Relatório do Projeto SD

Bruno Veloso (a78352) João Pimentel (a80874)
Rodolfo Silva (a81716) Pedro Gonçalves (a82313)

Janeiro 2019

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Sistemas Distribuídos
Grupo 10

Conteúdo

1 Introdução

Este relatório aborda a resolução do projeto prático de Sistemas Distribuídos. O projeto consiste em construir um sistema de alocação de servidores na nuvem e da contabilização do custo incorrido pelos utilizadores destes.

2 Estratégias de Resolução dos Objetivos do Projecto

Para a resolução deste projeto, o mesmo foi divido em três partes.

A primeira parte é relativa aos Utilizadores, e para estes é guardada a sua informação pessoal(nome de utilizador, password), os servidores a que este tem acesso e também o custo da utilização dos mesmos.

A segunda parte do projeto consiste nos Servidores em si. Para este é guardado o seu identificador, o nome do mesmo, o seu PVP, o valor de licitação mínima e também o nome do Utilizador que possui os direitos de utilização do servidor.

A terceira parte debruça-se sobre o Leilão em si, onde os Utilizadores podem alugar servidores, através de um leilão ou a pedido. Para isso o Leilão tem acesso a todos os servidores disponíveis para aluguer, tanto por leilão ou a pedido, e também tem acesso a todos os Utilizadores que pretendam alugar um servidor.

2.1 Controlo de concorrência

Tendo em consideração o panorama da UC, o utilizador vai tentar conectar-se ao servidor através de uma *port* e um *host*. Neste caso, a *port* será 2222 e o *host* trata-se do *localhost*. Caso a conexão seja efetuada, serão gerados leitores de input e output, sendo inicada uma Thread para receber as respostas do servidor, começando um ciclo para enviar mensagens ao servidor. Consoante a mensagem que o cliente enviar ao servidor, algo será feito, dentro dos comandos definidos previamente. Caso contrário, apenas será indicado que o comando não é válido.

O servidor vai configurar um socket, ficando à espera de conexões ao mesmo. Quando um utilizador se conecta, uma thread é criada e adicionada à lista de threads. A classe userThread possui a maioria do código, sendo o local onde é analisado o input gerado pelo utilizador, bem como o output a retornar. Cada thread irá receber a lista dos servidores em leilão. Note-se que estes servidores não passam de itens que os utilizadores podem adquirir.

Sabendo que ter várias threads a aceder aos mesmos dados pode criar *race conditions*, tem que se ter em mente as possíveis dependências de leitura e escrita. Não é suposto um utilizador licitar com o lance 30€, enquanto outro utilizador licitou 40€ mas este valor ainda não se registou no objeto. Assim, garantindo que todos os acessos aos objetos são sincronizados, as possíveis colisões de leitura e escrita que se podiam ter são impossibilitadas.

Note-se que não faria sentido ter construtores sincronizados, pois apenas uma Thread cria o objeto, não criando colisões. Todos os métodos nos itens em leilão, ou seja, os servidores, são synchronized, incluindo getters e setters, prevenindo, assim, possíveis *race conditions*.

Todas as alterações aos estados dos servidores possíveis para aquisição, pedidos para visualizar servidores, libertar um servidor, ter conhceimento da dívida atual, pedir um servidor de um certo tipo, têm que ter forma de gerir as colisões. Assim, o método adotado foi sincronizar métodos e/ou o objeto em questão, não permitindo que mais do que uma thread entrasse na região crítica. Tendo em conta que as threads partilham a lista de servidores para aquisição, sincronizando a mesma, apenas uma thread pode trabalhar sobre os servidores num determinado momento, sendo garantido que o acesso a esta variável se torna atómico.

2.2 User Interface

Neste capítulo será discutido as opções que um Utilizador possui depois deste iniciar sessão no Leilão.

Aquando do ínicio de sessão o Utilizador tem a opção de executar vários comandos para realizar diferentes tipos de tarefas:

1. *bid* - Este comando permite que o Utilizador faça uma licitação no leilão para o aluguer de um dado servidor;

2. *list* - Este comando apresenta todos os servidores disponíveis para aluguer;

3. *mine* - Este comando apresenta todos os servidores na posse do Utilizador;

4. *release* - Este comando permite ao Utilizador cancelar o aluguer de um dado servidor;

5. *debt* - Este comando apresenta o custo atual da utilização dos servidores alugados pelo Utilizador;

6. *buy* - Este comando permite ao Utilizador alugar um servidor ao PVP, sem ter de licitar num leilão;

7. *want* - Caso não existam servidores disponíveis para aluguer direto, o Utilizador utiliza este comando para alugar um servidor que estava previamente alocado para leilão.

Com a execução de todos estes comandos torna-se possível para o Utilizador utilizar todas as funcionalidades pedidas no enunciado do projeto.

3 Conclusões

Cumprindo todos os requisitos propostos no enunciado, o grupo considera o trabalho um sucesso, apesar de existirem aspetos a melhorar, nomeadamente a possível inserção de um registo em ficheiro dos dados dos utilizadores, uma possível gestão mais rigorosa de colisões, criação de mais comandos para o utilizador, entre outros.

Apesar disto, o desenvolvimento do projeto permitiu solidificar o conhecimento adquirido durante a UC, permitindo uma melhor visão sobre as vantagens e perigos do uso de *threads*, bem como gerir regiões críticas.

Em suma, o trabalho corresponde aos desafios propostos no enunciado do projeto, apesar de possuir falhas que, num futuro próximo, serão implementadas como forma de solidificar, ainda mais, o conhecimento adquirido.