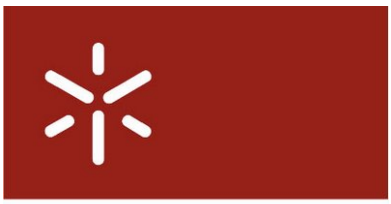

Universidade do Minho

Trabalho prático de Sistemas Distribuídos

MIEI - 3º ano - 1º semestre
Universidade do Minho

Trabalho Prático - Java

António Gomes (a67645), Guilherme Miranda (a55735) 6 de Janeiro de 2019

Conteúdo

.1 Introdução 2

.2 Funcionalidades 2

.3 Networking 3

.4 Cliente 3

.5 Servidor 4

.5.1 Classes Auxiliares 4

.5.2 Métodos 5

.6 Conclusão 6

.1 Introdução

Um sistema distribuído tem como objectivo fornecer serviços distintos a vários utilizadores. Estes sistemas têm a particularidade de serem por norma constituídos por uma ou mais máquinas que trabalham em conjunto de modo a que, do ponto de vista do utilizador, este é um único sistema conciso.

Este projeto prático tem como objectivo desenvolver um sistema distribuido de aluguer e leilão de servidores. Clientes podem aceder, através da aplicação em causa, às seguintes medidas.

1. Reserva de servidores a pedido, ficando reservado para uso do utilizador até este o libertar.
2. Reserva de servidores em leilão em que cada utlilizador propõe o preço que está disposto a pagar pela reserva de um servidor de um determinado tipo.

Os utlilizadores podem interagir com o serviço através de uma aplicação cliente que comunica com o servidor. Ambos devem ser implementados em Java, realizando-se a comunicacão entre ambos com recurso a sockets TCP. O protocolo de comunicação deverá ser baseado em texto e orientado a linha. Para evitar ficar vulnerável a clientes lentos, cada thread do servidor não deverá escrever em mais do que um socket.

Servidores a serem alugados seguem as seguintes normas de funcionamento e escalonamento de serviços:

1. Um utilizador que não possua uma conta pode criar a conta.
2. Um utilizador poderá libertar um servidor que lhe foi anteriormente atribuido.
3. Um utilizador poderá pagar um valor fixo por um servidor, sem necessidade de ir a leilão, ou poderá licitar um valor que este está disposto a pagar pela utilização de um servidor.
4. Em caso de indisponíbilidade de servidores, uma reserva a leilão poderá ser forçosamente terminada de modo a este seja atribuído a um pedido a pronto que tenha surgido.

.2 Funcionalidades

Ao cliente é fornecido uma aplicação remota que, através de um terminal, efetua a ligação com o servidor. A aplicação de terminal do cliente está munida de algumas ferramentas que permitem ao cliente executar várias tarefas.

Ao ser iniciada a aplicação cliente, é possível executar a autenticação ou, caso o cliente em questão não tenha conta ainda, criar conta. Uma conta exige um login e uma palavra passe. Após a autenticação a aplicação permite ao cliente executar um pedido de um servidor de uma determinada categoría por leilão em leilão, ficando este em espera de forma a acumular licitações de vários clientes escolhendo por fim a maior licitação.

Licitações consistem na ondocaçao pelos clientes indicarem do preço a que estão dispostos a pagar. Cada cliente apenas licita uma vez. O cliente disposto a pagar o maior valor obtém o servidor.

Servidores podem também ser obtidos pagando um preço fixo sem necessidade de leilão. Pedidos de servidores a pronto têm prioridade sobre pedidos sobre leilão pelo que quando um novo pedido a pronto chega e não há servidores livres de momento, pode ser retirado a um cliente que tenha alugado por leilão de forma a este possa ser alocado ao pedido de maior prioridade.

À medida que servidores vão sendo alocados ou libertados, o servidor mãe tem a responsabildiade de marcar cada servidor como estando ocupado ou livre, decidir quais os pedidos que devem ser atendidos, armazenar informação relativa a contas de utilizadores.

É importante garantir que não existem erros no tratamento de pedidos. Estes erros podem surgir maioritariamente devido ao acesso simultâneo de várias threads às mesmas estruturas de dados quando estas tratam de mútliplos pedidos ao mesmo tempo. É importante também Garantir a integridade dos dados armazenados e da informação a ser transmitida aos clientes.

.3 Networking

O servidor e os respectivos clientes estão em constante comunicação sendo os dados transmitidos entre estes sobre o formato de texto UTF-8 de linhas únicas por cada mensagem.

Cada mensagem transporta informação relativa à açao que o cliente pretende tomar, quando esta circula do cliente para o servidor, podendo esta ação ser de login, requisito de um servidor para leilão, indicação de que o servidor alocado a este pode passar a estar disponível, entre outras.

Por parte do servidor, este encontra-se em constante comumicação para com os clientes de modo a enviar respostas a pedidos, responder a consultas por parte dos clientes, indicar quando um servidor é retirado a um cliente, entre outos.

A comunicação entre clientes e servidores é efectuada através do uso de sockets TCP. Por parte do servidor, todas as interações com clientes são tratadas por parte de um método clientHandler o qual toma recurso de vários outros métodos os quais implementam as tarefas necessárias.

.4 Cliente

O Cliente é o programa através do qual o utilizador comunica com o Servidor. Devido a isto o que o Client faz é navegação de menus e resposta a queries por parte do MasterServer.

O main do Cliente simplesmente corre o método mainMenu() caso o utilizador não tenha feito login, e corre a função loggedInMenu() de pois de o utilizador o ter feito. Caso a flag exit tenha sido posta a 1, o programa termina.

O método mainMenu() é usado para correr as funções que comunicam com o Server para fazer log-in e fazer o registo de um utilizador novo, dando instrução ao MasterServer para correr os métodos correspondentes. Também tem a opção de sair, que põe a flag exit a 1, fazendo o Cliente fechar.

O método loggedInMenu() é usado para correr as funções que tratam da reserva de servidores e tudo o que se passa em volta disso, dando instrução ao MasterServer para correr os método correspondentes. Esta função corre a função respetiva à ação que o utilizador quer pedir ao MasterServer, apresentando ao utilizador as queries apropriadas.

O método logIn() começa por perguntar ao utilizador qual o seu e-mail. Caso o String dado seja nulo o Client pergunta outra vez. Se não, o Client envia a string ao MasterServer. Se o Server responder que não existe nenhum utilizador com esse e-mail o Client informa o utilizador e volta ao menu. Se o e-mail existir, o Client pergunta ao utilizador a sua password, que depois envia para o MasterServer se está não for nula. O Client acaba a função dando ao utilizador a resposta do servidor e pondo a flag loggedIn a 1 se o login foi bem sucedido.

O método registerAcc() é muito semelhante ao método logIn(). O servidor pergunta o e-mail e a password que o utilizador deseja e imprime as respostas do MasterServer. No fim o Client não faz login automático e volta para o main menu.

O método requestServer() faz o Client ler as respostas do MasterServer até receber uma mensagem que indique o fim da lista de servidores. Depois o utilizador escolhe um servidor e o Client envia ao MasterServer e imprime a sua resposta.

O método bidOnServer() começa como requestServer(), imprimindo a lista de servidores. Depois de o utilizador escolher um servidor válido o Client pede ao utilizador para indicar o preço pelo qual deseja reservar o servidor pedido. Se o preço não for menor que o dado pelo último utilizador e se não for maior que o preço base do servidor pedido, o Client recebe e imprime uma última mensagem de sucesso do MasterServer.

O método freeServer(), pede ao utilizador o código do servidor que quer libertar. Depois o Client imprime a resposta do Server.

O método fetchRentedServers() imprime a lista, dada pelo MasterServer, de todos os servidores de momento alugados pelo utilizador.

O método consultDebt() recebe do servidor o valor total que o utilizador deve devido a alugueis de servidor.

.5 Servidor

.5.1 Classes Auxiliares

O ficheiro MasterServer tem três classes auxiliares para além da classe principal MasterServer e da classe ClientHandler() que comunica com o Client , estas são as classes Server, User e Database.

A classe User é usada para representar cada utilizador no sistema, contendo as variáveis relativas aos dados correspondentes a cada utilizador. Esta classe contém variáveis que guardam a informação do e-mail, password e da dívida do utilizador. Existe tambem uma lista chamada userServers que contém todos os Id dos servidores actualmente alugados pelo utilizador. Finalmente a variavel l é usada para fazer lock do utilizador quando este está a ser utilizado.

A classe Server é usada para representar cada servidor no sistema, contendo as variáveis relativas aos dados respectivos a cada server. Esta tem variáveis que contêm informação do servidor respectivos ao Id do servidor, ao nome do servidor, aos preços de aluguer, ao estado do servidor, ao código de libertação deste e ao e-mail do user que o está a alugar, se aplicavel. A classe também tem a variavel l que é usada para fazer lock do servidor quando este está a ser utilizado.

A classe Database contém maps para os users e para os servers e contém também uma variavel l que é usada para fazer lock à database. Esta classe só existe para se poder fazer lock à database para impedir que outras instâncias do ClientHandler tenham acesso a esta enquanto está a ser usada.

.5.2 Métodos

O método main() da classe masterServer chama o método init() e depois fica à espera de que um client tente ligar-se. Quando isto acontece ele cria uma nova thread da classe clientHandler() para processar esse client.

O método init() da classe masterServer inicializa um objecto da classe database e adiciona 9 servidores a esta. Também adiciona dois users para testes.

O método run() da classe clientHandler() recebe as escolhas do utilizador quando o Client está em um dos seus menus. Depois de receber uma mensagem, o run() corre o método correspondente a essa escolha, excepto quando o utilizador escolhe sair, nesse caso, o run() põe a sua flag exit para 1, fecha a ligacão ao Client e finalmente termina a thread.

O método registerUser() pergunta ao Client qual é o e-mail desejado, repetindo a questão se for dado null. Depois, pergunta ao Client qual a password desejada, repetindo a questão se for dado null. Caso o e-mail dado já existir na database, é enviada uma mensagem ao Client explicando isso, se não, é enviada ao client uma mensagem de sucesso.

O método userLogIn() pergunta ao Client qual o seu email, repetindo a questão se a resposta recebida for null. Apos receber o email, o método compara este com os email de todos os users na database, dando uma mensagem ao Client e fazendo return caso um user com esse email não exista. Depois, pergunta ao Client qual é a password, repetindo a questão se a resposta recebida for null, e compara esta com a password do user com o email dado. Depois o método envia uma mensagem de sucesso ao Client, se a password for igual à dada, ou de falhanço caso oposto.

O método showServersRented() dá ao Client uma listagem de todos os servidores que o user está a alugar de momento, mostrando o ID, nome, tipo e código de libertação do servidor.

O método grantServerRequest() manda ao Client a lista de servidores disponíveis e pergunta qual o servidor escolhido. Se a resposta não for o id de um dos servidores mostrados o método diz ao Client que não existe ou que já está a ser usado e termina. Se o id dado for válido, o método muda o estado do servidor para Y, coloca o id do servidor na lista de servidores do user e o e-mail do user na variavel userEmail do servidor. Finalmente, o método dá uma mensagem de sucesso ao Client.

O método auctionServer() manda ao Client a lista de servidores disponiveis e pergunta qual o servidor escolhido. Se a resposta não for o id de um dos servidores mostrados o método diz ao Client que não existe ou que já está a ser usado e termina. Depois, pergunta ao Client qual

o valor que deseja pagar. Se este valor for menor que o valor actual ou maior que o valor base do servidor, o método dá uma mensagem ao Client e termina. Se o valor dado for aceite o método muda o estado do servidor para A, coloca o id do servidor na lista de servidores do user, e o email do user na variavel userEmail do servidor e, se este exitir, tira o servidor da lista do utilizador antigo. Finalmente, o método dá uma mensagem de sucesso ao Client.

O método freeServer() pede ao Client o código de libertação de um servidor. Depois o método compara este código com os códigos de todos os servidores que o user está a alugar. Se encontrar o que tenha o código de libertação igual ao dado o método muda o estado do servidor para N, retira o servidor da lista de servidores alugados pelo utilizador. Finalmente, dá ao Client a resposta de sucesso. Se nenhum dos servidores do user tiver código de libertação igual ao dado, o método dá ao Client uma mensagem a dizer que não tem alugado nenhum server com aquele código de libertação.

O método showUserDebt() simplesmente dá ao user o valor da dívida do user.

.6 Conclusão

Não foi possível cumprir todos os objectivos exgidos sobretudo devido à existência de runtime exceptions que não foi possível tratar por parte da aplicação cliente. Contudo disponibilizou-se também uma versão da aplicação servidor que funciona em modo stand alone para efeitos de teste.