Sistemas Distribuídos - Trabalho Prático

MIEI - Universidade do Minho

Sofia Santos - A89615 Ema Dias - A89518
Sara Queirós - A89491 Tânia Teixeira - A89613

25 de janeiro de 2021

1 Introdução

Este trabalho consistiu no desenvolvimento de um programa semelhante à aplicação *StayAway Covid*, em que vários utilizadores partilham a sua localização, e caso um deles fique infetado pode usar o programa para avisar outros utilizadores que tenham estado em contacto com este. Os utilizadores têm também a possibilidade de ver a ocupação de uma dada localização e de pedirem para ser avisados quando essa localização estiver vazia. Para além disso, podem existir ainda contas de administradores, que são capazes de ver a ocupação de todas as localizações no sistema, para além de saber quantos utilizadores infetados estiveram em cada localização.

O programa consiste num servidor que é capaz de comunicar com vários clientes concorrentemente e armazena informação como a posição dos clientes ou quais os clientes que estão infetados. Cada cliente comunica com o servidor através de sockets TCP, de forma síncrona e assíncrona. Para operações como iniciar sessão ou obter a ocupação de uma localização usamos comunicação síncrona. Deste modo, o cliente deve esperar até receber uma resposta do servidor de que, por exemplo, a sua sessão foi iniciada com sucesso antes de continuar a usar o programa. Por outro lado, a comunicação a um cliente de que este esteve em contacto com um infetado deve ser feito de forma assíncrona, de modo a que os clientes possam usar o programa normalmente sem terem que estar constantemente a aguardar por uma resposta do servidor.

O nosso trabalho possui todas as funcionalidades básicas e adicionais descritas no enunciado. Para além destas, possui ainda uma funcionalidade de serialização, ou seja, se o servidor for abaixo, este consegue armazenar informação relativa às contas dos utilizadores e às suas localizações, permitindo assim ter um sistema mais seguro e fiável.

Tal como sugerido no enunciado, o mapa do nosso programa consiste numa grelha $N \times N$, logo as diferentes localizações no mapa são identificadas por um par de inteiros, como $(1, 4)$ ou $(-7, 0)$.

2 Cliente

Um cliente é uma entidade capaz de interagir com o servidor. Cada cliente tem associado a si uma conta. Ao utilizar o programa pela primeira vez, é pedido ao cliente para criar uma conta ou para iniciar sessão, caso já tenha usado o programa antes.

Quando um cliente se regista, os seus dados são enviados para o servidor, que valida os mesmos, isto é, verifica se o endereço de email não está já em uso. Caso sejam validados, os dados do cliente são armazenados, para este poder posteriormente usar o programa sem precisar de se registar novamente. Por outro lado, ao iniciar sessão, o servidor verifica se o endereço de email introduzido existe na sua base de dados e se a palavra-passe introduzida corresponde à palavra-passe armazenada. Em ambos os casos, o servidor informa o cliente se o registo/início de sessão foi bem sucedido ou não, e em caso negativo indica ainda a razão pela qual a operação não foi possível.

Após se registar ou iniciar sessão, é pedido ao cliente para indicar a sua localização. Numa aplicação real, este processo seria automático, ou seja, seria possível, através do GPS do seu dispositivo, determinar automaticamente a localização do utilizador, após autorização do mesmo. Como o nosso programa é limitado nesse aspeto, e para ser mais simples testar o seu correto funcionamento, é pedido ao cliente para introduzir manualmente a sua localização. Enquanto utiliza o programa, o cliente pode alterar a sua localização a qualquer momento.

O cliente é capaz de saber a ocupação de uma certa localização, através de um simples pedido ao servidor. Se essa localização não estiver vazia, o cliente pode pedir para receber um alerta, que o informará quando esta localização estiver vazia. Esta funcionalidade usa um thread para funcionar. Quando um cliente pede para ser alertado, é criado um thread que fica à espera de uma mensagem do servidor a informá-lo de que a dada localização já se encontra livre. Após receber a dita mensagem, o thread mostra uma mensagem ao cliente e morre. Deste modo, os clientes podem continuar a usar o programa normalmente enquanto esperam que a localização fique livre, permitindo assim uma comunicação assíncrona entre estes e o servidor. Cada cliente pode criar N alertas, sendo que numa aplicação real esse valor de N teria que ser fixo, para evitar que o seu dispositivo fique sobrecarregado ou lento.

A outra funcionalidade que um cliente tem é indicar que está doente. Aqui, o cliente apenas envia uma mensagem ao servidor a indicar que está doente e o servidor trata de informar outros clientes que tenham estado em contacto com este. Após enviar a mensagem, o programa fecha do lado do cliente, pois este está doente e deverá ficar em casa até recuperar. Se o cliente voltar a entrar no programa, assume-se que este já recuperou.

Um cliente é informado que esteve em contacto com um doente através de um thread que é criado quando o cliente inicia sessão, e que aguarda por uma mensagem especial do servidor a informá-lo que o cliente associado a esse thread esteve na mesma localização que um cliente infetado. Este thread, tal como os threads de alarme, morre após mostrar ao cliente a mensagem que recebeu do servidor. Se entretanto o cliente sair do programa sem ter estado em contacto com nenhum doente, estes threads também são "mortos", através de uma mensagem do servidor.

Por último, existem as contas de administrador. Por convenção, uma conta de administrador é qualquer conta cujo endereço de email comece por "admin". Não é um sistema muito seguro, mas dentro do contexto do nosso projeto é suficiente. Estas contas, para além de tudo o que um cliente é capaz de fazer, conseguem ainda consultar um mapa, onde para cada localização é indicado quantos clientes saudáveis e doentes já passaram por ali.

3 Conexão

A conexão entre o cliente e o servidor é assegurada por duas classes, *Connection* e *Demultiplexer*.

A classe *Connection* é capaz de receber uma mensagem e enviá-la através do

socket a ela associado ou ainda de receber dados pelo socket e convertê-los numa mensagem que o cliente/servidor consiga ler. Por outras palavras, esta classe é um intermediário entre os clientes e o servidor, que trata de trocar mensagens entre os dois lados, usando para isso *DataInputStreams* e *DataOutputStreams*.

Existe ainda a classe *Demultiplexer*. Esta classe opera apenas do lado do cliente, entre a *Connection* e o cliente em si. Como um cliente pode ser vários threads, e estes threads podem ter que esperar que uma mensagem seja recebida, o *Demultiplexer*, através de métodos *await* e *signal*, assegura que estes threads fiquem a "dormir"enquanto a mensagem que pretendem receber não chega, recebendo um sinal para "acordar"quando a mesma chega.

4 Frame

Para simplificar o processo de enviar e receber mensagens, estas são representadas por *frames*, objetos de uma classe do mesmo nome. Um *frame* contém uma *tag*, um identificador do cliente que enviou a mensagem e a mensagem em si. A *tag* é um simples valor numérico. Na seguinte tabela podemos ver o que cada valor de *tag* representa:

Tag	Tipo de mensagem
0	Pedido de início de sessão ou resposta.
1	Pedido de registo ou resposta.
2	Pedido de mudança de localização.
3	Pedido de saber ocupação de localização ou resposta.
10	Pedido de mapa de localizações.
30	Pedido de aviso de localização livre ou resposta.
99	Pedido de indicar doença ou sair do programa.
112	Indicação de contacto com doente.

Figura 1: Tipo de mensagem associado a cada *tag*.

O identificador do cliente é utilizado pelo servidor para saber qual o cliente que enviou a mensagem respetiva. Nas mensagens enviadas pelo servidor, este valor é irrelevante, podendo ser uma *string* vazia.

Por último, a mensagem é armazenada num *array* de bytes, pois é mais eficiente usar formatos binários para enviar grandes quantidades de dados.

5 Servidor

O servidor tem como principal função manter um registo das localizações dos seus utilizadores, usando para isso a classe *Locations*, que armazena a posição atual e o histórico de posições de cada cliente. Para além desta classe, o servidor usa a classe *Accounts* para armazenar e consultar os dados relativos às

contas dos clientes. O uso destas classes permite uma maior abstração, algo bastante importante em Programação Orientada aos Objetos. Para além desta informação, o servidor armazena ainda informação relativa aos alarmes definidos pelos clientes, aos clientes que informaram estar doentes e aos clientes que estão com sessão iniciada.

Para evitar problemas de exclusão mútua, usamos locks nas classes *Locations*, *Accounts* e nas estruturas de dados do servidor. Os locks das localizações e das contas são *ReadWriteLocks*, já que estas classes contêm muitas operações de leitura e poucas de escrita. Desta forma, conseguimos melhorar o desempenho do programa.

Quando um cliente define um alarme, o servidor cria um thread responsável por esperar que a dada localização fique livre. Esse thread cria uma *condition* associada à localização, e sempre que um utilizador se move, se a localização de origem tiver uma ou mais *conditions* associadas a ela, é enviado um *signal* que acorda os threads relativos aos alarmes. Ao acordar, os threads verificam se a localização já está vazia. Em caso afirmativo, enviam uma mensagem ao cliente respetivo e morrem.

Da mesma forma, quando um cliente inicia sessão, é criado um thread que fica num estado de *await* até a *condition* associada à lista de utlizadores doentes receber um sinal. Quando um cliente avisa estar doente, o servidor envia um sinal a estes threads, que acordam e verificam se o cliente associado a si esteve em contacto com algum dos utilizadores na lista dos doentes. Em caso afirmativo, enviam uma mensagem ao cliente e morrem. De igual modo, se o cliente já tiver saído do programa entretando, estes threads morrem. Se o cliente voltar a iniciar sessão, este thread volta a ser criado, garantindo que um cliente é avisado que esteve em contacto com um doente, mesmo que não tivesse a usar o programa quando o utilizador doente enviou a sua mensagem ao servidor.

6 Conclusão

Este trabalho permitiu-nos consolidar os conhecimentos obtidos nas aulas de Sistemas Distribuídos de uma forma útil e interessante. O facto de aplicações como a *StayAway Covid* funcionarem de modo semelhante ao nosso projeto permite-nos perceber o quão importantes estes conceitos, como locks, sockets ou threads, são no mundo real, e que aquilo que aprendemos este semestre poderá ser-nos muito útil no nosso futuro académico e profissonal. Como fomos capazes de implementar com sucesso todas as funcionalidades pedidas, acreditamos ter um projeto bem conseguido.

Tal como foi referido no enunciado, não tivemos em conta o aspeto temporal, isto é, para saber se dois clientes estiveram em contacto apenas verificamos se estiveram na mesma localização, não verificamos se estiveram nessa localização ao mesmo tempo. Para incluir essa funcionalidade no nosso sistema, teríamos que, para além das localizações de cada cliente, armazenar também a data e hora a que os clientes chegam e saem de cada localização, e depois ao verificar se dois clientes estiveram em contacto, verificar também se existe uma interseção

nos intervalos de tempo em que estiveram na mesma localização. Da forma como o nosso sistema foi implementado, seria bastante simples implementar essa funcionalidade, caso tal fosse necessário.

Tendo em conta que estamos a trabalhar com uma linguagem orientada aos objetos, poderíamos ter usado uma arquitetura de camadas, como MVC, para implementar o sistema. Contudo, devido ao reduzido tamanho do projeto, considerámos que usar uma arquitetura de camadas iria tornar o projeto bastante mais complexo e que portanto não se justificava. Contudo, num projeto em larga escala, apesar de não ser necessário seria algo a considerar, para simplificar a manutenção do sistema.