Universidade do Minho
Mestrado Integrado em Engenharia Informática

SISTEMAS DISTRIBUÍDOS
ALARME COVID-19
GRUPO 11

André Carvalho da Cunha Martins a89586
António Jorge Nande Rodrigues A89585
José Pedro Castro Ferreira A89572
Rui Emanuel Gomes Vieira A89564

25 de janeiro de 2021


A89586


A89585


A89572


A89564

Conteúdo

1 Introdução

O trabalho prático desta unidade curricular desafiou-nos a construir um programa para simular uma aplicação de rastreio de contactos de infeções por COVID-19. A aplicação tem por bases o registo de utilizadores (podendo estes terem, ou não, uma funcionalidade adicional) e posterioor análise dos seus contactos. Cada utilizador regista-se numa posição de um mapa N por N. Através do histórico das posições de cada utilizador, é possível rastrear os contactos do mesmos e avisar se estes estão em perigo de ter contraído uma infeção, uma vez que estiveram em contacto com outro utilizador sinalizado como infetado. É também possível a cada utilizador saber quantas pessoas estão, num dado momento, numa certa localização do mapa, bem como serem avisados quando uma certa localização de encontra vazia. Finalmente, uma vez que um utilizador se declare infetado com COVID-19, já não pode mais efetuar qualquer uma das funcionalidades, ficando, assim, bloqueado.
Esta aplicação foi implementada em Java, com os clientes a poderem interagir com um servidor multi-threaded, recorrendo à comunicação via sockets TCP.

2 Implementação

2.1 Package Client

2.1.1 Client

Esta classe representa o executável que qualquer utilizador da aplicação irá utilizar para conseguir usufruir das funcionalidades da nossa aplicação.

2.1.2 Menu

Esta classe tem os diferentes menus que a UI da nossa aplicação apresenta aos seus utilizadores. Existem 3 tipos de menus. O menu de login, comum a todos os utilizadores. O ecrã inicial de um utilizador comum e o ecrã inicial de um delegado de saúde. Tem, també, o aviso que aparece assim que um utilizador se marca como infetado por COVID-19.

2.1.3 ThreadClientInput

Esta classe é responsável por transmitir a informação pretendida entre cada cliente e o servidor.

2.1.4 ThreadClientOutput

Esta classe recebe a informação do servidor que é direcionada ao cliente

2.2 Package Server

2.2.1 Server

Classe onde está implementado o servidor, mantendo-o ligado e tornando toda a aplicação funcional.

2.2.2 ThreadServerRead

Esta classe interpreta os pedidos do cliente, excutando a funcionalidade pretendida

2.2.3 ThreadServerWrite

Esta classe transmite ao cliente a informação, após ser executada a funcionalidade que foi pedida

2.2.4 ServerBuffer

Esta classe é fulcral na comunicação entre cliente e servidor, enviando ao cliente as mensagens que cada ação despoleta no servidor

2.2.5 Utilizador

Esta classe representa um utilizador do programa. Cada utilizador tem uma localização e uma variável a indicar se se encontra, ou não, infetado. O utilizador pode verificar se uma posição está livre, quanta gente está numa posição e declarar-se como infetado. O delegado de saúde é um utilizador especial, possuindo a variável credencial diferente de 0, podendo imprimir o mapa de utilizadores. Possui, também, um histórico de localizações, que será necessário para rastrear os utilizadores que estiveram em contacto com ele

2.2.6 Localização

Esta classe representa uma posição no mapa N por N

2.2.7 ListUsers

Esta classe pode ser encarada como a base de dados da nossa aplicação. Possui todos os registos de utilizadores, um map com ServerBuffer utilizado para os registos, o mapa, representado por uma matriz de inteiros, dizendo quanta gente está em cada posição, dois Locks, um para as posições e outro para os utilizadores, acompanhados de duas Conditions, necessárias para as outras as 2 threads adicionais que criamos.

2.2.8 ThreadPosicaoLivre

Esta thread auxiliar foi criada para a funcionalidade 3. Quando um cliente faz o pedido de uma posição livre, esta thread é criada e fica em await(). Quando uma posição é atualizada, e se a posição ficou com 0 pessoas, envia um signalAll() para a todas as threads. Na thread, vai ser analisar se a posição pretendidade ficou vazia. Se ficou vazia, envia o sinal ao cliente correspondente. Caso não esteja livre, volta a ficar em await()

2.2.9 ThreadInfecao

Esta thread auxiliar foi criada para enviar o aviso de infeção aos utilizadores que estão em risco. Uma vez feito o login, esta thread é criada e fica em await(). Quando um cliente se declara como infetado, irá enviar um signalAll() a todas as threads. Na thread, será analisado se o utilizador cruzou com algum doente. Caso se tenha cruzado, irá receber um aviso de risco. Caso contrário, não irá acontecer nada e a thread voltará em ficar em await() até outro cliente ser dado com infetado

2.3 Packages Exceptions

Para fazer o tratamento de erros, criamos 4 Exceptions, sendo elas InvalidLocationException, InvalidLoginException, InvalidRegistrationException, UserInfetadoException.

3 Controlo de Concorrência

Um dos principais desafios deste trabalho prático é a gestão e controlo de concorrência. Uma vez que os utilizadores podem mudar de posição durante a execução do programa, e outro utilizador pode querer saber quantos utilizadores estão numa posição, por exemplo, é necessária a utilização de Locks. Sempre que o estado de um utilizador é alterado, procedíamos ao seu Lock, garantindo, assim, que nenhuma outra thread entrava nessa região crítica. Este controlo de concorrência também é efetuado na classe ServerBuffer, possuindo esta um Lock, que fica bloqueado enquanto um utilizador adiciona uma mensagem, não havendo, assim, conflitos entre os utilizadores. As variáveis de condição também desempenham um papel muito importante no funcionamento do nosso programa. Utilizando o exemplo das 2 Threads auxiliares que criamos, cada uma terá uma Condition (ThreadInfecao terá uma Condition associada a um userLock e ThreadPosicaoLivre associada a um posicaoLock) que irá ser responsável pela ativação, ou não, dessa thread, sinalizando, quando for o caso, os utilizadores em questão

4 Descrição das funcionalidades do programa desenvolvido

4.1 Login

Ao efetuar o login, o utilizador irá introduzir as suas credenciais (nome e password). Estas serão passadas ao servidor, que irá analisar a mensagem e devolver o respetivo utilizador. Se algum dos dados estiver incorreto, a mensagem de erro correspondente será transmitida, através do uso do ServerBuffer.

4.2 Registo

Ao efetuar o registo de um novo utilizador, este deverá inserir o seu nome, a sua password, a sua localização, respeitando os limites do mapa, e, caso se verifique, a sua credencial de delegado de saúde. A comunicação entre servidor e cliente funciona de forma análoga ao login

4.3 Número de pessoas por localização

Nesta funcionalidade, o utilizador indica a posição que pretende analisar, sendo essa localização passada ao servidor. O servidor irá ao mapa presente na classe ListUsers e irá devolver o número de utilizadores presentes na dada posição. É efetuado um lock, para não existir problemas de concorrência com outros utilizadores com o mesmo mapa

4.4 Altera localização de utilizador

Nesta funcionalidade, o utilizador irá alterar a sua posição atual para um nova posição. Ao atualizar a sua posição, a nova posição será adicionada ao seu histórico. Se a posição antida do utilizador ficou vazia, irá enviar um signalAll() para todas as ThreadPosicaoLivre que se encontram à espera, podendo, assim, notificar os utilizadores que uma dada posição se encontra livre

4.5 Notificar posição livre

Nesta funcionalidade, recorremos ao uso da thread ThreadPosicaoLivre. Assim que o utilizador indica a posição para a qual pretende ser avisado, esta thread é criada. Se a posição já se encontrar livre, o utilizador será, de imediato, avisado, não havendo a necessidade da criação da thread. Caso contrário, a thread será criada e ficará em espera até a dada posição ficar livre. Quando isso acontecer, o utilizador receberá um aviso, assíncrono, resultante de um signalAll() da funcionalidade descrita acima

4.6 Notificar Infeção

Sendo esta a funcionalidade mais complexa do programa, também recorremos ao uso de uma thread adicional, a ThreadInfecao. Esta thread é iniciada aquando do login de um utilizador, ficando em await() até esse utilizador ser sinalizado como contacto de risco de infeção de COVID-19. Quando um utilizador comunica infeção por COVID-19, fica bloqueado do acesso às restantes funcionalidades do programa, podendo, apenas, terminar sessão. A partir desse momento, será enviado um signalAll() a todas as ThreadInfecao que estão ativas. Nas threads, serão avaliados os contactos de risco de cada utilizador, recorrendo ao histórico de localizações que cada um possui. Caso algum utilizador tenha tido um contacto de risco, este será avisado que esteve em contacto com alguém infetado.

4.7 Apresentação do mapa de utilizadores

Esta funcionalidade é apenas restrita aos utilizadores que possuam uma credencial de delegado de saúde. Irá imprimir o mapa de utilizadores, dizendo quantos doentes e quantos utilizadores já passaram por uma dada posição. À semelhança da primeira funcionalidade descrita, também aqui foi necessário recorrer ao uso do userLock, para não existirem problemas de concorrência no acesso ao mapa.

5 Conclusão

Durante a realização deste trabalho prático, implementamos os conhecimentos adquiridos nas aulas práticas de Sistemas Distribuídos, nomeadamente o uso de Locks e Conditions, e a comunicação entre Cliente e Servidor, recorrendo a sockets TCP.
Consideramos que o nosso programa implementa todas as funcionalidades que foram propostas, havendo sempre o cuidado com o controlo de concorrência e com a possível existência de deadlocks.
Concluindo, a realização deste projeto ajudou a cimentar os conhecimentos adquiridos nas aulas práticas, utilizando um exemplo bem real, ainda que numa escala menor, mas que nos permitiu perceber e analisar a forma de comunicação entre cliente e servidor e aplicar todas as técnincas de controlo de concorrência aprendidas