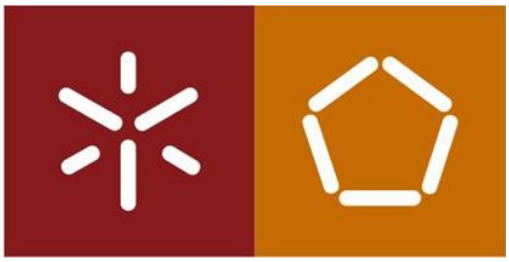


Universidade do Minho
Licenciatura em Engenharia Informática

Sistemas Distribuídos

Relatório

Trabalho Prático

16/01/2022

André Gonçalves Pinto - a93173

Rui Pedro Alves - a93252

Índice

Introdução

Este relatório foi proposto no âmbito do trabalho prático da UC Sistemas Distribuídos, no qual nos é pedida a implementação de uma plataforma de reserva de voos sob a forma de um par cliente-servidor em Java utilizando sockets e threads.

Na nossa implementação criamos um servidor com multithreading, disposto a atender vários pedidos de diferentes users.

TaggedConnection

Esta é a estrutura mais primitiva da aplicação. Esta classe contém as funções que permitem a escrita e a leitura de para um socket.

A escrita para um socket necessita um **int** e um **byte[],** o inteiro irá ser a tag para estabelecer conexão com a thread desejada . Quando é feita a leitura do socket a função retorna uma **frame**, uma estrutura que contém um **int** e **byte[].** A escrita e a leitura do socket têm ambos um lock designado.

Demultiplexer

A estrutura principal do nosso trabalho é a classe Demultiplexer.java. Esta classe é a responsável por ter uma thread única e a ler do Socket constantemente.

Quando recebe uma frame da **taggedConnection,** identifica qual é a sua tag, e, de seguida, vai buscar a estrutura indicada para essa tag, a classe **Entry.**

Tendo a **Entry** de uma determinada tag, é adicionado à sua queue os dados acabados de receber do buffer e é acordada devidamente qualquer thread que esteja à espera de ler da queue.

Toda esta implementação teve de ser devidamente pensada pois foi a fundação para o nosso programa funcionar devidamente.

Cada **entry** tem um lock, desta forma, se uma thread tentar ler da **queue** ao mesmo tempo que a thread principal insere dados, a execução dessas threads é síncrona.

Base de Dados

A base de dados do servidor contém 4 classes instanciadas quando o servidor é inicializado, **Users, GrafoCidades, Reservas** e **DiasEncerrados**.

Todas estas classes têm o seu designado lock para o servidor fazer alterações nas suas estruturas, desta maneira é possível o servidor poder atender vários clientes em simultâneo.

Mesmo que sejam feitas milhares de escritas de reservas por segundo, isso é tudo gerido apenas na classe **Reservas**, sendo que as outras classes podem ser modificadas sem necessidade de espera.

Threads Servidor

O servidor inicia bastantes threads para poder ler informação do **demultiplexer**. Para o envio de mensagens ao servidor utilizamos uma classe abstrata “Phase”, várias sub-classes desta classe “Phase” são utilizadas para formatar os inputs do user, dependendo da funcionalidade escolhida, e enviar para a respetiva Thread do servidor, o servidor irá devolver uma mensagem.

A Thread principal está em um ciclo infinito à espera de clientes que comuniquem através de um protocolo TCP. Estando a conexão aceite é criada uma nova **thread de autenticação** para esse novo socket, retomando a thread principal à escuta de novos clientes.

Algumas das threads implementadas foram as seguintes:

Thread autenticação

Esta thread é responsável por ler do demultiplexer na tag **1,** a informação de autenticação do cliente.

Formato da mensagem de autenticação do cliente: **“username”;” password”**;

O servidor pega nesta informação e invoca a função da sua base de dados **checkUser(*username,password)*.** Quando esta thread está a verificar se o usuário é válido, a lock da classe **Users** está encerrada até a função devolver um boolean. Durante este tempo nenhuma outra thread pode fazer **get** ou **set** nos dados contidos em Users.

Se o usuário não for válido o servidor retorna uma resposta inválida ao cliente. Caso seja válido, irá iniciar várias Threads para responder às necessidades do usuário.

É também iniciada uma thread que se dedica a enviar a informação do grafo do servidor para o cliente.

Apenas se o usuário for um admin que são iniciadas threads como adicionar um voo,encerrar um dia, registar um usuário, caso seja apenas um usuário normal, são iniciadas threads para marcar voos e cancelar reservas.

Thread ShowMenu

A thread “ShowMenu” é responsável por enviar ao cliente mensagens contendo, primeiramente, a lista de todas as cidades presentes no grafo, e em seguida, todas as possíveis conexões (voos) entre cidades, esta thread comunica com a thread do cliente “ThreadGetInfoServer” fornecendo a esta dados, ficando também, o cliente com acesso ao grafo.

Formato da mensagem 1: Número de cidades e respetivas cidades.

Formato da mensagem 2: Possíveis voos entre cidades (delimitador “@”).

```
Sent [3] 9;Turim;Pyongyang;Beijing;California;Porto;Madrid;Braga;Nevada;Veneza;
Sent [3] Veneza;California;Turim;Nevada;@Beijing;@Pyongyang;@Madrid;Beijing;California;@Braga;@Braga;California;
@Porto;Madrid;Veneza;Braga;@Turim;@Turim;@
```

Thread HandleVoos

Esta thread lê do demultiplexer na tag **4,** a informação de Reserva de um voo vinda do cliente “PhaseBooking”, onde através do algoritmo BreadthFirst é criada uma stack com o caminho entre a cidade de origem e a de destino e enviada à thread do servidor.

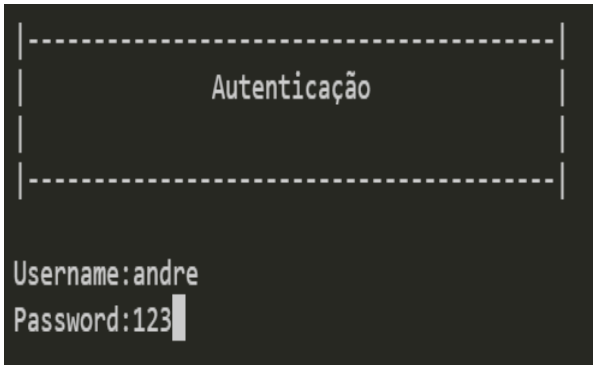
O servidor faz a verificação do caminho, se este é válido então tenta criar uma reserva desde que o dia da reserva não esteja cancelado e haja espaço suficiente no voo.

Formato da mensagem input do cliente: **“Origem”;”Destino”;”Dia”**;

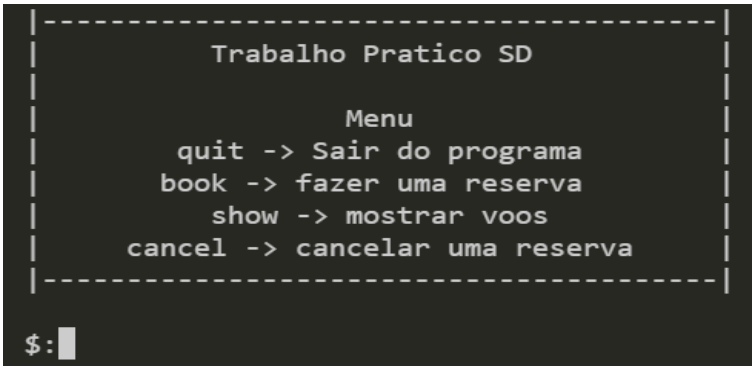
A thread irá criar a respectiva reserva se os dados inseridos forem válidos.

Execução

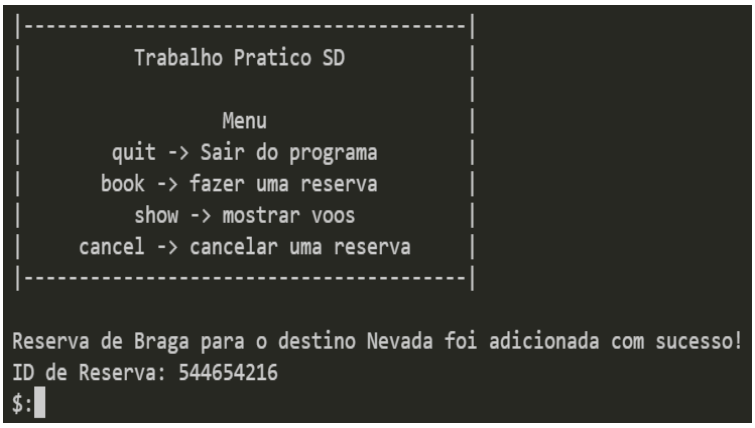
Autenticação



Menu Principal



Reserva de um Voo



Reserva de um voo feita com sucesso

Conclusão

No desenvolver da aplicação deparamo-nos com alguns obstáculos, não sendo, a implementação obtida a implementação desejada. Uma das alterações na implementação é o caso da reserva de voo ser efetuada, apenas recebendo a origem, destino e dia como input e através de um algoritmo de breathfirst é calculado o caminho com menos escalas.

Em suma pensamos ter alcançado o essencial e fundamental para o funcionamento de uma aplicação paralela com várias threads a funcionar e a fazer operações de escritas e leituras.