Universidade do Minho
Escola de Engenharia

Sistemas Distribuídos

Trabalho Prático
Grupo 6

André Rodrigues Soares (a67654)
Bruno Manuel Macedo do Nascimento (a67647)
Manuel Maciel Roriz Oliveira (a68410)
Rui Alves dos Santos (a67656)

Introdução **3**

Estrutura da aplicação. **4**

Métodos de comunicação **5**

Controlo de concorrência **5**

Funcionalidades Obrigatórias **6**

Autenticação e Registo de Utilizador 6

Reservar Servidor a pedido 6

Reservar uma instância em leilão 6

Libertar um servidor 6

Consultar a sua conta corrente 7

Conclusão **7**

Introdução

As aplicações distribuídas lidam normalmente com vários utilizadores e, como tal, devem estar preparadas para responder a vários pedidos feitos simultaneamente.

Assim, neste tipo de sistemas, o controlo de concorrência torna-se essencial para que o sistema tenha um bom desempenho, mantendo ao mesmo tempo a consistência dos dados.

Neste trabalho procuramos explorar as funcionalidades do modelo cliente servidor com comunicação orientada à conexão via TCP, através do desenvolvimento de um servidor multithread que funciona como intermediário na comunicação entre clientes, utilizando a linguagem de programação Java para implementar tanto o servidor como o cliente.

O tema desenvolvido consiste na implementação de uma aplicação distribuída que permite a gestão de um serviço de cloud, no qual existem variados tipos de servidores para aluguer.

O cliente tem a possibilidade de alugar qualquer servidor que se encontre livre ou alugado por leilão, mediante pagamento de um valor fixo por unidade de tempo. Este servidor é do cliente por tempo indeterminado. Existe ainda a possibilidade de alugar um servidor por leilão por valores inferiores, mas com a possibilidade de perder o aluguer para um cliente que pague o valor fixo, quando todos os servidores para esse efeito já estão todos em uso.

Estrutura da aplicação

BaseDeDados - Classe onde estão guardadas as hashmap dos utilizadores, servers e leilões; é também a classe que trata da parte lógica do programa.

Mensageiro - Classe que é criada como uma thread do cliente que está sempre pronta a “ouvir” mensagens vindas do servidor.

MensageiroSV - Classe que é criada como uma thread sempre que um servidor pretenda enviar uma mensagem a outro utilizador que não o seu. Isto acontece de modo a que os utilizadores mais lentos não atrasem os utilizadores com uma melhor conexão.

Server - Classe de dados, onde é guardada toda a informação de um server. Os servers são os produtos de venda da cloud.

Utilizador - Classe de dados onde é guardada toda a informação dos utilizadores do sistema.

AcceptClient - É o “verdadeiro” servidor mas apenas fica à espera de conexões de clientes, e quando estes clientes se conectam cria uma thread da classe “Servidor” que funciona como um handler do cliente.

Cliente - Classe que trata de dar outputs ao utilizador e receber os inputs, e que depois comunica com a classe “Servidor”

InterfaceC - Classe onde estão guardados os menus do programa “Cliente”

Leilao - Classe de dados onde é guardada a informação de um leilão a decorrer, também funciona como thread que faz “wait()” durante 30 segundos para depois acabar o leilão

Servidor - Classe que recebe mensagens da classe “Cliente”, ou seja do utilizador, e depois envia os dados à base de dados para esta responder com os resultados.

Métodos de comunicação

Para fazer a comunicação entre cliente e servidor foi usado um protocolo de mensagens codificadas, por exemplo, quando o utilizador queria fazer o logIn, enviava ao servidor a mensagem “1-user-pass” através do primeiro elemento da mensagem, neste caso o “1”, o servidor sabia que o cliente queria fazer o login. Este método foi utilizado ao longo do programa para o servidor perceber quais as vontades do cliente com a mensagem que lhe estava a enviar.

No lado do cliente a partir do login o cliente passava a ter uma thread “Mensageiro” que estava sempre a “ouvir” as mensagens vindas do servidor, visto que este podia mandar mensagens já esperadas, como por exemplo a lista de servidores quando esta era pedida, ou mensagens inesperadas, por exemplo, quando um leilão acabava e o servidor queria que o cliente soubesse. Para resolver possíveis atrasos de mensagens esperadas, o cliente sempre que precisava de uma mensagem vinda do servidor ía à classe “Mensageiro” onde, caso a flag estivesse a 0, o que queria dizer que não havia mensagens fazia uma “wait()”. Sempre que o “Mensageiro recebesse uma mensagem esperada ele mudava a flag para 1, e enviava um “notifyAll()” para “acordar” o cliente que já tinha a mensagem preparada para ler.

Para diferenciar as mensagens esperadas das inesperadas foi usado o mesmo método, as mensagens sem prefixo eram mensagens esperadas e as mensagens inesperadas possuíam o prefixo “1”, por exemplo "1-Parabéns ganhou o leilão do servidor".

Controlo de concorrência

Durante a resolução do trabalho foram utilizadas algumas técnicas de concorrência para impedir que os resultados fossem alterados a meio de uma leitura, causando resultados não verdadeiros.

Na classe “BaseDeDados” foram criadas três locks, “lUtil”, “lServers”, “lLeiloes”, essas locks eram bloqueadas sempre que a classe pretendia correr uma delas, seja para adicionar algo, ou para procurar algo. Depois quando era necessário fazer diversas alterações num “server” a classe “BaseDeDados” fazia um bloco “synchronized (server){}” de modo bloquear todo o acesso a esse server até o trabalho estar concluído. Todas as estruturas de dados, “Utilizador”, “Server”, “Leilao”, têm os seus métodos synchronized para evitar que informação seja alterada por duas threads diferentes ao mesmo tempo.

Funcionalidades Obrigatórias

- ## Autenticação e Registo de Utilizador

Em ambos os casos o servidor comunica com a classe “Base de dados”, que começa por utilizar um lock explícito para bloquear qualquer acesso ao hashmap onde estão guardados os utilizadores. No fim o lock é desfeito de modo a que outras threads possam aceder à hashmap.

- ## Reservar Servidor a pedido

Quando um cliente expressa a vontade de alugar um servidor em primeiro o servidor chama um método na classe “Base de dados” que bloqueia, com um lock explícito, outros acessos à hashmap “servers” onde estão guardados os servidores, depois recolhe informação de todos os servidores que estão livres, caso nenhum servidor esteja livre então a base de dados vai procurar quais os servers em leilão ou comprados através de leilão, envia a informação ao servidor que depois envia a informação ao utilizador. O cliente seleciona o servidor que pretende alugar, o servidor envia a informação à base de dados, que bloqueia quaisquer outros acessos ao “server” em questão, altera as informações necessárias e informa o cliente que o server foi comprado com sucesso. Caso o servidor tenha anteriormente reservado através de leilão, a class “servidor” cria uma classe thread “MensageiroSV” que envia uma mensagem ao antigo dono do servidor.

- ## Reservar uma instância em leilão

Quando o programa é iniciado encontram-se alguns servers em leilão, quando um leilão é iniciado uma thread da classe “Leilão” é iniciada e fica em “wait()” durante 30 segundos, quando essa thread acorda, bloqueia qualquer novo acesso a esse leilão, e depois qualquer acesso ao server em leilão, caso tenha sido comprado atualiza a informação do server, cria uma thread da classe “mernsageiroSV” que manda uma mensagem ao cliente que ganhou o leilão, caso ninguém tenha licitado no leilão então o server passa para a lista de disponíveis.

Um cliente comunica ao servidor que pretende reservar um server através do sistema de leilão, de seguida o servidor apresenta uma lista de servidores em leilão e o cliente seleciona o server que quer licitar, o valor, e os dados do cliente são guardados na classe “Leilão” que enquanto está a guardar os dados está bloqueada, proibindo o acesso a outras threads, e depois o programa ocorre como descrito no parágrafo anterior.

- ## Libertar um servidor

O cliente pede para ver uma lista dos server que é dono, e depois selecionando um server que quer libertar, esse server é bloqueado e não pode ser acedido por mais nenhuma thread. Depois a base de dados dá “reset” à informação do server para o meter a disponível, compara a hora de compra com a hora que é

cancelada a compra e a esse valor é multiplicado o preço de aluguer, que pode ser fixo ou à escolha do cliente em caso de ter ganho o leilão, esse valor é guardado no utilizador.

- Consultar a sua conta corrente

O utilizador envia uma mensagem a dizer que quer ver o seu saldo, o servidor avisa a base de dados, a base de dados bloqueia a Hashmap “utilizadores” até retirar o utilizador em questão e depois retira o saldo do utilizador e envia ao servidor para ele enviar ao cliente.

Conclusão

Durante o desenvolvimento deste trabalho conseguimos pôr em prática muito do que estudámos durante este semestre, usando os mecanismos de controlo de concorrência do java conseguimos criar uma aplicação distribuída capaz de garantir o bom funcionamento de um serviço de aluguer de servidores tal como esperado.

Ao longo do projecto foram surgindo alguns problema relacionados com a linguagem java, ou até mesmo com a maneira que inicialmente foi pensado o projecto. Contudo a maior dificuldade enfrentada foi perceber quando era necessário aplicar controlo de concorrência ou quando não era. No entanto consideramos que conseguimos alcançar uma solução desejada, evitando situações de “deadlock” ou “starvation”.