

UNIVERSIDADE DO MINHO
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Programação Concorrente 2019/2020

Trabalho Prático

Autores:

André Pereira	A78586
Ilda Durães	A78195
João Cerqueira	A65432

3 de Junho de 2020

Conteúdo

1	Estrutura da solução desenvolvida	2
2	Implementação	3
2.1	Classe Servidor	3
2.2	Classe Worker	3
2.3	Classe Usuário	4
2.4	Classe Registos	5
3	Resultados obtidos	6

1 Estrutura da solução desenvolvida

O problema proposto consiste no desenvolvimento de um servidor que permita recolher estimativas da proporção de infectados numa pandemia. A comunicação é orientada à linha. Cada cliente deve registar-se e autenticar-se para se ligar ao servidor. De seguida, o servidor aguarda que o cliente indique quantos casos de doença este conhece nos seus contactos. Sempre que algum cliente fornece informação ao servidor, o servidor envia a todos os clientes ligados uma nova estimativa da proporção média. Os cliente podem indicar valores de casos de doença as vezes que entenderem, até fecharem a conexão.

Assim sendo, de modo a desenvolver o pretendido, criamos as classes *Servidor*, *Worker*, *Registos* e *Usuario*, que serão descritas com detalhe no capítulo seguinte. Estas classes relacionam-se da seguinte forma:

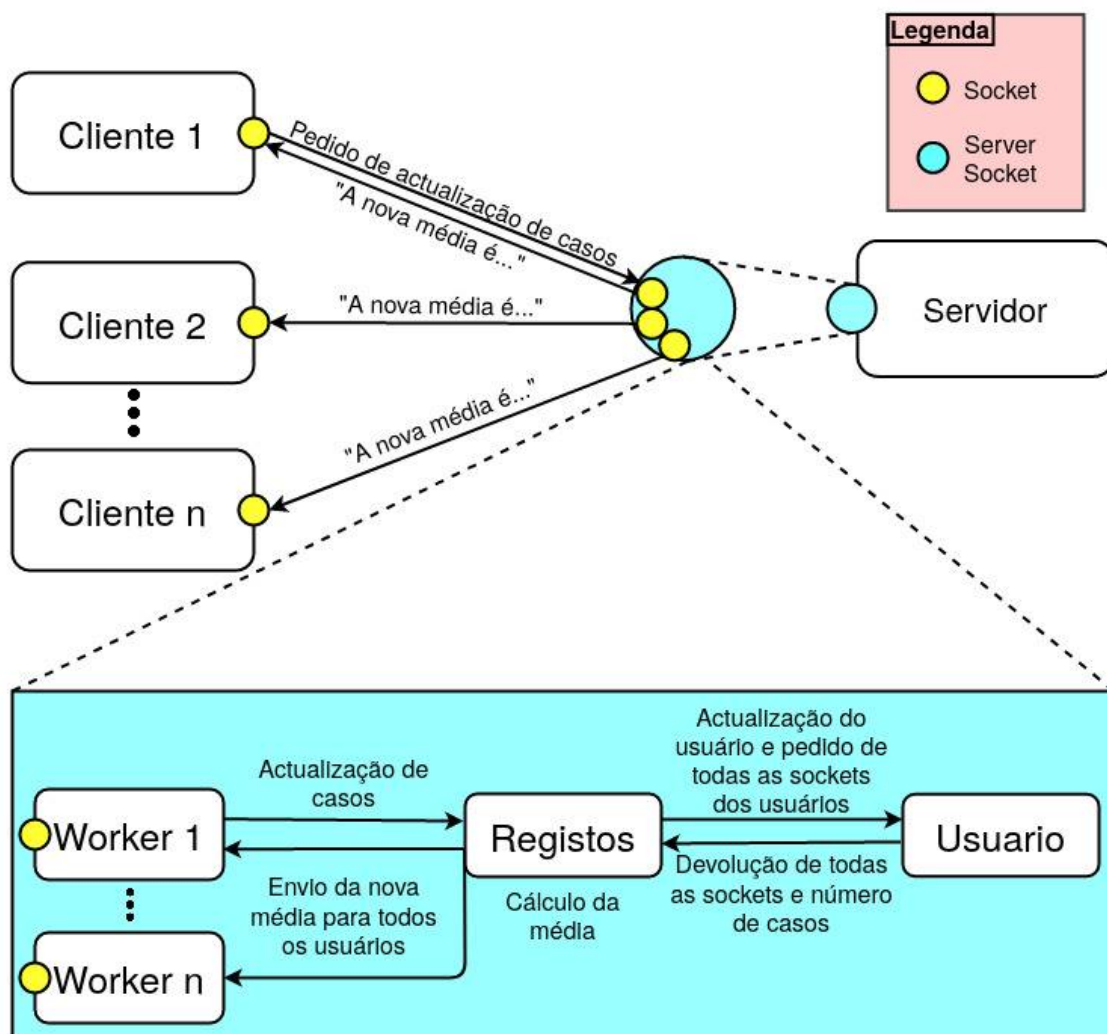


Figura 1: Arquitectura da solução desenvolvida.

2 Implementação

2.1 Classe Servidor

O servidor vai ser a classe que vai oferecer um serviço aos diferentes usuários, onde estes efetuam vários pedidos ao servidor e ele trata das respetivas respostas. A comunicação entre ambos deve ser fiável, quer isto dizer, sem perda de dados e com uma entrega ordenada dos mesmos. Posto isto um servidor fica à espera de ligações num determinado porto, quando um usuário se conecta ao servidor é estabelecida uma conexão bidireccional, onde vai existir dois *sockets* extremos, o *socket* do usuário e o *socket* do servidor.

Posto isto, começamos por definir uma variável de instância que vai ser o porto, *int port*, uma outra do tipo *Registos* que é um objeto implementado por nós, e por fim uma *sSock* do tipo *ServerSocket*. Inicialmente, o servidor começa por criar um novo objeto *ServerSocket* com a variável *sSock* mandando uma mensagem positiva caso tenha sido criado com sucesso, ou então manda um excepção, mais detalhadamente uma *IOException* caso não tenha havido sucesso na criação do mesmo.

Posteriormente, é criada uma variável *socket* do tipo *Socket* iniciada a *null*, e num ciclo infinito *while(true)* o servidor manda uma mensagem a dizer que está a correr, aceita conexões ao mesmo guardando esse acesso na variável *socket*, lançando assim uma *thread*, onde cria *multithreading*. Quando a conexão é feita com sucesso, o servidor manda uma mensagem a avisar do sucesso, e é possível a troca de dados entre este e o *socket* conectado. Caso não seja possível criar uma conexão, lança uma excepção, *IOException* mandando a mensagem que a conexão falhou. Por fim, é criado uma variável *worker* do tipo *Worker* implementado por nós, e com isto é iniciada uma nova *Thread*.

No nosso projeto, o porto usado é *port = 12345*, é criada uma variável servidor do tipo *Servidor*, e posteriormente com o método descrito em cima é iniciado o servidor e as respectivas conexões.

2.2 Classe Worker

Na classe Servidor é lançada uma Thread com uma nova instância da classe *Worker*, criada previamente onde são passados uma instância de *Registos* e o *socket*. Como cada instância da classe *Worker* é executada por uma *thread*, então a classe implementa a *interface Runnable*. Assim sendo, para além do construtor parametrizado e de vários métodos auxiliares que podemos observar na análise do código, a classe têm também um método *run()*.

De uma forma muito sucinta, no método *run()*, inicialmente é evocada o método *login_and_signup()* que é responsável por efetuar o registo ou o *login* do utilizador. Inicialmente, é assegurado que o cliente de facto inseriu um nome e de seguida é verificado se o *username* inserido já existe ou se é um novo utilizador. Quando o *username* inserido já existe, caso esteja já logado é pedido para inserir outro *username*, se não estiver logado são dadas três ten-

tativas ao utilizador para inserir a palavra passe. Por sua vez, caso se trate de um novo *username* é necessário proceder ao registo, e portanto inserir uma *password* que será associada ao *username*. De seguida, é evocado o método *get_n_infected_and_update_users* que visa enviar ao servidor quantos casos de doença o cliente conhece nos seus contactos, também neste método são tratados os erros, isto é, é assegurado que o cliente inseriu de facto um número e que este número não é superior a 150 (o número de pessoas que cada cliente conhece). Posteriormente, após o cliente inserir um número é evocado o método *set_casos_and_update_all_users* à instância de *Registos* associado a este *Worker*, passando o *username* e o número inserido como argumentos.

Por fim, quando o cliente se desconecta, é feito *logout*, através do método *set_user_to_logged_out(username)* da instância de *Registos* associada e é fechado o *socket*, assim como os respectivos canais.

2.3 Classe Usuário

A *Classe Usuário* visa armazenar as informações associadas a cada cliente do sistema. Tem como variáveis: *String password* onde é guardada a palavra-passe, *int casos* que armazena quantos casos de doença o utilizador conhece, *BufferedWriter bufferedwriter* é o canal de comunicação no qual são escritas as respostas para o usuário, *Boolean logged_in* que identifica se o usuário está logado ou não, e por fim, o *ReentrantLock locker*.

Esta classe usa locks explícitos, uma vez que vamos precisar de efectuar operações de leitura e escrita numa só operação atómica (sem libertar o objecto), e consequentemente é necessário bloquear cada objecto por completo de forma a ter a garantia que não ocorrem alterações enquanto trabalhamos sobre esse objecto. Isto tem como efeito secundário o acesso sequencial a cada usuário, não permitindo múltiplas leituras de um objecto em nenhum momento. Porém, como cada usuário terá em princípio acesso apenas ao seu objecto, o impacto na performance será menor.

Os métodos desta classe são os *getters* e os *setters* que, respectivamente devolvem e alteram conteúdo presente nas variáveis *bufferedwriter*, *password*, *logged_in* e *casos*. Também, *lock()* e *unlock()* são métodos desta classe, estes aplicam ao *locker* os métodos *lock()* e *unlock()* da classe do Java *ReentrantLock*, isto é, quando evocamos o método *lock()* o valor do *hold count* é incrementado e se a instância da classe *Usuário* estiver livre, esta é bloqueada, quando o método *unlock()* é evocado o valor do *hold count* é decrementado e quando atinge o zero o recurso é libertado.

2.4 Classe Registos

A classe *Registos* é a classe que é instanciada apenas pelo servidor, uma vez que apenas pode existir uma instância desta classe de modo guardar os dados de todos os usuários juntamente com os seus *usernames*, os quais são guardados como uma par (*key,value*) numa *HashMap*, em que a chave é o *username*, impedindo o registo de dois usuários com o mesmo.

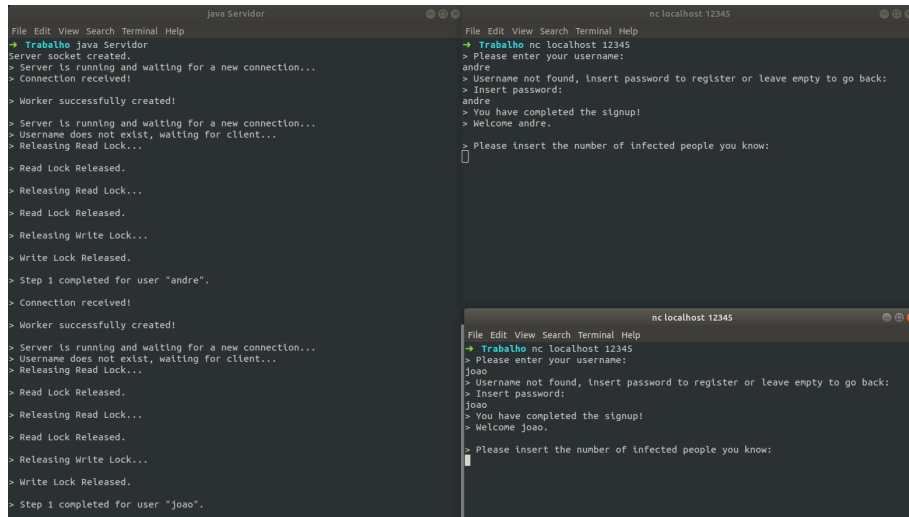
Tendo em consideração que na classe *Registos* temos vários *Workers* sempre a aceder à *Hashmap* para leitura e escrita, optamos por implementar *Monitores* com variáveis de condição, de forma a permitir múltiplas leituras por vários *Workers* diferentes. Estabelecemos duas condições: A condição *reader_wait* é activada quando existem *writers* à espera para escrever ou activos, e portanto é necessário informar os *Workers* que querem ler que têm de esperar. A condição *writer_wait* informa os *writers* que existem *readers* activos ou um *writer* activo, e portanto é preciso esperar até que nenhum esteja activo.

A classe *Registos* é responsável por controlar e gerir a forma como o *Worker* e o *Servidor* podem alterar e aceder a cada instância de *Usuário*. Ou seja, possibilita que cada *Worker* aceda ao registo que o *Servidor* têm de todos os *Usuários* de forma a poder ler e escrever dados para cada *Usuário* individual, assim como para a *HashMap* em si.

Assim sendo, através de vários métodos implementados nesta classe, é possível alterar a informação sobre se um *Usuario* está ou não logado (e consequentemente a variável *logged_in* da instância de *Usuário*), verificar se um utilizador está ou não logado, verificar se um *username* já existe, registar um novo usuário e fazer *login* ao mesmo.

Para além disso, a Classe *Registos* também tem um método crucial neste sistema que é o *set_casos_and_update_all_users*. Inicialmente, é feito *lock* à classe *Registos* para leitura. Após receber a informação sobre a atualização do número de casos que um usuário conhece, é feito *lock* à instância *Usuario* associada ao *username* e feita esta atualização. De seguida, vão sendo adquiridos os *locks* de cada *Usuário* e é recolhida a informação sobre o número de casos associados a cada um, para posteriormente recalcular a estimativa de número de casos e o *lock* de leitura da classe *Registos* é libertado. Por fim, pela ordem inversa da aquisição dos *locks* a informação sobre a nova estimativa é enviada a todos os *Clientes* logados e é feito o *unlock*.

3 Resultados obtidos

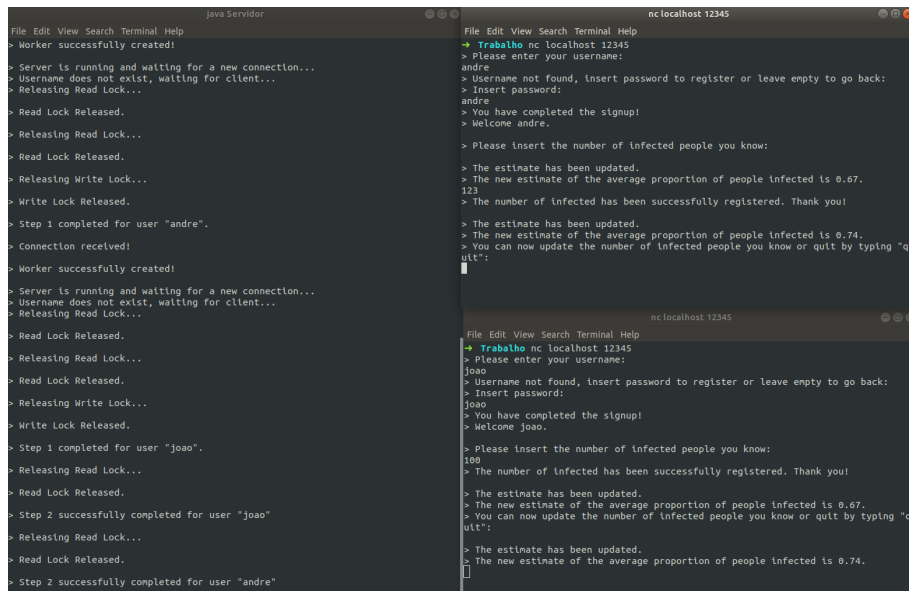


```
File Edit View Search Terminal Help
Trabalho java Servidor
> Server socket created.
> Server is running and waiting for a new connection...
> Connection received!
> Worker successfully created!
> Server is running and waiting for a new connection...
> Username does not exist, waiting for client...
> Releasing Read Lock...
> Read Lock Released.
> Releasing Read Lock...
> Read Lock Released.
> Releasing Write Lock...
> Write Lock Released.
> Step 1 completed for user "andre".
> Connection received!
> Worker successfully created!
> Server is running and waiting for a new connection...
> Username does not exist, waiting for client...
> Releasing Read Lock...
> Read Lock Released.
> Releasing Read Lock...
> Read Lock Released.
> Releasing Write Lock...
> Write Lock Released.
> Step 1 completed for user "joao".

File Edit View Search Terminal Help
Trabalho nc localhost 12345
> Please enter your username:
andre
> Username not found, insert password to register or leave empty to go back:
> Insert password:
andre
> You have completed the signup!
> Welcome andre.
> Please insert the number of infected people you know:

```

Figura 2: Exemplo de conexão de dois usuários ao servidor



```
File Edit View Search Terminal Help
Trabalho java Servidor
> Worker successfully created!
> Server is running and waiting for a new connection...
> Username does not exist, waiting for client...
> Releasing Read Lock...
> Read Lock Released.
> Releasing Read Lock...
> Read Lock Released.
> Releasing Write Lock...
> Write Lock Released.
> Step 1 completed for user "andre".
> Connection received!
> Worker successfully created!
> Server is running and waiting for a new connection...
> Username does not exist, waiting for client...
> Releasing Read Lock...
> Read Lock Released.
> Releasing Read Lock...
> Read Lock Released.
> Releasing Write Lock...
> Write Lock Released.
> Step 1 successfully completed for user "joao"
> Step 2 successfully completed for user "andre"

File Edit View Search Terminal Help
Trabalho nc localhost 12345
> Please enter your username:
joao
> Username not found, insert password to register or leave empty to go back:
> Insert password:
joao
> You have completed the signup!
> Welcome joao.
> Please insert the number of infected people you know:
100
> The number of infected has been successfully registered. Thank you!
> The estimate has been updated.
> The new estimate of the average proportion of people infected is 0.67.
123
> The number of infected has been successfully registered. Thank you!
> The estimate has been updated.
> The new estimate of the average proportion of people infected is 0.74.
> You can now update the number of infected people you know or quit by typing "q
uit":

```

Figura 3: Exemplo de atualização de estimativa de ambos os usuários

De modo a assegurar a correta execução da solução, desenvolvemos um *script* que estabelece comunicações infinitas com o servidor e obtivemos o seguinte resultado:

