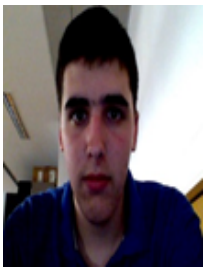
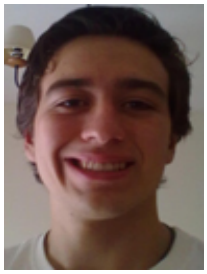
SISTEMAS DISTRIBUÍDOS - SD 2014/2015

Licenciatura em Engenharia Informática
3º Ano
Universidade do Minho

TRABALHO PRÁTICO - GESTÃO DE ARMAZÉM



Miguel Machado
67641



Pedro Cunha
67677

Pedro Silva
67751

Introdução

No âmbito da Unidade Curricular de Sistemas Distribuídos foi nos pedido que desenvolvêssemos uma aplicação para gestão de armazéns, tarefas e utilizadores utilizando os conceitos lecionados nas aulas práticas.

Para facilitar a realização do projeto reutilizamos as classes Warehouse, Cliente, Servidor, ClientHandler, utilizados no decorrer do semestre.

Este trabalho tem o objetivo de consolidar os conhecimentos adquiridos sobre partilha de memória, situações de Cliente-Servidor, utilização efectiva de Threads e programação concorrente.

De seguida está apresentada uma breve descrição de cada uma das classes utilizadas na realização deste projecto:

Objecto

A unidade básica de um armazém será o “Objecto”, que terá nele guardados o seu nome, quantidade em armazém, um ReentrantLock. O ReentrantLock será utilizado para que, ao mesmo tempo, apenas se realize uma operação (incQuant, decStock) sobre este Objecto.

Warehouse

A classe Warehouse é composta por um HashMap<String, Objecto> em que a chave de cada objecto será o seu nome, um ReentrantLock, e uma Condition para caso a quantidade do Item em questão seja 0. Mais uma vez, usamos o ReentrantLock para que no mesmo instante temporal se realize uma operação de escrita (supply, consume) na Warehouse.

O método supply aplica lock ao armazém e verifica se já existe algum Objecto com o nome que se quer fornecer. Caso exista, acede-se ao Objecto em questão e atualiza-se o stock. Caso contrário uma nova instância de Objecto é criada com os parâmetros passados ao método supply(Nome e quantidade), e depois adicionada ao HashMap. Finalmente sinalizam-se todas as Threads que estão em espera (devido à Condition vazio) que há novos Objectos no armazém e liberta-se o lock.

O método consume recebe um ArrayList de String (nomes dos objectos). Para simplificar, cada elemento do ArrayList irá consumir 1 unidade. O método aplicará o lock ao armazém, esperará enquanto não exista o Objecto, ou que a sua quantidade seja 0 (vazio.await()). De seguida, chegando ao último else, fará-se o consume do Objecto em questão (chegando a este ponto está assegurado que o Objecto existe e a sua quantidade é superior a 0). Finalmente o lock é libertado.

Tarefa

Esta classe é utilizada para a definição/execução de uma tarefa. Nela tem guardados um ID's (único a cada tarefa) de forma a ser guardados num HashMap com facilidade, um ArrayList<String> para a lista de Nomes de Objectos necessários para a sua realização, um boolean que indica se a tarefa está concluida ou não, uma String que representa o nome do Utilizador que a criou e outra String que é o nome que o utilizador dá a tarefa.

Menu/Input

Estas classes, fornecidas no âmbito de Programação Orientada aos Objectos, servirão para facilitar a criação de Menus (em linha de comandos) e simplificação de Input e seus problemas relativos a Excepções.

Utilizador

Esta classe terá em si guardados o nome do utilizador, a sua palavra-passe, um boolean que indica se o utilizador já tem sessão iniciada no servidor, um ArrayList<Tarefa>, um para as tarefas criadas, e um HashMap<Integer,Tarefa> para as tarefas que este utilizador tem de momento a executar.

Utilizadores

Esta classe contém um HashMap<String, Utilizador> em que a chave é o nome do Utilizador e um ReentrantLock, para obrigar a que apenas se efectue um login de cada vez, já que é a única operação afectada pelo lock. Esta classe está suportada por 4 tipos de Exception, que são UnexistentUserException (Quando se tenta efectuar login a partir de um nome de utilizador inexistente no Map), InvalidLoginException (Quando a Password fornecida não coincide com a guardada no utilizador do Map), UserAlreadyLoggedException (Caso um utilizador já esteja ligado e esteja a tentar ligar-se a partir de outro fio de execução) e UsernameAlreadyExistsException (Quando no acto de registo, o nome de utilizador já exista no Map), que servirão para controlar os vários erros que poderiam ocorrer no decorrer do registo/login.

TarefaMap

Esta classe é utilizada para registar as tarefas criadas e as tarefas a executar de todos os utilizadores no lado do servidor.

Para esse efeito a classe contém 2 HashMap<Integer,Tarefa>,um para guardar as tarefas criadas e outro para registar as tarefas a executar. E um ReentrantLock de mobo a bloquear quando se adiciona/remove uma tarefa de alguma dessas listas.

O bloqueio e necessário pois e usado a quantidade de tarefas já existente para dar o ID á tarefa, e com o bloqueio garanti-mos que não há ID's iguais para tarefas diferentes.

Database

Esta classe combina uma instância da classe Warehouse, uma da classe Utilizadores e uma da classe TarefaMap.

Uma instância desta classe será o estado passado como parâmetro na criação de ClientHandler/AdminHandler.

AdminHandler

Esta classe que implementa Runnable (será uma nova Thread para correr uma instância desta classe) servirá para apresentar ao administrador do Sistema a mesma funcionalidade que um cliente tem, para além de tratar da sua interação com o servidor. Para além da variável "dados" (do tipo Database), passada como parâmetro ao construtor, terá guardado também uma variável do tipo String representativa do username do Utilizador atual.

ClientHandler

Esta classe que implementa Runnable (será uma nova Thread para correr uma instância desta classe), servindo de intermediário entre cliente e servidor. No momento da sua construção será usado o Socket que o mesmo está a utilizar e uma variável de estado, que no nosso caso será a variável dados do tipo Database. Para além da mesma teremos guardada uma variável do tipo String representativa do username do Utilizador atual.

Tem também um BufferedWriter e um BufferedReader para ler e escrever no socket.

Cliente

Esta classe será a classe que se liga ao servidor e que oferece uma interface com o utilizador. Terá um Socket que será criado, a partir do qual serão criados um BufferedReader e um BufferedWriter, tratando a partir daqui de mostrar os menus e esperar por input.

Servidor

Esta classe terá como variável de classe uma Database vazia, que será povoada pelos clientes ou pelo próprio AdminHandler, para além de um ServerSocket, cria um AdminHandler para que o mesmo disponibilize na consola para uso local (sem o uso de sockets) a mesma funcionalidade do Cliente. Depois de aceitar uma conexão cria um novo ClientHandler para permitir que este servidor funcione com várias conexões (Multi-Threaded).

Funcionamento

Após a iniciação do servidor, a primeira coisa que acontece, é carregar o estado do servidor guardado na última vez. Este projeto já tem um estado pré-guardado que contém três utilizadores o user1, user2 e user3 com passes iguais ao seu nome de usuário.

De seguida é iniciada uma thread com a classe AdminHandler para fornecermos a funcionalidade de cliente no lado do servidor. É também através deste interface que podemos desligar o servidor caso seja necessário guardando o estado atual .

Finalmente o servidor entre em ciclo infinito (até ser parado manualmente), em que a única coisa que faz é esperar por clientes para estabelecer ligação.

Após estabelecer ligação com um cliente o servidor inicia uma nova thread com uma instância de ClientHandler para de seguida voltar a esperar por a ligação de um novo cliente.

Cada cliente tem os seus sockets de comunicação, que são fechados assim que o cliente termina a sessão.

Também quando um cliente termina a sessão é gravado o estado do servidor.

Comunicação

A comunicação entre o cliente e o servidor é restrita e controlada por nós. Apenas se manda dois tipos de mensagens inteiros e strings terminadas em ‘\n’.

Isto é garantido pois nos inputs do utilizador apenas aceitámos estes tipos de inputs.

Conclusão

Durante a realização deste projecto tivémos de superar os desafios da concorrência e partilha de variavéis que devem estar sempre atualizadas.

Sentimos também que auxiliou a cimentação dos tópicos dados nesta unidade curricular. E serviu para nos despertar para a importância deste tipo de programação, fazendo-nos refletir sobre os tipos de serviço que não teríamos hoje sem ela.

Trabalhos Futuros

No futuro algumas coisas a corrigir neste projecto seria:

- A possibilidade de starvation no caso de alguém tentar iniciar uma tarefa para o qual não a recursos suficientes e nunca ninguém adicionar esses recursos.
- Terminar a sessão dos utilizadores de uma maneira segura caso o servidor falhe.
- Tratar de falhas repentinas no lado do cliente.