Universidade do Minho
Escola de Engenharia

Servidor de recolha de estimativas

Programação Concorrente

Marco Barbosa
A81428

Tomás Sousa
A81411

June 1, 2020

Contents

Chapter 1

Problema

O trabalho proposto, na disciplina de Programação Concorrente foi implementado em Java e consiste no desenvolvimento de um servidor que permite recolher estimativas da proporção de infectados numa pandemia. Para tal estabelecemos uma comunicação orientada à linha onde cada cliente tem a possibilidade de se registar ou de se autenticar para assim estabelecer a ligação ao servidor. De seguida, o servidor envia o numero de casos anteriormente reportados pelo cliente, se for esse o caso, mas também envia a estimativa global a todos os clientes conectados. Ficando à espera do novo número de casos do cliente, quando esta for atualizada volta a enviar a nova estimativa a todos os clientes ligados. Posteriormente implementamos uma interface gráfica, GUI, par que fosse mais fácil e mais interativo para o cliente comunicar com o servidor. Finalmente foi implementado uma prova de conceito onde introduzimos a existência de países permitindo ao cliente reportar casos em mais do que um país.

Chapter 2

Solução

2.1 Protocolo de comunicação

O protocolo de comunicação é caracterizado por 7 pedidos que o cliente pode fazer ao servidor.

Check o cliente envia *ck* e um *username*, separados por espaço. O servidor responde com um acknowledge(*ack*), ou com um erro que indica a indisponibilidade do *username*.

Create o cliente envia *cr* seguido do *username* e da *password*, separados por espaços. O servidor reponde com um acknowledge.

Login o cliente envia *lgi* seguido do *username* e da *password*, separados por espaços. O servidor responde com um acknowledge, ou indicando que houve um erro na verificação do *username*, ou da *password*.

View o cliente envia *vw* seguido da sigla correspondente ao país("pt":Portugal, "es":Espanha, "it":Itália, "cn":China), separados por espaço. O servidor responde com um ackowledge seguido do número de casos já reportados pelo utilizador, separados por espaços.

Update o cliente envia *up* seguido do número de casos que o utilizador reportou, separados por espaço. O servidor responde com um acknowledge.

Logout o cliente envia *lgo*. O servidor responde com um acknowledge.

Remove o cliente envia *rm* seguido do *username* e da *password*, separados por espaços. O servidor responde com um acknowledge, ou indicando um erro na verificação da *password*.

Nos casos onde o acknowledge(*ack*) não devolve informação útil, este é seguido da sigla que designa o comando recebido, separados por espaços.

As mensagens de erro são da forma: *err* seguido da causa do erro, separados por espaço.
As atualizações tem como prefixo *est* ou *cest*, para a estimativa global ou do país respetivamente.

2.2 Interface Grafica

A interface gráfica é a parte do programa que permite ao utilizador interagir com o servidor sem ter de recorrer a um programa cliente como o "nc". A interface gráfica foi implementada recorrendo a uma GUI widget toolkit do Java chamada *Swing*. De forma a permitir o envio de pedidos para o servidor todas as interfaces implementam uma instancia da classe Coronita-Client-Account. Na implementação adotada o cliente tem acesso a três ambientes gráficos, de interação restrita:

1. Ambiente do Login;
2. Ambiente do Sign in;
3. Ambiente da Aplicação;

Login Aqui o cliente tem a possibilidade de introduzir as credenciais de autenticação, caso as tenha ou de se registar sendo redirecionado para a segunda interface. A autenticação passa por introduzir um Utilizador e a palavra passe, que serão verificados recorrendo ao método *authenticate* caso não seja lançada nenhuma excepção será definido o pais que o utilizador selecionou através do método *setCountry*. Caso o método lance uma excepção esta será apanhada aqui e o utilizador será alertado ou por introduzir um nome de utilizador invalido, *InvalidAccount* ou por introduzir uma palavra passe invalida *MismatchPassException*. Por fim será iniciada uma novo ambiente da aplicação.

Sign in Aqui o cliente tem de introduzir novas credenciais de modo a poder prosseguir. Após introduzidas será feita uma verificação do nome de Utilizador recorrendo ao método *checkUsername*, que poderá lançar a excepção *InvalidUsername* levando a um aviso e a uma sugestão de um novo nome. De seguida será feita um registo seguida de uma autenticação de modo a poder inicializar o cliente. Em caso de insucesso poderão ser apanhadas exceções correspondente a palavra passe invalida ou diferente, *PasswordException* e *MismatchPassException* respetivamente. Por fim será definido o país que o utilizador selecionou e será criado um novo ambiente correspondente à aplicação.

Aplicação A interface correspondente à aplicação oferece ao utilizador a possibilidade de atualizar o numero de casos conhecidos, de alterar o pais onde quer reportar casos, de sair da aplicação e ainda de remover a sua conta do servidor. A interface estabelece ainda uma visualização

numérica e gráfica, apresentando um gráfico circular, a estimativa de casos global e do país selecionado e o número de casos reportados pelo Utilizador.

2.3 Cliente

O Cliente efetua pedidos ao servidor de acordo com as ações da GUI, e atualiza-a de acordo com as resposta do servidor. As classes que constituem este Cliente têm como principais objetivos estabelecer uma conexão bidirecional ao servidor e implementar o protocolo de comunicação definido. Para tal este Cliente encontra-se dividido em quatro classes:

1. Coronita-Client-Account
2. Client-Server
3. Middleman → classe que cria uma thread que espera pelas respostas provenientes do servidor e as encaminha;
4. Bag → classe que lida com as respostas vindas do servidor para o qual o Middleman não tem acção.

2.3.1 Coronita-Client-Account

A classe Coronita-Client-Account, cria uma instância da classe Client-Server à qual necessita de especificar o porto e o ip onde será feita a conexão. Invoca também os métodos presentes na classe Coronita-Server fazendo ainda a verificação da palavra-passe e da validade do número de casos introduzido.

Verificar palavra-passe - *isValid* Verifica se ao invocar o método *registerAccount*(que envia um pedido de registo de conta ao servidor) a palavra-passe enviada cumpre com os requisitos estipulados, sendo estes: Mais que 3 caracteres e menos de 8 e que pelo menos um dos caracteres tem de ser um numero de 0-9;

Verificar numero de casos -*isValidNum* Verifica se o numero de casos introduzido é valido, para tal tem de cumprir com os requisitos, sendo estes: Numero inteiro de 0-150, não podendo conter outro tipo de caracteres nem vazio.

2.3.2 Client-Server

A clase Client-Server, inicia um novo socket no porto/ip definidos. Cria uma nova instância da classe *PrintWriter* que permite establecer um canal de escrita. E implementa ainda uma nova instância da classe Midleman e da classe Bag, que permitem estabelecer um canal de leitura. Garantindo assim que a comunicação é Threadsafe.

Verificar Nome de Utilizador - *checkUsername* Faz um pedido de **check** ao servidor e em caso de erro irá lançar a excepção *InvalidUsername*;

Registar Conta - *registerAccount* Faz um pedido de **Create** ao servidor, após terem sido asseguradas todas as condições de sucesso para o pedido.

Autenticar Conta - *authenticate* Faz um pedido de **Login** ao servidor e em caso de erro irá lançar uma das duas excepções possíveis *InvalidAccount* ou *MismatchPassException*;

Definir País - *setCountry* Faz um pedido de **View** ao servidor;

Atualizar Estimativa - *updateEstimate* Faz um pedido de **Update**;

Log out - *logout* Faz um pedido de **Logout** ao servidor;

Remover Conta - *removeAccount* Faz um pedido de **Remove** ao servidor e em caso de erro irá lançar a excepção *InvalidAccount*;

2.3.3 Bag

Cada objeto da classe Bag contém uma String que corresponde à resposta que o cliente espera do servidor. Oferece um método bloqueaste que espera que a mensagem seja atualizada e outro que atualiza a mensagem.

2.3.4 Middleman

Uma instância desta classe recebe a instancia de Bag partilhada, os componentes da GUI que devem ser atualizados com a informação das notificações assíncronas, e uma instância de BufferedReader que permite receber as mensagens do servidor.
O código do método run vai lendo as linhas enviadas pelo servidor e atualizando a GUI ou colocando essas mensagens na instância de Bag, de acordo com o conteúdo.
Esta classe disponibiliza métodos para:

- Iniciar uma thread que corre o código do método run.
- Fazer join com essa thread.

2.4 Interface

A interface define a assinatura dos métodos que realizam os procedimentos adequados a cada uma das ações definidas no protocolo de comunicação.
Estes métodos devem ser implementados de forma independente pelo servidor e pelo cliente. Ao cliente compete definir como fará os pedidos e como processará as respostas do servidor. Ao servidor compete definir como processar os pedidos recebidos e como devolver as respostas adequadas.

2.5 Servidor

O servidor cria uma instância da classe Accounts onde serão guardadas todas as contas registadas no servidor e um objeto da classe Estimates que contém as estimativas mais recentes, ambos serão partilhados por todos os workers. O servidor fica então à espera de conexões. Para cada nova conexão é criada uma thread worker que lida com a ligação a esse cliente.

2.5.1 Worker

A classe Worker implementa a interface Runnable, permitindo por isso criar uma thread que executa o código definido no seu método run. Uma instancia da classe Worker recebe as *accounts*(objeto da classe Accounts), e as *estimates*(objeto da classe Estimates) e recebe ainda o socket onde está conectado o cliente, que este irá servir.
A thread, criada a partir do worker, irá ficar à escuta no input stream do socket de mensagens do cliente, respondendo de forma adequada aos comandos recebidos. Irá ainda criar uma thread usando um objeto da classe Writer, para enviar notificações assíncronas ao cliente, após este escolher um país.
O servidor aceita os seguintes comandos, definidos pelo protocolo de comunicação:

Check verifica em *accounts*, se existe alguma *account* correspondente ao *username* especificado.

Create adiciona a *accounts* uma instancia da classe *Account* com o username e a password especificados e inicializa os casos reportados, indicando que não foram ainda reportados casos conhecidos em nenhum país.

Login verifica se existe alguma *account* correspondente ao username, e em caso afirmativo compara a password fornecida com a password associada à conta. Se ambas as condições se verificarem é enviada uma mensagem a reconhecer o sucesso da operação, caso contrário é enviada uma mensagem que detalha o erro ocorrido.

View se já tiver sido escolhido um país, a thread responsável pelas notificações assíncronas é parada. Em seguida, invocamos o método de *writer* que inicia uma nova thread, que envia mensagens para o *idCliente* que está autenticado, sobre o país escolhido(*currentCountry*).

Update atualiza o número de casos reportados para o *currentCountry*, considerando a primeira atualização como um *report* e as atualizações seguintes como correções a esse report. [1]

[1] Isto significa que na atualização inicial é feita uma média pesada entre a estimativa inicial, e_{ini}, e a estimativa previamente calculada pelo servidor, e_{old}, e a estimativa atual usando $e_{old} \cdot \frac{reports-1}{reports} + \frac{e_{ini}}{reports}$, nas atualizações seguintes é usado $e_{old} + \frac{e_{novo}-e_{ini}}{reports}$

Logout a thread responsável pelo envio assíncrono das atualizações é parada, e é enviada uma mensagem ao cliente a confirmar o sucesso da operação.

Remove verifica a validade do *username* e da *password*, em caso de sucesso remove a conta e envia uma mensagem a confirmar a remoção. Caso contrário, envia uma mensagem com o erro correspondente.

2.5.2 Writer

Tal como a classe Worker, também esta implementa a interface Runnable.
Uma instancia da classe recebe apenas as *estimates* e uma instância da classe SafePrint, que lhe permite enviar mensagem ao cliente.
Sempre que as *estimates* são atualizadas, a thread criada envia uma mensagem ao cliente com a atualização.
Esta classe disponibiliza métodos para:

- Iniciar uma thread que recebe o *idCliente* e um *country*, e envia mensagens sobre esse país ao cliente.
- Indicar à thread que, assim que possível, deve parar.
- Fazer join com essa thread.

2.5.3 Estruturas de dados

Estimates

Cada objeto da classe Estimates contém várias instâncias da classe Estimate, correspondentes a cada país, e uma estimativa global(*globalEstimate*).
Esta é a média aritmética das estimativas presentes em cada uma dessas instâncias.
Esta classe oferece métodos para:

- Esperar por um update
- Recolher o valor da estimativa mais recente
- Fazer update da estimativa de um país e atualizar a *globalEstimate*
- Fazer trigger a uma thread que esteja à espera na variável de condição *update*.

Estimate

Cada objeto da classe Estimate contém uma estimativa*estimate* e o número total de *reports*. Oferece um método para ler a *estimate*, e dois para fazer update da mesma. Oferece ainda um método para verificar se um cliente já leu a *estimate* atual.

Account

Cada objeto da classe Account contém uma password e o número de casos reportados para cada país, -1 se não tiver sido feito nenhum report. Contém ainda o país atual onde estão a ser reportados os casos.
Disponibiliza métodos para:

- Fazer get ou set da *password*.
- Fazer get ou set do *currentCountry*.
- Fazer get ou set dos *cases* associados ao *currentCountry*.
- Verificar se um utilizador já reportou algum caso num país.

Accounts

Cada objeto da classe Accounts contém um HashMap que associa a cada nome de utilizador, um objeto da classe Account.
Disponibiliza métodos para:

- Verificar se já existe alguma *account* associada ao *id* recebido.
- Comparar uma *password* com a password da conta associada ao *id* fornecido, levantando exceções caso não exista conta associada ao *username* ou caso as passwords não coincidam.
- Adicionar uma nova entrada ao HashMap que faz corresponder ao *id* uma conta com a *password* indicada e sem casos reportados para qualquer país.
- Alterar país atual(*currentCountry*) onde a *account* associada ao *id* está a reportar casos. Retornando o número de casos reportados anteriormente nesse país.
- Verificar já houve algum report num dado país(*country*), feito pela *account* associada a *id*.
- Alterar o número de casos reportados num dado país pela *account* associada a *id*.
- Remover conta associada a um *username*.