Universidade do Minho

Relatório de desenvolvimento do Trabalho Prático Unidade Curricular de Programação Concorrente

Licenciatura em Ciências da Computação
Universidade do Minho

Manuel Marques (A85328)

Pedro Oliveira (A86328)

João Rodrigues (A84505)

2 de Junho de 2020

Índice

Capítulo 1

Contextualização

O trabalho consiste no desenvolvimento de um servidor que permita recolher estimativas da proporção de infetados numa pandemia. A comunicação é orientada à linha. Cada cliente efetua registo uma única vez e autentica-se para se ligar ao servidor. O servidor aguarda que o cliente indique quantos casos de doença este conhece nos seus contactos (valor superior a 0). Sempre que algum cliente fornece informação ao servidor, o servidor envia a todos os clientes ligados uma nova estimativa da proporção média.

Nota: Para este cálculo assume-se que cada cliente conhece 150 contactos, pelo que um cliente que indique 15 casos, está a indicar a proporção de 0.1.

O servidor reporta a média aritmética das proporções recebidas. Os clientes podem indicar valores de casos de doença as vezes que entenderem, até fecharem a conexão.

É importante realçar que as notificações são assíncronas e os clientes devem receber estas mensagens mesmo sem estar a fazer pedidos ao servidor. Por outras palavras, não se deve assumir que o servidor só envia estimativas globais como resposta a uma nova operação remota do cliente.

Ainda, o trabalho deve considerar a autenticação e registo de utilizadores, dado o seu nome e palavra-passe. Sempre que um utilizador desejar interagir com o serviço deverá estabelecer uma conexão e ser autenticado pelo servidor.

São de notar alguns aspetos:

- Cada máquina cliente sabe o IP e Porto do servidor.
- São utilizadas primitivas de programação com sockets, programação concorrente e controlo de concorrência local sempre que necessário.
- O desenvolvimento foi feito na linguagem Java
- Para o cliente, optamos por criar um programa próprio, ao invés da utilização do "nc"

Capítulo 2

Metodologia utilizada

Para facilitar o trabalho definimos a porta como "12345"e o endereço é o local de "127.0.0.1", sendo que todos os clientes, quando inicializado o programa, vão conectar a esse endereço.

2.1 *Server*

O nosso servidor tem a a capacidade de receber diversas conexões de clientes.

Quando inicializamos o *server* criamos um novo *ServerSocket* e uma *Thread*, cujo propósito é permitir a interação com o teclado de forma a que o *Server* possa ser desligado,usando o comando "exit", para ser efetuada a gravação dos registos.

Inicializamos depois um ciclo em que ficamos à espera de conexões.

Assim que um cliente se conecta, é criada uma segunda *Thread* com um *ServerWorker*, que vão receber os pedidos e enviar as respostas a esses pedidos.

Após *login* ou registo com sucesso por um cliente , é criada uma outra *Thread* associada a esse mesmo cliente que ficará responsável por enviar a média atualizada, sempre que houver alterações por parte de outros utilizadores. Esta *Thread* está adormecida até que haja uma notificação *signalAll()* aquando de uma alteração do número de casos.(Usamos *signalAll()* porque poderão existir várias *Threads* nesta condição). Isto garante que o processo de *multicast* não é afetado por um cliente que tenha ligações instáveis.

2.1.1 Controlo de Concorrência

Uma vez que temos duas *Threads* a comunicar com um cliente pelo mesmo *PrintWriter*, foi necessário controlar a forma como cada uma delas consegue acesso a esse extremo de escrita. Para isso, antes de cada *Thread* enviar mensagem ao cliente, deve garantir acesso ao *PrintWriter*.

Controlo de Concorrência nos métodos

```
public boolean registerClient(String user, String pass, String region,PrintWriter out) throws InterruptedException {
    lock_clientRegisters.lock();
    for (InfoClient c : client_registers) {
        if (c.getUsername().equals(user)) {
            lock_clientRegisters.unlock();
            return false;
        }
    }
    InfoClient cl = new InfoClient(user, pass, region);
    client_registers.add(cl);
    //talvez nao seja necessário bloquear os case_registers(fica a dúvida)
    case_registers.put(user, 0);
    lock_clientRegisters.unlock();
    return true;
}
```

Figura 2.1: Método de registo de um cliente

Como podemos ver na figura 2.1, antes de registar-mos um cliente, fazemos *lock* ao *ArrayList* que contém os dados de todos os clientes, de forma a garantir que nenhum outro utilizador se registe com o mesmo *username* e só depois de inserido o novo cliente, é libertado o *lock*.

```
public boolean updateCases(String user, int x) { // Precisa de ser revista a concorrência
    int c = case_registers.get(user);
    if(x+c <= 150) {
        lock_clientRegisters.lock();
        case_registers.put(user, x + c);
        total_cases += x;
        lock_clientRegisters.unlock();
        return true;
    }
    return false;
}
```

Figura 2.2: Método de atualização do número de casos de um cliente

A variável c, irá guardar o número de casos atuais para um determinado utilizador.Aqui não há necessidade de garantir o *lock*, pois trata-se apenas de leitura e há garantia que esses dados não serão alteradas pelo utilizador antes do passo seguinte. De seguida, fazemos *lock* do *ArrayList* que contém os dados de todos os clientes e da variável "total_cases"de forma a garantir que durante o processo de atualização, não existam outras *Threads* na mesma secção crítica. Atualizados os registos, libertamos o *lock*.

Restantes métodos são similares, e serão debatidos na apresentação, caso necessário.

Definição das Classes

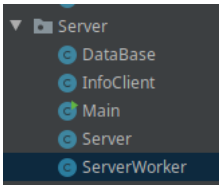

Figura 2.3: Classes do programa *Server*

- *DataBase*: Classe que contém os métodos para tratamento dos pedidos dos clientes
- *InfoClient*: Classe que contém os dados de um cliente
- *Server*: Classe que recebe as ligações dos clientes e despoleta a criação das *Threads*
- *ServerWorker*: Classe que interage com o cliente
- *Main*: Classe que arranca o *Server*

2.2 Client

Cada cliente, terá duas *Threads*, uma para receber interação com o teclado e enviar mensagens ao *Server*. A segunda *Thread* é utilizada para receber as mensagens do *Server*.

Optamos por colocar do lado do programa *Client* a impressão dos menus com as opções, evitando desta forma sobrecarregar o servidor, sendo apenas enviados para este, pedidos de trabalho concretos.

Definição das Classes

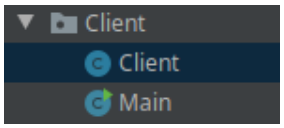

Figura 2.4: Classes do programa *Client*

- *Client*: Classe que lê os *inputs* e disponibiliza as opções ao utilizador e as comunica ao *Server* e recebe as respostas
- *Main*: Classe que arranca o *Client*, fazendo a ligação através de IP e porto.

Capítulo 3

Conclusão

Com a realização deste projeto foi possível ter um contacto com mecanismos de controlo de concorrência que nos permitiram ganhar sensibilidade para a necessidade existir uma correta organização desses mesmos mecanismos, principalmente nos sistemas distribuídos. Por vezes existiram algumas dúvidas sobre quando é ou não necessário garantir proteção das secções críticas. Nesses casos e, quando não foi possível dissipar essas dúvidas, optamos por manter protegidas as secções críticas. Será certamente um tema de debate na apresentação.