Universidade do Minho
Escola de Engenharia

Sistemas Distribuídos

Grupo 10

João Pedro da Santa Guedes A89588
Luís Pedro Oliveira de Castro Vieira A89601
Carlos Miguel Luzia Carvalho A89605
Bárbara Ferreira Teixeira A89610


A89588


A89601


A89605


A89610

25 de janeiro de 2021

Índice

1 Introdução **1**

2 Descrição das Classes Implementadas **1**
2.1 Classes Relacionadas com o Cliente 1
2.1.1 Client 1
2.1.2 ClientDrawer 1
2.1.3 ClientReader 1
2.1.4 ClientStatus 2
2.2 Classes Relacionadas com o Servidor 2
2.2.1 Servidor 2
2.2.2 ClientHandler 2
2.2.3 HistoricParser 2
2.2.4 NotificationHandler 3
2.2.5 EstadoPartilhado 3
2.2.6 User 3

3 Descrição das operações **3**
3.1 LOGIN 3
3.2 REGISTER 3
3.3 LOGOUT 4
3.4 WRITEMAP 4
3.5 CHANGEZONE 4
3.6 CONSULTZONE 4
3.7 CONSULTMAP 4
3.8 INFORMSTATE 4
3.9 SERVER 5
3.10 CONSULTZONENOTIFY 5
3.11 DOWNLOADPLZ 5

4 Observação **5**

5 Conclusão **5**

1 Introdução

Este trabalho prático foi realizado no âmbito da Unidade Curricular de Sistemas Distribuídos e teve como objetivo o desenvolvimento e implementação de uma plataforma inspirada no problema do rastreio de contactos e deteção de concentração de pessoas sob a forma de cliente/servidor em Java utilizando sockets e threads, com um funcionamento similar à APP StayAway COVID. Ora este Sistema segundo o enunciado deve permitir ao utilizador autentificar-se/registar-se estabelecendo sempre uma conexão com o servidor, deve também ser possivel ao utilizador manter o servidor informado da sua localização e saber quantas pessoas se encontram numa dada localização e ser avisado caso uma dada localização solicitada pelo mesmo se encontre vazia, devendo ainda permitir que um utilizador comunique com o servidor a informar que está infetado.

Foi nos ainda proposto adicionalmente que fosse possível notificar todos os utilizadores potencialmente contagiados por um doente, que tenham estado na mesma localização que ele e ainda, ser possìvel para um utilizador especial, ver um mapa onde se indique quantos utilizadores quer infetados quer não infetados, visitaram uma determinada localização.

2 Descrição das Classes Implementadas

2.1 Classes Relacionadas com o Cliente

2.1.1 Client

Classe onde se inicia a ligação Cliente/Servidor e se dá o início à execução de duas Threads, uma associada a um objeto da classe ClientDrawer e outra associada a um objeto da classe ClientReader.

2.1.2 ClientDrawer

Classe responsável pela implementação da interface apresentada ao Utilizador, assim como o envio dos inputs do utilizador ao servidor. Antes de estes inputs serem enviados ao servidor, esta classe é responsável por concatenar os inputs do utilizador numa mensagem que o servidor é capaz de reconhecer. As mensagens são enviadas a parir do método **server_request** onde, de seguida, a thread fica à espera duma resposta do servidor. O método que bloqueia a thread encontra-se na classe ClientStatus, que será descrita posteriormente. É o objeto da classe ClientReader que imprime a resposta do servidor e acorda a thread correspondente do tipo ClientDrawer, que atualiza o estado da interface.

2.1.3 ClientReader

Classe responsável pela leitura das respostas do Servidor, imprimindo-as no stdout. Esta classe está, como descrito anteriormente, também responsável por sinalizar/acordar a Thread correspondente ao ClientDrawer, caso esta se encontre adormecida.

2.1.4 ClientStatus

Esta classe permite a "comunicação" entre ClientDrawer e ClientReader, permitindo saber se o cliente se encontra à espera de resposta do servidor, se o *Login* foi bem sucedido, se este efetuou *Logout*, se o utilizador é *Especial* e se o Utilizador está infetado.

2.2 Classes Relacionadas com o Servidor

2.2.1 Servidor

Classe responsável pela criação do **EstadoPartilhado**, do **NotificationHandler** e dos sockets a usar. O Servidor criado é multi-threaded, estando cada thread associada a um objeto da classe **ClientHandler** por cada conexão ao Servidor.

2.2.2 ClientHandler

Esta classe é responsável por receber e processar as mensagens enviadas do cliente para o servidor, sendo que a cada conexão realizada é associado um objeto da mesma. Uma vez efetuado o login com sucesso é guardado no objeto informação relativa ao utilizador associado.
As mensagens recebidas podem ser as seguintes:

- LOGIN : para efetuar o login de um utilizador;
- REGISTER : para registar um utilizador;
- LOGOUT : para terminar sessão de um utilizador;
- WRITEMAP : escrever o mapa com todas as suas zonas;
- CHANGEZONE : regista a alteração de zona por parte de um utilizador;
- CONSULTZONE : consulta o número de utilizadores numa dada zona;
- CONSULTMAP : informa o utilizador da existência de utilizadores nas várias zonas;
- INFORMSTATE : registar um utilizador como infetado ou não;
- SERVER : informa utilizador que esteve em perigo de contágio;
- CONSULTZONENOTIFY : adicionar o utilizador à lista de utilizadores a notificar por zona;
- DOWNLOADPLZ : donwload de um mapa com as estatísticas dos utilizadores;

2.2.3 HistoricParser

Esta é a classe responsável pela criação e atualização de um ficheiro de logs "historico.txt". Este ficheiro é necessário para uma das funcionalidades adicionais, onde vai sendo guardado, em cada linha, o nome do utilizador, as várias zonas em que esteve e finalmente um booleano que corresponde ao seu estado de infeção, da forma : "**user**;**zonas**;**isInfected**".

2.2.4 NotificationHandler

Classe responsável por notificar os utilizadores que devem ser notificados. Esta guarda o socket correspondente a cada utilizador, para que futuramente possa comunicar. Esta é a classe que atua quando um cliente fica infetado ou quando uma zona fica vazia. Caso o cliente não se encontre on-line, ele deixa uma "mensagem".

2.2.5 EstadoPartilhado

Esta é uma das, senão a classe mais importante visto que é aqui que assentam todas as informações relativas ao estado do programa. É nesta classe que se situam todos os utilizadores registados, a informação sobre eles, a informação sobre o mapa e quais os clientes que pretendem receber notificações.

2.2.6 User

Classe que representa um utilizador, contendo assim o seu **username**, **password**, **posição geográfica**, **estado de infeção**, juntamente com uma **lista de utilizadores** com quem esteve em contacto e uma **lista de mensagens** entregues pelo Servidor. Além disso, contém também um **booleano** que representa se o Cliente é ou não especial. Para simplificação, neste trabalho decidimos que todos os clientes cujo username começa por um dígito, é considerado especial.

3 Descrição das operações

Anteriormente, evidenciámos as 10 operações correspondentes aos tipos de mensagens que o servidor recebe no **ClientHandler** e que vamos aprofundar de seguida.

3.1 LOGIN

Para efetuar login no servidor, o cliente deve enviar uma mensagem com o seguinte formato: **LOGIN;username;password**. Uma vez decomposta a mensagem através do método *commandLogin()*, é executado o método *logIn()* que vai verificar se o username e a password existem. Além disso, em caso afirmativo, é enviado para o cliente informações sobre o seu estado (infetado/ não infetado), se tem ou não mensagens e se é um utilizador especial ou não. Se o username e a password não existirem, o login é negado.

3.2 REGISTER

Para efetuar o registo no servidor, o cliente deve enviar uma mensagem com o seguinte formato: **SIGN;username;password**. O método utilizado para registar um cliente é o método **registerClient()** que vai garantir que o *username* que estão a tentar registar ainda não existe, e que a zona indicada é válida. Se isto se verificar, é registado o novo cliente, ou seja, o username e password ficam guardados no Map que contém a informação dos clientes.

3.3 LOGOUT

Com o envio deste comando, dá se a execução do método *commandLogout()*, sendo que o servidor responde com a mensagem **LOGGED OUT**. O ClientReader ao receber o sinal, vai alterar o status de login para falso.

3.4 WRITEMAP

Esta operação é executada pelo método *writeMap()*, utilizada no momento em que o mapa necessita de ser apresentado para que o utilizador escolha uma zona, por exemplo, no momento em que um novo cliente se regista e tem que indicar a zona onde se encontra, como referido na operacão *REGISTER*, ou quando o utilizador pretende alterar a sua localização.

3.5 CHANGEZONE

O cliente quando pretende atualizar a sua localização envia uma mensagem com o formato: **CHANGEZONE;localização**. O método utilizado para alterar a zona é o método *changeZone()* que vai alterar a localização do utilizador em causa e, caso a localização inicial do utilizador deixe de ser habitada, o servidor notifica os utilizadores que requisitaram essa notificação.

3.6 CONSULTZONE

Quando um cliente pretende consultar uma determinada zona deverá enviar uma mensagem com o seguinte formato : **CONSULTZONE;zona**. O método executado será *commandConsultZone()* que utiliza o método *zoneConsult()* para saber quantas pessoas se encontram na zona pretendida e, de seguida, é enviada uma mensagem para o cliente com a informação pretendida.

3.7 CONSULTMAP

Para o mapa ser consultado, o cliente, que deverá ser um utilizador especial, deve enviar uma mensagem com o seguinte formato: **CONSULTMAP**. O método que executa esta operação é o método *mapConsult()*, que imprime o mapa dividido nas suas várias zonas, e a cada zona está relacionado o número de pessoas que se encontra nessa zona nesse preciso momento.

3.8 INFORMSTATE

Esta operação é executada pelo método *commandInformState()* que é chamado quando o servidor recebe uma mensagem do tipo **INFORMSTATE;boolean** e, de acordo com esse boolean, o estado do utilizador é atualizado para infetado, notificando todos os users que se cruzaram com o doente, ou caso o boolean seja falso, atualiza a lista de proximidade.

3.9 SERVER

Quando é recebida uma mensagem do tipo **SERVER;notify**, o servidor envia uma mensagem aos utilizadores que estiveram em contacto com o utilizador infetado, a avisar que este esteve em contacto com uma pessoa infetada.

3.10 CONSULTZONENOTIFY

Quando é recebida uma mensagem do tipo **CONSULTZONENOTIFY;zona**, o Servidor vai adicionar a um Map a informação que o cliente pretende receber notificação para a zona dada para que, mais tarde, consiga notificar todos aqueles que pediram a notificação. No final, o Servidor comunica com o Cliente se a ação foi feita com sucesso.

3.11 DOWNLOADPLZ

Quando é solicitado, por um utilizador especial, o download de um Mapa de estatística, o Servidor recorre ao método da classe HistoricParser "*statisticsMapFile()*" onde percorre o ficheiro "historico.txt" e vai preenchendo um array com as informações até ao momento do pedido. No final, toda a informação é agregada e escrita num ficheiro com um nome gerado aleatoriamente. Esse método acaba por devolver o path de onde o ficheiro se situa e o utilizador é informado da existência desse mesmo ficheiro.

4 Observação

Como o nosso mapa é baseado nas letras do alfabeto (maiúsculas), o tamanho máximo, devido a esta limitação, é 25 (5x5).

5 Conclusão

Tendo diversos desafios à frente, e implícitos no projeto, como corridas, concorrência e hipotéticos deadlocks, problemas comuns e ocorrentes no âmbito dos sistemas distribuídos, o trabalho realizado e implementado não só cumpre todos os requisitos básicos do enunciado, bem como os requisitos adicionais propostos no mesmo, como também é capaz de lidar de forma correta e competente com estes problemas comuns acima mencionados.

Isto visto que, apesar de haver concorrência na sua execução as regiões críticas são acedidas por uma thread, não prejudicando a integridade e consistência dos dados necessários ao bom funcionamento do trabalho.

Assim, como esperávamos, o projeto funciona como requisitado segundo o enunciado, tendo sempre em conta os conceitos e procedimentos adquiridos ao longo da UC Sistemas Distribuídos.