Sistemas Distribuídos
Exame de Época Especial

16 de Setembro de 2011

02/01/2018

II

```
public class Entrada {
    private long hora;
    private String user;
}
```

```
public enum OpsEspaco {
    containsCliente;
    getEntrada;
}
```

```
public class Espaco {
    private String id;
    private int capacidade;
    private HashMap<String, Entrada> atuais;
    private ReentrantLock lock = new ReentrantLock();
    private Condition cond = lock.newCondition();

    public boolean containsCliente(String e) {
        lock.lock();
        boolean res = atuais.containsKey(e);
        lock.unlock();
        return res;
    }

    public Entrada getEntrada(String e) {
        lock.lock();
        Entrada é = atuais.get(e);
        lock.unlock();
```

```
    return e;
}

public void entrar (String e){
    lock.lock();
    while (atuais.size() == capacidade){
        lock.unlock();
        cond.await();
        lock.lock();
    }
    atuais.put(e, new Entrada(e, System.currentTimeMillis()));
    lock.unlock();
}

public Entrada sair (String e){
    lock.lock();
    Entrada e = atuais.remove(e);
    if (e != null){
        cond.signalAll();
    }
    lock.unlock();
    return e;
}
}
```

FALTA A PARTE DO SERVIDOR!

public ... <String, Entrada> ...

I

1. O monitor é uma primitiva de controlo de concorrência que basicamente apenas permite a execução de um método num determinado "objeto" de apenas permite a que esteja 1 "objeto" com ele adquirido, pelo que é um lock reentrante. Esta primitiva de controlo estruturado pode ser considerada de alto nível, pois apesar de necessitar da "ajuda" do hardware neste controlo de concorrência, esta é realizada de uma forma transparente.

Uma vantagem é esta transparência e facilidade na utilização da primitiva, que nos permite não nos preocupar com questões que poderiam ser complicadas (garantir atomicidade nas operações de verificação e atualização) e no fato de evitar já as esperas ativas.

Uma desvantagem poderá ser a facilidade com que a sua utilização descuidada ou indevida levará a situações de deadlock.

2.

A função atribuída ao cliente é basicamente utilizar os serviços fornecidos pelo servidor. Normalmente invoca um pedido ao servidor e espera (preferencialmente sem ser ativa) pela resposta do servidor.

A função atribuída ao servidor é basicamente fornecer serviços ao cliente. Este servidor, usualmente, está à espera de pedidos do cliente (preferencialmente sem ser ativa) para que depois possa responder a esse pedido. Um objetivo é que um servidor seja eficiente e aceite conexões de múltiplos clientes.

II

```
public class AceitaCliente implements Runnable {
    Espaco e;                    int porta;
    ArrayList<String> pedido;    ServerSocket ss;

    public void run() {
```

```
        while(true){
Socket s = SS.accept();
    }   (new Thread(new TrataCliente(s, e, resto, porta))).start();
}
}
                    public enum OpsEntrada{
                        Entrar,
                        Sair,
                        Onde_Esta;
                        (ver depois 'migrar')
                        CONTEM;
                        N_CONTEM;
                    }
public class TrataCliente imp. Runnable{
    String cliente;
    Socket s;
    Entrada e;
    ArrayList<String> resto;
    int porta;

public void run(){
    try{
        OOS out = new OOS(s.getOS());
        OIS in = new OIS(s.getIS());
        OpsEntrada oh;
        while(true){
            oh=(OpsEntrada) in.readObject();
        try{switch(oh){
            case Entrar:
                e. entrar(cliente);
                break;
```

```
case Sair:
        e.sair(cliente);
case Onde_Esta:
    if(es.contemCliente(cliente)){
        Entrada ent = e.getEntrada()
        out.println(ent);
    }
    else{
        ArrayList<Thread> ts = new AL<>();
        for(String s : resto){
            Thread th = new Thread(new VaiBuscar(s, cliente,
                                                 horta,
                                                 out));
            ts.add(th);
            th.run();
        }
        for(Thread th: ts){
            th.join();
        }
    }
    break;
    default: Erro!
    }catch(Exception e){}
}catch(Exception e){}
    s.shutdownOutput();
    s.  "   Input();
    s.close();
}
```

```
public class VaiBuscar ... {
    private String cliente;
    // // host;
    // int porto;

    private OOS output

public void run(){
    Socket s= new Socket(host, porto);

    OOS out= ...
    OOS in = ...

    out.writeObject(OpsEspaco.Onde_Esta);
    out.write(cliente);

    OpsEspaco op=(OpsEspaco) in.readObject();
    switch(op){
        case(CONTEM):
            Entrada e= (Entrada) in.readObject();
            output.write(object e);
            break;
        default: //N CONTEM
    }
}
}

public class AceitaServidor ... {
    ServerSocket ss; Entrada e;
    public void run(){

    while(true){
    Socket s= ss.accept();
```

```
        (new Thread(new BuscaServidor(ss, e))).start();
    }
}

public class BuscaServidor ... {
    public Socket s;
    private Entrada e;

public void run() {
    try {
        OOS out = ...
        OIS in = ...

        OpsEntrada op = (OpsEntrada) in.readObject();
        switch(op) {

            case Onde_Esta:

                String user = in.read();
                if (e.contemCliente(user)) {

                    out.writeObject(OpsEntrada.CONTEM);
                    Exit.writeObject(e.getEntrada(user));
                }
                else {
                    out.writeObject(OpsEntrada.N_CONTEM);
                }

            default: //ERRO!
        }
    } catch (Exception e) {}
    finally { s.ShutdownOutput()
              s.shutdownInput();
              s.close();
    }
}
}
```

```
public class servidor {
public static void main(String [] args){
    String idE = args[1];
    ServerSocket ss = new ServerSocket(porto);
    int porto = Integer.parseInt(args[2]);
    if(porto == 9999){
        System.out.println("Porto definido para comunicação interno.");
    }
    else{ Estaco e = new Estaco(idE);
        ArrayList<String> resto = new AL<>()
        todos args a partir de 3 adicionar a resto;
        ServerSocket ss2 = new ServerSocket(9999);
        Thread t1 = new Thread(new AceitaCliente(e, resto, 9999, ss));
        Thread t2 = new Thread(new AceitaServidor(ss2, e));

        t1.start();
        t2.start();

        t1.join();
        t2.join();
    }
}
```