Unidade Curricular de Sistemas Distribuídos

Gestor de requisições/tarefas

Licenciatura em Engenharia Informática

Ano Lectivo: 2014/2015

Grupo:

Xavier Fernandes, Nº a55838
Carlos Morais, Nº a64306
Filipe Ribeiro, Nº a64315

Índice

Introdução

O trabalho proposto em enunciado define a implementação de um sistema distribuído que permite a gestão de tarefas de um armazém. Deste enunciado surgem vários problemas, sendo de destacar a concorrência entre o acesso aos recursos do sistema. O programa desenvolvido deve garantir o acesso em exclusão mútua às zonas críticas; que um utilizador que pretenda aceder a uma determinada funcionalidade deverá inevitavelmente poder fazê-lo e que nunca haja situações de interbloqueio ao aceder aos recursos do sistema. Para isso, utilizamos mecanismos de exclusão mútua e variáveis de condição. Foi também necessário o uso de *sockets* para estabelecer comunicação entre clientes e servidor.

Implementação

Ferramenta

Uma ferramenta será constituída por uma *string* nome, que a identificará, e dois inteiros para a quantidade e disponibilidade da mesma.

De forma a permitir que as ferramentas sejam acedidas concorrencialmente, a ferramenta tem um *lock* e uma *condition maisDisponivel*. É feito um *lock()* sempre que se pretende abastecer, ou reservar/devolver uma determinada quantidade. É feito um *unLock()* à Ferramenta no fim das respetivas operações.

O método *reservar(int quantidade)*, reserva, se estiver disponível, a quantidade. Caso não esteja a *Thread* será adormecida á espera do sinal *maisDisponivel*. Sempre que a Ferramenta seja abastecida ou lhe seja devolvida uma quantidade reservada, será enviado o sinal *maisDisponivel* para todas as *Thread's* adormecidas na Ferramenta para que possam verificar se já existe disponibilidade.

Tipo de Tarefa/Tarefa

Um tipo de tarefa é definido pelos utilizadores do sistema. Cada tipo é identificado por uma *string* id, e tem um *TreeMap<String, Integer> pedidos*, de nome da ferramenta para a quantidade a ser reservada da mesma. O facto de os pedidos estarem ordenados por ordem alfabética do identificador das ferramentas, permite que as mesmas sejam acedidas ordenadamente por *Thread's* diferentes, evitando que possa ocorrer *deadlock* entre tarefas que pretendem reservar ferramentas em comum.

Uma tarefa é criada para executar um tipo de tarefa.

A tarefa será definida por uma *string id*, *string utilizador* que identifica o utilizador que a criou e um *boolean* terminado, que quando for *true* indica que a tarefa já foi terminada.

Existe também um *lock* e uma *condition* termina. O método *espera()*, caso a tarefa não esteja terminada, adormece a *Thread* em execução à espera do sinal *termina*. Quando o método *termina()* é executado na tarefa, é enviado o sinal termina a todas as *Thread's* adormecidas à espera do sinal termina

.

Assim cada tipo de tarefa, tem um conjunto com todas as tarefas criadas para o seu tipo, o *Map<String, Tarefa> tarefas*, de identificador da tarefa para o objeto Tarefa.

Armazem

O sistema tem um armazém que será partilhado por todas as *Thread's* criadas para atender os clientes. Este é constituído por um conjunto de tipos de tarefas e ferramentas. Existem também dois *locks*, para que seja possível mapear concorrencialmente as ferramentas e os tipos de tarefas.

No nosso entender existiam duas abordagens para a implementação da gestão de tarefas no armazém.

Na primeira, para cada tarefa executada, apenas seriam reservadas as respetivas quantidades de cada ferramenta, quando todas tiverem a quantidade disponível suficiente. Isto poderia causar *starvation*, não por causa da aplicação, mas da gestão real do armazém, pois a tarefa poderia nunca conseguir efetuar as suas reservas, pois em diferentes instantes de tempo alguma das ferramentas requeridas não teria quantidade disponível suficiente.

Na segunda abordagem, que foi a escolhida, para cada tarefa executada, as quantidades pretendidas para cada ferramenta vão sendo reservadas, assim que fiquem disponíveis, sendo apenas devolvidas quando a tarefa for terminada. O problema desta solução adotada, é que enquanto uma tarefa espera pela disponibilidade de uma determinada ferramenta, não liberta as quantidades que já conseguiu reservar de outras ferramentas, quando estas poderiam estar a ser requeridas por outras tarefas.

Embora tenhamos optado pena segunda abordagem, a melhor solução estaria dependente do caso real.

Comunicação Servidor/Cliente

Servidor

O Servidor tem um Armzém e um conjunto de utilizadores definidos pelo próprio sistema.

No servidor é criado um *socket* para que os clientes possam estabelecer um conexão com o servidor.

Para cada nova conexão com um Cliente é criada uma nova *Thread TrataCliente*, essa *Thread* ficará encarregue de receber o pedido de autenticação do utilizador e, posteriormente, os seus pedidos. Assim o Servidor poderá atender concorrentemente vários Clientes. Para cada pedido recebido do Cliente, é criada uma *Thread TrataPedido*, que verifica o pedido recebido e executa-o no *Armazem*, assim a execução de um novo pedido não ficara pendente da conclusão de um pedido anterior e vários pedidos serão executados concorrentemente no *Armazem*. Para cada pedido concluído, é enviada uma resposta para o Cliente.

É ainda criada uma *Thread ThreadTerminal* que disponibiliza a utilização da consola localmente com as mesmas funcionalidades permitidas aos clientes. O funcionamento da *ThreadTerminal* é equivalente ao *TrataCliente*, reutilizando a mesma *Thread* para executar os pedidos.

Cliente

Cada Cliente cria uma conexão através do *socket* previamente definido no Servidor. Cada pedido lido no consola do Cliente é enviado para o Servidor, em cada Cliente é criada uma *Thread ClienteRead* para ler as respostas aos pedidos enviados para o servidor, de modo a permitir que um novo pedido não esteja pendente de uma resposta a um pedido anterior.

Manual de utilização

login e logout

Para utilizar o sistema, no lado do servidor ou no lado do cliente, é necessário que o utilizador efectue o *login*. Para isso deverá utilizar a seguinte *sintaxe*:

```
login:username:password
```

o utilizador poderá a qualquer momento efetuar *logout* com o seguinte comando:

```
logout
```

Adicionar ferramentas

O utilizador poderá adicionar/abastecer ferramentas com o seguinte comando:

```
abastece:ferramenta:quantidade
```

caso a ferramenta já exista no armazém, o comando anterior apenas irá atualizar a sua quantidade.

Adicionar tarefa

As tarefas deverão ser adicionadas especificando o seu nome, o material necessário para a sua execução bem com a quantidade desse material.

```
adiciona:nome_tarefa:ferramenta1:quantidade:…:ferramentaN:quantidade
```

Execução e conclusão de tarefas

Quando o utilizador pretender que determinada tarefa seja executada, deverá usar o seguinte comando

```
executa:nome_tarefa
```

este comando irá devolver ao utilizador um identificador único correspondente a tarefa em execução. Para dar uma tarefa como concluída deve ser usado o seguinte comando

```
termina:id_tarefa
```

Lista de tarefas

O sistema permite que sejam listadas as tarefas existentes bem como as suas instâncias em execução, identificando os tipos de tarefas e os utilizadores que criaram as instâncias em execução.

```
listagem
```

Sair do servidor

Para encerar o servidor, deverá ser executado o seguinte comando

```
sair
```

este comando apenas está disponível na consola do servidor.

Conclusão

Após efetuar vários testes, não detectamos situações de acesso simultâneo a recursos, situações de *starvation* ou situações de *deadloks*, fazendo estas parte dos grandes problemas na implementação de um sistema distribuído.

Depois de terminado o trabalho, chegamos a conclusão que poderíamos ter feito algumas coisas de forma diferente, nomeadamente o uso de *stub* no cliente e *skeleton* no servidor.