Universidade do Minho
Escola de Engenharia

SISTEMAS DISTRIBUÍDOS

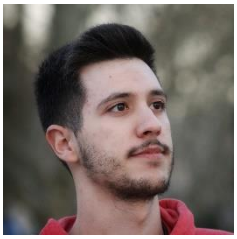
RELATÓRIO TRABALHO PRÁTICO

ALARME COVID

GRUPO 58

Luís Miguel Pinto A89506

Ana Luísa Carneiro A89533



Pedro Almeida Fernandes A89574

Ana Rita Peixoto A89612

ÍNDICE

INTRODUÇÃO

Neste trabalho prático era proposta a implementação de uma aplicação que tem como objetivo principal implementar um conjunto de requisitos semelhantes às funcionalidades da aplicação StayAway-Covid, com o uso de clientes e um servidor que comunicam entre si com sockets TCP-IP. Para a realização deste projeto foram utilizados vários conceitos aplicados nos guiões das aulas práticas e teóricas como o conceito de concorrência, exclusão mútua, secção crítica, serialização, middleware, notificação assíncrona, etiquetamento, entre outros.

SERVIDOR

O servidor tem como função processar pedidos provenientes de clientes. Nesse sentido, foi implementado um servidor ***Threaded-per-connection*** (tal como abordado no guião 8), isto é, por cada conexão com cada cliente existe um número limitado de threads. Foram escolhidas 10 *threads* de modo a realizar as diversas funcionalidades do programa sem que estes consumam muitos recursos. Com esta implementação promove-se a concorrência entre threads, ou seja, é possível o servidor estar a cumprir simultaneamente com pedidos de vários clientes e/ou do mesmo cliente.

De modo a suportar 2 tipos de clientes distintos, foi necessário considerar ***Listeners*** no servidor, ou seja, threads cuja funcionalidade é "escutar" o socket, e deste modo permitir o tratamento correto dos pedidos de cada cliente. Esse tratamento é feito através de **workers** que implementam um conjunto de funcionalidades refletidas através das classes ***ServerWorker*** e ***ServerWorkerAutorizado*** de forma a cumprir com requisitos do cliente.

CLIENTES

De modo a cumprir com os requisitos propostos no enunciado, foi necessário criar 2 tipos de clientes: cliente normal e cliente com autorização especial. Cada cliente possui funcionalidades particulares de acordo com o seu tipo. Assim sendo, a sua implementação teve como objetivo responder aos requisitos característicos de cada um, sendo deste modo necessário considerar duas portas diferentes, 12345 para o cliente normal e 56789 para o cliente autorizado.

CLIENTE NORMAL

Para o cliente normal, considerou-se uma implementação ***multi-threaded*** refletida através do uso da classe ***Demultiplexer***, que irá ser abordada em detalhe posteriormente, de modo a permitir efetuar múltiplos pedidos e receber respostas assíncronas.

Este tipo de cliente possui diversas funcionalidades, desde o momento de autenticação/registo até ao momento em que abandona o programa (através do log-out ou de notificação covid). Após autenticação, o cliente tem acesso às funcionalidades do programa que serão processadas pelo servidor.

CLIENTE AUTORIZADO

No caso do cliente com autorização especial considerou-se um cliente ***single-threaded*** uma vez que este só implementa uma funcionalidade síncrona. Caso o cliente fosse multi-threaded não se tiraria proveito desta implementação e haveria desperdício de memória com a criação de *threads* que não seriam utilizadas. Considerou-se que devido ao estatuto do cliente este só tem acesso à funcionalidade de descarregar o mapa com o número de utilizadores e doentes em

cada coordenada. Caso este cliente pretenda aceder às restantes funcionalidades terá de se autenticar no sistema como cliente normal.

CONEXÃO CLIENTE ↔ SERVIDOR

Como forma de implementar a conexão entre o cliente ↔ servidor, foram utilizadas diversas classes: ***TaggedConnection***, ***Demultiplexer*** e ***Listener***. A figura seguinte ilustra a conexão feita entre o cliente ↔ servidor.

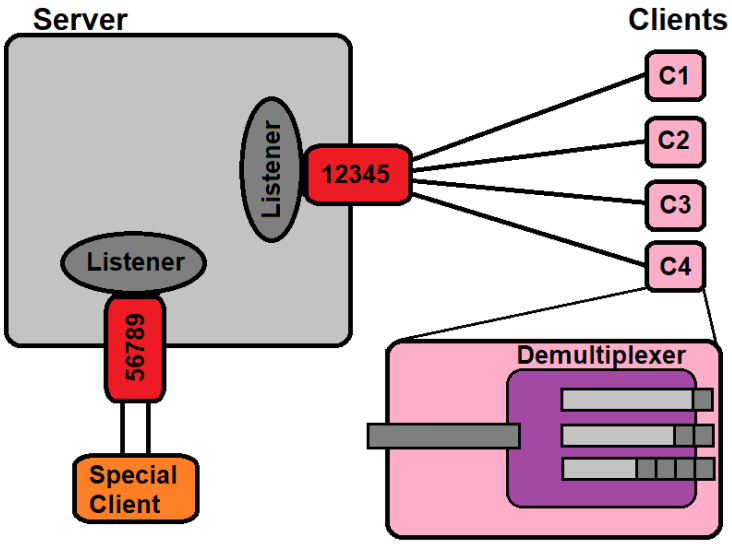


Figura 1: Conexão Cliente ↔ Servidor

A partir da imagem seguinte conseguimos ver que 2 tipos de clientes, isto é, com autorizações diferentes, estão conectados ao servidor em portas distintas. Cada porta está a ser "escutada" por um thread que vai criar as threads por cada nova conexão. Do lado do cliente normal (c1, c2, c3, c4) esta implementado um *demultiplexer* que tem como uma função "escutar" o socket de input como forma de distribuir as respostas do servidor pelos *threads* respetivos de acordo com a tag da mensagem. Do lado do cliente especial, como foi mencionado anteriormente, não há necessidade de demultiplexer pois este é *single-threaded*.

TAGGEDCONNECTION

A classe ***TaggedConnection*** tem como função a gestão da troca de mensagens entre cliente e servidor e vice-versa recorrendo a recursos de etiquetamento e serialização de mensagens. Deste modo, estão implementados métodos de envio e receção de mensagens para o socket, recorrendo à serialização e escrita em binário (*DataInputStream* e *DataOutputStream*). Como esta classe foi utilizada nas classes cliente e servidor, existem métodos de implementação diferentes de envio e receção particulares ao cliente e ao servidor. Nesta classe estão implementadas as classes ***FrameCliente*** e ***FrameServidor*** que tem o intuito de armazenar a informação recebida pelo servidor e cliente, respetivamente.

DEMULTIPLEXER

A classe *Demultiplexer* tem como objetivo agrupar as respostas provenientes do servidor de acordo com a sua tag de forma a distribuí-las pelas *threads* que enviaram o pedido respetivo. Para cumprir com esse objetivo, existe um *thread* que está à "escuta" de novas mensagens do socket de input do lado do cliente que após a sua receção irá "acordar" os *threads* que estejam à espera de resposta da tag da mensagem recebida.

FUNCIONALIDADES

Para a implementação do programa "Alarme Covid" foram considerados 2 tipos de clientes distintos, tal como referido anteriormente. Cada um destes clientes possui funcionalidades distintas.

O cliente normal pode: efetuar autenticação/registo (tag 1/0), mudar de localização (tag 2), obter o número de utilizadores numa dada posição (tag 3), mover-se para uma localização quando não houver lá mais utilizadores (tag 4), notificar que está contaminado com covid-19 (tag 5) e, por fim, efetuar log-out da aplicação (tag 6). No caso do cliente autorizado, este tem capacidade para descarregar um mapa onde consta o número total de utilizadores e o número de doentes que passaram em cada posição, desde o início do programa (tag 8).

Para cada funcionalidade do cliente normal, irá ser enviado um pedido ao servidor por uma *thread* específica, de modo a permitir respostas assíncronas e a evitar o bloqueio do programa.

AUTENTICAÇÃO E REGISTO

Cliente: No lado do cliente, o utilizador escolhe para iniciar sessão com uma conta já existente (autenticação) ou com uma nova conta (registar novo utilizador). Em cada método é pedido um nome de utilizador que deverá ser único e uma palavra passe que serão enviados para o servidor.

Servidor: No servidor, caso o utilizador tenha escolhido iniciar sessão, é validada a correspondência entre a palavra passe e o nome de utilizador. Posteriormente, é enviada uma mensagem de confirmação ou de erro cajo haja ou não haja correspondência. Para o caso de registar um novo utilizador é verificada a unicidade do nome de utilizador e caso se verifique é armazenado com a correspondente palavra passe. De seguida, é enviado para o cliente uma mensagem de confirmação com a posição inicial deste, gerada aleatoriamente.

Cliente: Finalmente o cliente vai verificar a mensagem recebida do servidor de forma a determinar se a autenticação ou o registo foram confirmados por parte do servidor. Caso seja confirmado, o utilizador fica autenticado e pode ter acesso a todas as funcionalidades do programa.

ALTERAR LOCALIZAÇÃO UTILIZADOR

Para implementação deste método foi necessário a criação de uma classe mapa que implementa um mapa 5x5 representado na figura ao lado.

Nesta classe estão implementadas 3 estruturas do tipo *map*: *localizacaoTotal* que associa a cada coordenada um conjunto de utilizadores que lá passaram desde o seu registo, *localizacaoAtual* que associa a cada coordenada um conjunto de utilizadores que se encontram atualmente nessa posição e *localizacaoDoentes* que associa a cada posição os utilizadores doentes que por lá passaram.

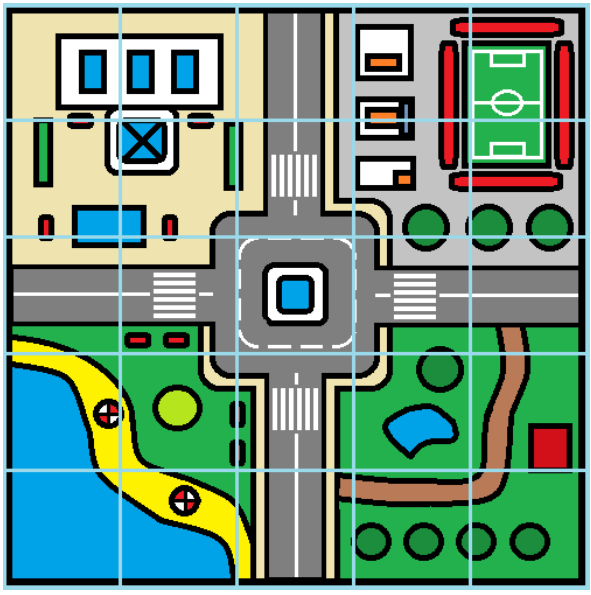


Figura 2: Mapa 5x5 implementado

IMPLEMENTAÇÃO DO MÉTODO

Cliente: É pedido ao utilizador que escolha entre mudar para uma localização gerada aleatoriamente ou uma localização manualmente inserida pelo mesmo. No caso desta última, é pedido ao utilizador que indique as coordenadas dessa nova posição que serão posteriormente enviadas para o servidor.

Servidor: O servidor vai gerar uma posição aleatória ou utilizar as coordenadas recebidas, dependendo da opção do utilizador. Em ambos os casos, o servidor vai atualizar o mapa implementado, isto é, vai alterar no map *localizacaoAtual* a posição do utilizador e acrescentar no map *localizacaoTotal* o nome do utilizador na nova posição. No utilizador correspondente também é acrescentada a nova posição na sua lista de posições. Posteriormente é enviada para o cliente uma mensagem de confirmação com a alteração da localização do utilizador.

Cliente: Finalmente, o cliente imprime a nova localização.

NÚMERO DE UTILIZADORES NUMA LOCALIZAÇÃO

Cliente: Solicita informação quanto ao número de utilizadores numa dada localização, fornecida pelo próprio.

Servidor: Efetua a contagem dos utilizadores, recorrendo à classe mapa. Para isso, conta-se o número de utilizadores que estão na posição recebida através do map *localizacaoAtual*. Posteriormente, essa contagem é enviada numa mensagem para o cliente.

Cliente: Finalmente o cliente vai devolver a mensagem com a contagem recebida pelo servidor ao utilizador.

MOVER-SE PARA UMA LOCALIZAÇÃO QUANDO ESTIVER LIVRE

Cliente: Solicita ao servidor que o informe quando mais nenhum utilizador estiver numa dada localização (fornecida pelo próprio) com o intuito de se vir a deslocar para lá.

Servidor: Após receber a posição desejada, o servidor invoca um método que adormece a *thread* enquanto essa posição não estiver vazia, recorrendo ao uso do *await* através da variável de condição "*notEmpty*". De seguida, verificou-se que a posição desejada poderia ficar livre quando os restantes utilizadores se moviam ou quando um utilizador era removido do mapa (após ser contaminado com covid-19). Assim, nos métodos da classe mapa que tratam estes acontecimentos, foi efetuado o *signalAll* da mesma variável de condição, de modo a acordar os *threads* anteriormente adormecidos no *await*. Esta *thread* pode agora readquirir o lock e prosseguir com a execução. O servidor altera a localização do utilizador para a desejada, dado que já mais nenhum utilizador se encontra lá. De seguida, envia a mensagem de sucesso ao cliente.

Cliente: Após o processamento do seu pedido no servidor, é lhe enviada uma mensagem de sucesso que notifica que a sua posição foi alterada.

NOTIFICAR POSSÍVEL CONTÁGIO E INFEÇÃO DE UTILIZADOR

Para cumprir com esta funcionalidade é necessário que se armazene todas as conexões feitas entre o servidor e cada cliente no *map connections* que associa um nome de utilizador com a respetiva conexão (*TaggedConnection*). Este armazenamento vai ser utilizado para posteriormente notificar os clientes que tenham sido possivelmente contagiados. Como forma de o cliente estar sempre à espera de uma notificação de possível contágio, após a autenticação de cada cliente existe uma *thread* que esta à espera dessa notificação proveniente do servidor. Assim, é possível que cada cliente consiga aceder a todas as funcionalidades da aplicação sem que esteja bloqueado.

Cliente: Um utilizador solicita envio de uma mensagem ao servidor a informar que se encontra contaminado.

Servidor: Quando o servidor receber uma notificação de contaminação vai invocar um método da classe mapa que remove esse utilizador dos *maps localizacaoAtual* e *localizacaoTotal* nas posições onde esteve e adiciona as posições por onde passou ao *map localizacaoDoentes*. Além disso, esse utilizador fica também impedido de voltar a utilizar o programa. Seguidamente, são também verificados quais os clientes que estiveram nas mesmas posições que o cliente contaminado, armazenando os seus nomes de utilizador numa lista temporária. Para cada utilizador dessa lista será enviada uma mensagem de possível contaminação através do recurso ao *map connections* (previamente populado com as conexões cliente-servidor).

Cliente: Para o cliente que notificou o servidor de contágio, a conexão entre o servidor e esse cliente será fechada e os restantes clientes que tenham sido notificados vão receber uma mensagem no terminal respetivo.

DESCARREGAR MAPA – CLIENTE AUTORIZADO

Cliente Autorizado: O utilizador solicita um mapa com informações relativas ao total de utilizadores e o número de doentes que passaram em cada localização.

Servidor: Recorre a um método presente na classe mapa que efetua as contagens requiridas. Para isso, percorre os diferentes *maps*. O número total de utilizadores pode ser obtido com recurso ao map *localizacaoTotal.* Analogamente, o número de doentes que passaram em cada coordenada é obtido através do map *localizacaoDoentes*.

Cliente Autorizado: O cliente recebe o mapa solicitado.

CONCLUSÃO

Dado por concluído o projeto “Alarme Covid”, consideramos necessário efetuar uma reflexão crítica do trabalho realizado e dos resultados obtidos.

Deste modo, é importante fazer um balanço do trabalho realizado, tendo em conta tanto os aspetos positivos como possíveis melhorias na implementação.

No espetro positivo, sublinhamos o facto de todas as funcionalidades estarem implementadas (básicas e adicionais). Além disso, a implementação do cliente normal com múltiplas *threads* impede esperas ativas no programa e permite a rececão de notificações assíncronas. Por fim, implementamos a conexão entre o cliente autorizado e o servidor através de uma nova porta distinta da porta que conecta o cliente normal ao servidor.

Por outro lado, existem melhorias que enriqueceriam o nosso programa. Seria mais benéfico se implementássemos um servidor *threaded-per-request* ao invés de *threaded-per-connection*, de modo a alocar uma *thread* a cada pedido ao invés de um número limitado de *threads* por conexão. Contudo, observamos que em termos práticos é suficiente haver cerca de 10 threads por conexão ou seja, em princípio, não haverá nenhum bloqueio por falta de *threads*.

Para finalizar, consideramos que obtivemos um balanço positivo na globalidade do trabalho, apesar das melhorias que poderiam ser efetuadas.