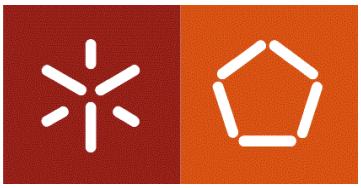

Universidade do Minho
Licenciatura em Engenharia Informática

µ-Kickstarter

Sistemas Distribuídos – Ano lectivo 2013/2014

Lucas Oliveira, nº 54729

Samuel Marques, nº 57791

Tiago Silva, nº 60030

Introdução

O projecto que nos foi apresentado propõe a elaboração de um programa do tipo "cliente-servidor" fazendo assim com que o trabalho seja dividido em 2 aplicações que são precisamente um cliente e um servidor.

Relativamente à aplicação cliente, esta desempenha o papel de ponte entre o utilizador e o servidor (onde se encontram as funcionalidades propriamente ditas). É apresentada sob a forma de menus na linha de comandos, tendo o utilizador acesso as funcionalidades do programa seleccionando o número a que cada uma corresponde. Esta é também responsável pela recolha da informação fornecida pelo utilizador e seu envio para o servidor. Por fim, recebe as respostas do servidor, interpretando-as e apresentando-as ao utilizador.

Do outro lado temos a aplicação servidor onde estão codificadas as funcionalidades propriamente ditas. Aqui é gerado o *output* correspondente as informações recebidas a partir do cliente e enviado de novo para este.

Descrição do programa

Aplicação Cliente

Começando pela aplicação cliente, esta é formada por 2 classes: ”Cliente” onde se encontra a sua *Main* e uma classe “Menu” a partir da qual são construídos os menus com que o utilizador interage.

É uma aplicação bastante simples na medida em que apenas trata de receber a informação do utilizador e envia-a para o servidor, a fim de obter uma resposta que encaminha de novo para o utilizador.

Para que esta comunicação seja possível o cliente estabelece uma ligação (TCP) com o servidor através de um *socket*. Depois disto, toda a informação que o cliente recolhe do utilizador é enviada sob a forma de linhas de texto para o servidor através do *OutPut Stream* do *socket*. Essa informação é interpretada pelo servidor que devolve depois uma resposta para o *InPut Stream* do *socket* do cliente, que a trata e apresenta de seguida ao utilizador.

Aplicação Servidor

Passando agora ao servidor, este é mais complexo pois é aqui que são implementadas as funcionalidades pedidas. É constituído por 8 classes: “Servidor” onde se encontra a sua *Main*, “ClienteHandler”, “Projecto”, “Projectos”, “Utilizador”, “Utilizadores”, “GestaoProjs” e por fim “CompareDecr”. Esta ultima é utilizada na ordenação da lista de utilizadores que mais contribuíram para um projecto.

A classe “Servidor” tem como principal funcionalidade lançar uma *thread* (“ClienteHandler”) para cada cliente que se ligue ao servidor. É aqui também onde se carrega a informação existente de outras sessões (caso exista) e guarda a informação da sessão actual. Por fim, está presente ainda nesta classe a funcionalidade que nos permite desligar o servidor de uma forma segura (guardando os dados da sessão actual).

Na classe “ClienteHandler” é controlado cada pedido que chega do cliente através da leitura dos “comandos” que chegam ao *InPut Stream* do *socket* do servidor. Assim cada linha é constituída por uma instrução seguida do número de argumentos que esta necessita. Todas as palavras estão separadas por “::”. Por exemplo:

instr::arg1::arg2

Lida a instrução é chamado o método que lhe corresponde e a ele são passados os argumentos necessários. Depois é enviada a resposta conforme o *output* do método para o cliente através do *OutPut Stream* do *socket* do servidor.

A classe "Projecto" foi definida para representar um projecto com todas as características que este deve possuir sendo depois utilizada na classe "Projectos" onde temos um *TreeMap* que guarda todos os projectos. Aqui temos também as operações possíveis sobre os projectos como por exemplo listar os projectos (financiados ou não), adicionar novos projectos ou verificar se é possível ou não investir num dado projecto.

À semelhança da classe "Projecto", a classe "Utilizador" representa um utilizador e todos os atributos que são necessários para este assim como a classe "Utilizadores" guarda todos os utilizadores (utilizando um *HashMap*) e efectua as operações necessárias sobre estes.

Por fim, a classe "GestaoProjs" permite a todas as *threads* ter acesso à informação, quer dos utilizadores, quer dos projectos.

Controlo de concorrência

É necessário em certas situações ter cuidado com a concorrência entre *threads*. Estas situações foram identificadas e tratadas conforme o caso. Passamos agora a identifica-las e a descrever o seu tratamento.

Login

No login é necessário controlo de concorrência pois é possível que 2 utilizadores tentem aceder à mesma conta ao mesmo tempo. Para resolver esta situação o que fizemos foi bloquear o utilizador (instância) que foi acedido pelo primeiro utilizador físico a tentar acede-lo, libertando-o depois de mudar a variável de instância "login" (esta variável é do tipo *boolean* e indica-nos se o utilizador já está ou não activo). Assim se um segundo utilizador físico tentar aceder aquela conta irá receber uma mensagem a dizer que a respectiva conta já está a ser utilizada pois esta terá o valor da variável "login" a *true*.

Registo de utilizador

No caso do registo de um novo utilizador, uma vez que estamos a alterar uma estrutura de dados (*HashMap* que contém os utilizadores), esta é bloqueada durante a inserção do novo elemento (utilizador) para evitar problemas de sobreposição de dados.

Inserção de novo projecto para financiamento

Este caso é um pouco mais complicado do que o anterior pois como nos é dito no enunciado, um utilizador ao inserir um projecto para financiamento deve ficar bloqueado até que projecto esteja financiado. A estrutura que guarda os projectos (neste caso uma *TreeMap*) vai ser alterada (inserção do novo projecto) sendo bloqueada e depois libertada após a inserção. De seguida a *thread* do cliente é adormecida e será acordada temporariamente cada vez que o projecto é financiado para informar o utilizador. Por fim, esta é acordada definitivamente quando o projecto está totalmente financiado, podendo o utilizador voltar a utilizar o programa normalmente.

Realizar investimento

O controlo de concorrência aqui é feito para impedir que dois utilizadores financiem o mesmo projecto ao mesmo tempo. Assim, sempre que um utilizador financia um projecto, este ultimo é bloqueado, é registada a alteração no seu financiamento e libertado de novo. É também acordada a *thread* correspondente ao utilizador que criou o projecto para o informar do financiamento. Quando o financiamento do projecto é concluído, esta mesma *thread* é acordada, libertando o utilizador e informando que o financiamento foi concluído.

Conclusão

Com este trabalho tivemos um primeiro contacto com a programação concorrente (para além da matéria leccionada ao longo do semestre) e por esse motivo deparamo-nos como novos problemas que até ao momento não tínhamos sentido na programação sequencial, desenvolvendo assim novas capacidades para dar resposta aos mesmos.

Por fim, gostaríamos de referir que nos sentimos satisfeitos com o trabalho desenvolvido, uma vez que conseguimos atingir os objectivos que nos foram propostos, não só em termos de funcionalidades do programa mas também porque crescemos como programadores, adquirindo um novo leque de ferramentas na área da programação concorrente.