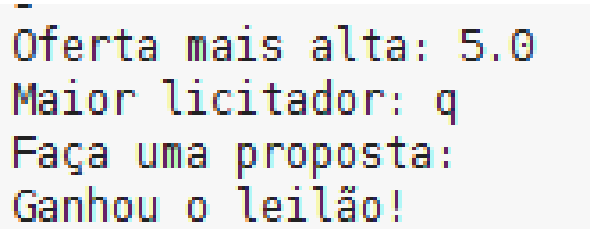
A interface da nossa aplicação é simples, pois teria de ser usada na linha de comandos.

```
Aluguer de Servidores
1 - Login
2 - Registar
quit para sair
```

```
1 - Listar Catálogo
2 - Reservar servidor
3 - Meus Servidores
4 - Libertar servidor
5 - Log Out
```

```
Servidores livres:
        cool.8k: 5
         Preço nominal: 579.63
        large.5k: 3
         Preço nominal: 208.33
        small.1k: 2
         Preço nominal: 41.67
1 - Reservar servidor pelo preço nominal
2 - Propor oferta de preço em leilão
```

```
Entrou no Leilão. Escreva o comando "quit" para sair.
Faltam 30 segundos para terminar
Oferta mais alta: 0.0
Maior licitador:
Faça uma proposta:
```



6 Conclusão

Este trabalho foi importante para consolidar e colocar em prática os conhecimentos obtidos ao longo da unidade curricular.

Tal como sabemos, o objetivo dos Sistemas Distribuídos é otimizar o desempenho de maneira a que seja possível dividir tarefas e processá-las separadamente, criando então um paralelismo e aumentar o tempo de execução. Para tal, foi necessário a implementação de threads paralelas capazes de executar tarefas concorrentemento. No entanto, ao mesmo tempo que promovemos a concorrência foi necessário de assegurar a fiabilidade dos dados, implementando exclusão mútua através do uso de classes específicas para cada estrutura de dados, permitindo controlar o seu lock implícito através do uso de métodos synchronized.

Através do desenvolvimento deste trabalho no qual nos deparamos com vários desafios de controlo de concorrência e do mantimento da integridade de dados com acesso partilhado, adquirimos um maior domínio da linguagem java, fluência no uso desta, e também uma maior destreza geral no processo de tomar decisões lógicas e organizacionais para resolver problemas no contexto de programação multi-threading.

Concluímos ter feito um trabalho satisfatório, conquanto a eficiência do código possa sempre ser melhorada; e esperamos que os conhecimentos adquiridos venham a ser tão úteis em projetos futuros como foram neste último.