Programação Concorrente (3º ano de L.C.C.)

Trabalho Prático

Relatório de Desenvolvimento

Paulo Jorge Alves de Sousa (A69318)

June 2, 2016

Contents

Chapter 1

Análise do código

1.1 Cliente Java

O cliente Java que implementei é bastante simples, sendo que a sua função é receber os inputs do utilizador, comunicá-los ao servidor e consoante a resposta deste encaminhar o utilizador para as funções pretendidas. Há cinco classes, a main presente na classe Cliente.java comanda toda a execução do programa, a classe Login.java é responsável pela autenticação do utilizador que já se encontra registado, a classe Register.java é responsável pelo registo dos utilizadores, de seguida têmos duas classes a classe Condutor.java destina-se a atribuir a disponibilidade de um condutor para realizar uma viagem e a classe Passageiro.java é responsável pelo pedido de um utilizador por uma viagem, estas duas classes justificam-se pois a um condutor e um passageiro são pedidos diferentes inputs e terão também de ser direcionados para diferentes partes do código.

Começo então por explicar a classe Cliente.java, esta classe como já foi dito rege todo o lado cliente, começa por verificar se os argumentos necessários para a conexão do socket foram inseridos, caso não tenham sido, lança um erro e termina. Após esta fase é inicializado o socket e os buffers de leitura e escrita usados para comunicar com o socket e o standard input.
De seguida dá-se inicio ao processo de autenticação pedindo ao utilizador que indique se pretende registar-se ou efetuar o login chamando de seguida o método da classe correspondente, isto é feito por intermédio de um switch statement com duas possibilidades.
Após a fase de autenticação estar concluida com sucesso, o utilizador é encaminhado para a função que define a disponibilidade de um condutor em fazer uma viagem ou a de um cliente realizar uma viagem, novamente isto é controlado por um case statement que consoante o utilizador é um condutor ou um passageiro o envia para a função correspondente. Por fim são testadas as excepções caso as haja, lança o erro respectivo.
A classe Register possui um método que recebe como argumentos os buffers de leitura e escrita, começa por enviar ao servidor a palavra registo, para o servidor

se encaminhar para a função responsável pelo registo de novos utilizadores, de seguida recebe do utilizador um username e uma password que envia para o servidor e por fim pede ao utilizador que escolha entre condutor ou passageiro, premindo 1 ou 2 consoante for a sua opção, esse inteiro é retornado pois será utilizado na classe Cliente pelo segundo switch.
A classe Login é semelhante à classe Register sendo que também é pedido um username e uma password, estas são comunicadas ao servidor que ao recebê-las testa se o utilizador existe e devolve uma resposta, consoante essa resposta for afirmativa a execução prossegue, caso não o seja a execução do programa é interrompida e apresentada uma mensagem de erro ao utilizador.
A classe Passageiro possui um método, novamente recebe como argumento os buffers de leitura e escrita, envia ao servidor a palavra "passageiro", para este se direcionar para o sitio correspondente. De seguida é pedido ao utilizador que indique a latitude e longitude da sua posição de origem e de destino, tudo isto é comunicado pelo socket para o servidor, para este efetuar o registo da posição, o cálculo do custo da viagem e o tempo esperado de chegada do condutor assim que emparelhado com um condutor.
Passada esta fase dá-se inicio assim que o servidor nos informar que emparelhou com um condutor, à viagem, o passageiro é informado pelo servidor que um veículo se está a dirigir para o local de partida e o tempo esperado até este chegar, sendo depois informado que pode cancelar a viagem a qualquer momento antes de entrar no veículo, sendo que esse cancelamento não terá custos se for efectuado até um minuto depois do emparelhamento. É pedido ao utilizador que cancele a viagem ou indique que entrou no veículo assim que este se encontrar na posição de origem. Por fim a classe Condutor, à semelhança dos anteriores é incializado com os buffers como argumento, comunica-se ao servidor que é um condutor, é solicitado ao utilizador que indique o modelo e a matricula do veículo, assim como a sua posição, comunicando tudo isto ao servidor. De seguida após ser emparelhado com um passageiro é pedido que informe o servidor quando estiver no local de partida. Após o passageiro informar o servidor que entrou no veículo é solicitado ao condutor que indique quando chegar ao destino, sendo este o ponto final de execução do programa para o condutor.

1.2 Servidor Erlang

O servidor implementado em Erlang é inicializado pela chamada da função server que recebe como argumento a porta. É inicializado o listen socket, são gerados e registados 4 processos que vão gerir a memória partilhada, um processo registado como autenticador responsável por gerir os usernames e passwords dos utilizadores, que corre a função regUtente incializada com uma lista vazia. Um processo tipos que corre a mesma função que o processo anterior mas desta vez para gerir os tipos dos utilizadores(condutor ou passageiro) este processo é necessário para quando um utilizador faz o login se passar na autenticação sabermos se este é um condutor ou passageiro e enviá-lo para a função correspondente. Finalmente dois processos drivers e clients, que correm também a mesma função

que se destinam a guardar os passageiros e os condutores disponiveis para uma viagem, são dois processos pois um condutor vai procurar ao processo dos passageiros e vice-versa, de seguida é chamada a função acceptor passando como argumento o listen socket.
A função acceptor fica à espera de receber uma conexão ao listen socket, assim que receber lança um novo processo nesta mesma função para permitir que vários utilizadores se conectem em simultâneo, cada um no seu próprio processo.
Após é lançada a função gestor, esta função fica à espera de receber do cliente a palavra "registo" ou "login" sendo que caso receba registo segue para a função reg, seguindo para a função login caso contrário.
A função login, recebe um Username e uma password, pede ao autenticador que lhe envie o tuplo correspondente aquele username, se a password digitada pelo utilizador for igual à recebida do autenticador, então este pede o tipo do user ao tipos e chama a função correspondente, se não corresponderem as passwords então a execução termina. A função reg fica à espera de receber do cliente o username e a password, comunicando estes dados ao processo autenticador de forma a estes serem guardados e acessiveis a todos os processos. Posteriormente fica à espera de receber uma mensagem, caso essa mensagem seja condutor, recebe o modelo e a matricula e chama a função viagem, se for passageiro chama a função viagem2.
A função viagem2 recebe as posições de origem e destino, faz o calculo da distancia de Manhattan entre a origem e o destino para com base nessa distância calcular o custo da viagem tomou-se aqui como preço 2 euros por unidade, ou seja se a distância dá 10, o preço será 20 euros. De seguida é comunicado ao processo clients o Pid do processo e a sua posição de origem, é requisitado ao drivers que envie a lista de condutores disponiveis e é chamada a função search_drivers responsável por encontrar o condutor mais proximo do local de partida do passageiro.
A função viagem destinada aos condutores recebe do cliente a posição onde se encontra, comunica essa posição ao processo drivers e chama a função loop_condutores.
A função search_drivers recebe como argumentos o socket, a lista de condutores disponiveis, o Pid inicialmente é zero pois aqui é onde vamos guardar o Pid do condutor mais próximo,a posição de origem e destino, uma distância inicialmente é um numero grande pois é suposto esta ir sendo atualizada com a distância mais baixa a que um condutor se encontra e por fim o preço calculado na função viagem2.
Esta função testa inicialmente se a lista de condutores é vazia e o Pid é 0, se for este o caso é sinal que não há condutores disponiveis logo ela chama-se recursivamente até haver alguém disponivel. Se a lista for vazia mas o Pid diferente de 0, quer dizer que a lista já foi toda percorrida portanto está encontrado o condutor mais próximo, por isso calcula o tempo de chegada e comunica ao Pid do processo em questão que emparelhou com ele informando-o do seu Pid, da origem e destino da viagem e o preço. Caso falhe as duas condições anteriores é sinal que ainda está a percorrer a lista, então calcula a distância entre o local de partida e o local onde se encontra o utilizador que está à cabeça da lista, se essa distância for menor que a que está guardada em Dist, é chamada recursi-

vamente a função com a nova distância e o Pid do condutor correspondente e é retirado este condutor da lista. Caso não seja menor apenas retiramos o condutor da lista e chama-se a função recursivamente com os mesmos parametros que possuía anteriormente.
A função emparelhado é a função para onde é direconado o passageiro depois de emparelhar com um condutor, esta fica a espera de receber um going para informar o utilizador que um carro se encontra a caminho, o tempo esperado e o custo da viagem. Pode receber do utilizador um "cancelar" e nesse caso comunica ao condutor o cancelamento da viagem, este vai lhe responder o preço que tem de pagar que pode ser 0 ou metade do valor da viagem, pode receber do condutor um onSite que é sinal que o condutor chegou ao local de partida ou pode ainda receber do utilizador um "entrei" que quer dizer que o utilizador já entrou no veículo e a partir deste momento já não pode cancelar a sua viagem, logo é chamada a função viagemEmCurso que apenas recebe informação por parte do condutor assim que chegar ao local de destino de forma a este informar o passageiro que chegou ao destino.
O loop_condutores é a função "análoga" ao emparelhado mas para os condutores, ou seja enquanto não emparelham e depois ficam nesta função, aqui podem receber um lock com uma posição inicial e final e um preço, ao receber isto é sinal que emparelharam, então informam o utilizador para se dirigir para o local de partida, mandam para o passageiro respetivo um going, já referido atrás e lançam um processo clock inicializado a 0, este processo é um relógio que conta até um minuto que é o tempo que é possivel cancelar sem custos, de seguida chama-se recursivamente. Pode receber também um updateTime do processo clock, que é sinal que o minuto passou, logo atualiza o preço a pagar para metade do custo da viagem, de seguida têmos que pode receber um cancel do passageiro para cancelar a viagem, aqui ele informa o condutor que a viagem foi cancelada e o passageiro do valor a pagar.Por fim pode receber do utilizador um "cheguei" para indicar que está no local de partida e um destino para informar que chegou ao destino final.
A função condutores é responsável pela gestão de memória, como já foi referido esta função é chamada pelos processos drivers e clients, caso receba um Pid e um tuplo posição esta armezena-os, se receber um remove e um Pid, procura o Pid e remove o tuplo da lista, se receber um getConds e um Pid envia para esse Pid a lista de utilizadores que possui.
A regUtente é chamada pelos processos autenticador e tipos, é semelhante à função anterior ao receber um register e um Data, que aqui será um tuplo composto ou pelo username e a password no caso do autenticador, ou um username e um tipo no caso do tipos, sendo as outras duas possibilidades iguais às da função condutores.

Chapter 2

Conclusão

Penso que no geral o objetivo do trabalho prático foi conseguido, apesar de alguns problemas, regra geral as funcionalidades pedidas foram implementadas, este trabalho teve um grau de dificuldade elevado essencialmente por duas razões, a primeira por ter realizado o trabalho individualmente foi complicado gerir um trabalho de uma dimensão considerável como este, sozinho e também por até aqui desconhecer completamente a linguagem Erlang, isso dificultou a tarefa pois tive de ir aprendendo enquanto fazia o que numa primeira fase se tornou bastante penoso, penso que de todo o tempo que tivemos para realizar o trabalho três quartos do tempo foi de progressão lenta sendo que grande parte do trabalho foi feito no restante um terço, numa altura em que já compreendia melhor a linguagem devo dizer também que efetivamente aprendi bastante com este trabalho prático especialmente como já foi referido, sobre o Erlang.