Universidade do Minho
Escola de Engenharia

Mestrado Integrado em

Engenharia Informática

1,2,3 Vendido - LEllões

Projeto da disciplina de Sistemas Distribuídos

Trabalho realizado por Grupo 15:

Ana Isabel Castro a55522

Lisandra Silva a73559

Lúcia Abreu a71634

Luciano Silva a75155

2 de Janeiro de 2016

1. Introdução e breve descrição do enunciado

Neste relatório apresentaremos o trabalho desenvolvido para a disciplina de Sistemas distribuídos. Este trabalho consistiu na implementação de uma aplicação distribuída que permita a gestão de um serviço de leilões. Pretende-se que este serviço seja acedido por dois tipos de clientes: vendedores e compradores. Clientes que pretendem leiloar um item, devem criar um novo leilão indicando uma descrição do item. O serviço atribui um número a cada leilão. Os potenciais compradores podem licitar o item indicando o número de leilão e um valor em euros. A qualquer momento o vendedor pode encerrar um seu leilão em curso e o serviço deve declarar vencedora a oferta mais alta registada até ao momento. Os utilizadores devem poder interagir, usando um cliente escrito em Java, intermediados por um servidor multi-threaded também escrito em Java.

Para alcançar estes objetivos foram aplicados todos os conhecimentos adquiridos no âmbito da disciplina de Sistemas Distribuídos principalmente a comunicação via sockets e os mecanismos de controlo de concorrência.

2. Arquitetura da aplicação

Para o desenvolvimento desta aplicação foram criadas algumas classes em Java sendo as principais a classe do Servidor do leilão e a classe do Cliente do leilão.

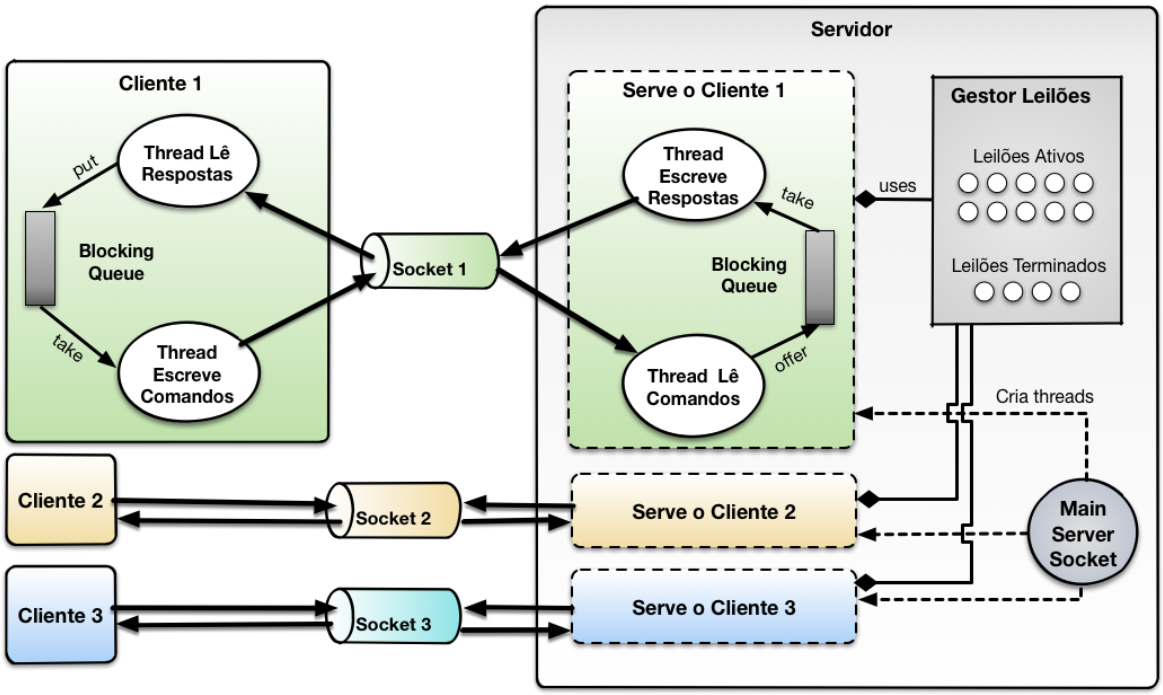

Figura 1 – Arquitetura geral da aplicação

2.1. Comunicação Cliente-Servidor

Os clientes comunicam com o servidor através de sockets TCP. Quando é iniciado, o servidor abre um serversocket associado a uma porta específica, ficando à espera que os clientes se liguem. Os clientes por sua vez criam um socket com o qual se tentam ligar a essa porta aberta pelo servidor. Quando recebe uma conexão, o servidor inicia duas novas threads: uma para atender os pedidos que vierem desse cliente, executando esses pedidos e escrevendo as respostas numa queue, na qual estão as respostas destinadas a esse cliente; a outra thread está continuamente à espera de ler respostas dessa queue para enviar ao cliente. A thread principal do servidor permanece à escuta de novas conexões.

Optou-se pela implementação desta queue, onde são escritas as mensagens para o respetivo cliente, pois assim quando um leilão é terminado, é chamada o método "terminar" do respetivo objecto leilão, onde é escrita a mensagem de aviso do término desse leilão para todos as queues dos clientes que tenham feito uma licitação nesse leilão. Da mesma forma, é enviada uma mensagem extra a um cliente quando este detinha a maior licitação sobre um leilão e esta é ultrapassada. A classe escolhida foi o ArrayBlockingQueue que é uma classe específica do Java, que oferece métodos que bloqueiam enquanto não há nada para ler na queue ou bloqueiam enquanto não há espaço para escrever na queue, para leituras e escritas respetivamente.

2.2. Classe Cliente

A classe Cliente vai representar os utilizadores da aplicação e vai ter como função estabelecer a comunicação com o servidor disponibilizado uma interface textual através da qual os vendedores/licitadores do serviço interagem com o sistema. Assim, esta classe vai ter como principais variáveis de instância um socket através do qual se liga com o servidor e uma queue (também ArrayBlockingQueue) onde vão ser escritas as respostas que chegam do servidor. Além disso, o cliente será composto por duas threads, uma delas está sempre à espera de comandos do cliente, mandando-os para o socket e de seguida ficando à espera da resposta que aparecerá na queue. A outra thread está passivamente à espera de mensagens do servidor pelo socket, e, no caso de se tratar de uma resposta de um dos comandos então escreve-a na queue, no caso de se tratar de uma mensagem de aviso do término de um leilão em que o cliente tenha participado, então escreve-a diretamente no ecrã do cliente já que se trata de uma mensagem sem correspondência com nenhum comando do cliente enviado.

Definimos que os utilizadores da aplicação vão ser de apenas um tipo, ou seja, um utilizador registado pode ser comprador e vendedor ao mesmo tempo, não necessitando de criar outra conta. De seguida apresentamos as opções de interação disponibilizadas aos utilizadores:

Menu principal:

1 - Registar

2 - Login

s – Sair

Menu após login:

1 – Inicial Leilão

2 – Listagem dos leilões do momento

3 – Licitar

4 – Finalizar um leilão

s – Sair

2.3. Classe Servidor

A classe servidor é classe que corre a base do sistema de leilões, que processa os pedidos dos clientes e responde de acordo com estes.

O funcionamento desta classe é baseado em outras duas: uma para atender os pedidos que vierem do cliente, executar os pedidos e escrever as respostas numa queue que tem as respostas que vão para esse cliente e outra que está continuamente a ler dessa queue sempre que alguma coisa é escrita nela.

Para o funcionamento desta classe, ela tem um ServerSocket que faz a conexão a uma porta para o qual os clientes tentaram conectar-se através do seu Socket, produzindo um novo Socket do lado do servidor. Também temos duas classes, de leitura e escrita de comandos/respostas ao cliente que partilham entre elas uma queue, da qual servirá para manter temporariamente respostas destinadas ao cliente, enquanto não estas não forem enviadas. Para além destas, existe também uma classe GestorLeilão que contém todos os leilões criados e uma classe Conta que contem todas as contas dos clientes registados.

2.4. Classe Leilão

A classe Leilão vai guardar a informação relevante sobre o respetivo leilão e tem por isso como variáveis de instância, nomeadamente, o item a que o leilão se refere, o valor da licitação mais alta até ao momento e respetivo licitador, e quem é o vendedor do item.

Além disso tem também um HashMap<String, BlockingQueue<String>> onde se guarda todos os utilizadores que fizeram licitações nesse leilão e as suas respetivas queues, para assim quando um leilão termina, este mapa é percorrido e é escrito em cada queue o aviso de que o leilão já terminou e qual o respetivo vencedor. Além da mensagem de terminar, também é enviada uma mensagem para a queue de um cliente sempre que este perde a licitação mais elevada de um dado leilão. Tem também um boolean que diz respeito ao estado do leilão; este booleano existe para auxiliar o controlo de concorrência, uma vez que quando o leilão é terminado pode haver posteriormente outros clientes a

tentar fazer a licitação deste leilão e este boolean garante a segurança de não permitir licitar um leilão que já tenha sido terminado.

Por último, a classe Leilão tem também um Lock, de modo a garantir que não há mais do que um cliente a fazer uma licitação do leilão simultaneamente. Isto poderia ter sido garantido declarando o método atualizaLicitacao como synchronized, no entanto ao optarmos pelo lock garantimos também que um utilizador não pode fazer uma licitação se simultaneamente o "dono" do leilão o estiver a encerrar, uma vez que para aceder a estes dois métodos é feito o lock do leilão.

2.5. Classe GestorLeilão

A classe GestorLeilao é a classe responsável por gerir os leiloes ativos correntes. Desta forma tem dois Maps do tipo <Integer, Leilao>. Um para os atuais e um para os leiloes terminados. A chave do Map é a ordem do leilão criado. Por exemplo se é o 1º leilão que existe nesta aplicação ou o 50º. De forma a garantir as funcionalidades necessárias pela gestão dos leilões, esta classe disponibiliza os métodos:

```
public List<String> getLeiloesActivos(); (Este método é o responsável por obter a lista dos leilões ativos.)
public Map<Integer,Leilao> getLeiloesTerminados();
public Leilao getLeilaoAtivo(int numLeilao);
public Integer adicionaLeilao(Leilao leilao);
public void terminaLeilao(Integer num);
public int actualizaLicitacao(int numLeilao, String username, float valor,
BlockingQueue<String> q);
```

Em todos eles é controlado a concorrência através dos ReentrantLocks com condições de escrita ou leitura. Permitimos hajam múltiplos leitores ao mesmo tempo a aceder aos mapas, mas apenas uma thread a escrever. Esta funcionalidade está encapsulada na classe RWLock. A concorrência do leilão é também controlada internamente, por exemplo ao listar os ativos para ter a certeza que obtemos os leilões corretos sem que alguém os esteja a alterar entretanto.

2.6. Classe Contas

A classe contas vai ser uma classe auxiliar que vai ter como variáveis de instância um Map<String,String> que vai corresponder às contas dos utilizadores. Cada par chave-valor do Map corresponde ao username e password de cada utilizador. Esta classe tem como funcionalidade gerir as contas dos utilizadores e para tal, disponibiliza os métodos:

```
public boolean validaAcessoConta (String username, String password);
public synchronized boolean adicionaConta (String username, String password);
public synchronized void removeConta(String username);
```

Note-se que o método `adicionaConta` é synchronized para garantir o controlo de concorrência.

3. Conclusões

No desenvolvimento deste projeto, deparamo-nos com algumas dificuldades e com a realidade de que concorrência e paralelismo não é um conceito fácil de explorar e tem que ser visto com cuidado. Com ele conseguimos adquirir conhecimentos de concorrência e desenvolver o raciocínio na resolução de problemas de que outra forma não seria possível atingir através das aulas PL.

Com este trabalho também nos foi possível expandir os nossos conhecimentos de java pela exploração de objetos que já existiam e ajudam no controlo de concorrência como é o caso do BlockingQueue<> que oferece métodos que implementam concorrência.

Em suma, com este trabalho conseguimos adquirir conhecimentos úteis e indispensáveis no nosso desenvolvimento académico e assim melhorarmos as nossas habilidades como engenheiros informáticos.