Sistemas Distribuídos
Avaliação Escrita

20 de janeiro de 2014

07/01/2018

II

```
//peso mínimo: 200 Kgs

class Transporte {

    int pesoMinimo = 200        int vez = 0
    int pesoAtual = 0
ReentrantLock l = new ReentrantLock();
Condition c = l.newCondition();
int encomendaAtual = 0;

    public Transporte() { }

    public int queroEncomenda(int peso) {

        l.lock();
        int minhaVez = vez
        pesoAtual += peso;
        int meuNum = encomendaAtual++;
        while ((pesoAtual < pesoMinimo) && (minhaVez == vez)) {
            c.unlock();
            c.await();
        } l.lock();


        if(minhaVez == vez){
                vez++;
                pesoAtual = 0;
                c.signalAll();
        }
        l.unlock();
        return meuNum;
    }
}
```

```
class TrataCliente {

    Transporte t;
    Socket s;
public TrataCliente(t, os){

    this.t=t;
    this.s=s;
}

public void run(){
    try{ <- declaração aqui

        PrintWriter out= new PrintWriter(es.getOutputStream(), true);
        BufferedReader in = new BufferedReader(es.getInputStream());

        int peso;
        String msg;

        msg= in.readLine();

        peso= Integer.parseInt(msg);

        int numero= t.queroEncomenda();

        out.println(numero);

    finally{ out.close();
             in.close();
             S.close();
    }
}
}
```

```
class Servidor {

  public static void main(String [] args) {
    Transporte t = new Transporte();
    ServerSocket s = new ServerSocket(9999);

    while (true) {

      Socket socket = s.accept();
      TrataCliente e = new TrataCliente(t, socket);

      Thread te = new Thread(e);

      te.start();
    }
}

class Cliente {

  public static void main(String[] args) {

    Socket s = new Socket("127.0.0.1", 9999);
    PrintWriter out;
    BufferedReader in;
    Scanner ler;

    try {
      out = new PW(s.getOS(), true);
      in = new BR(s.getIS());
      ler = new Scanner(System.in);
      String msg;
      System.out.println("Indique o peso da encomenda!");
      msg = ler.nextLine();

      out.println(msg);

      msg = in.readLine();
      System.out.println("O número da sua encomenda é "
          + msg + "!");
```

```
        System.out.println("Obrigado pela preferencia!");
    } finally { out.close();
                in.close();
                s.close();
            } ler.close();
    }
}
```

I

1. Região crítica é alguma região no código que por alguma razão (partilha de variáveis) apenas pode ser acedida de uma forma controlada por diferentes processos. Para que não aconteçam "race conditions" é necessário haver sincronização. Para isto, podemos verificar os casos da exclusão mútua que garante que só 1 processo está na sua região crítica, mais ninguém o pode fazer, o que permite que apenas 1 processo aceda aos recursos partilhados de cada vez.
Para garantir exclusão mútua, e outro tipo de sincronização são utilizados, muito frequentemente, as variáveis de condição que permitem que um processo teste se pode entrar na região crítica e caso não possa, "adormece voluntariamente". Estão sempre associadas a 1 lock.

2. Em ambos é preciso sincronização, para aceder às contas/gastos (todas).

[Em ambos é necessário uma ordem de ter acesso (lock) à conta/gasto que queremos aceder.

Basicamente a conta é o gasto!

3. Consegue ter o mecanismo de transparência desejado, devido ao facto de que o sistema externo não se aperceber da invocação

remota destes procedimentos. Para o sistema externo a invocação comporta-se da mesma forma como se fosse feita localmente e até em casos de falha, irá existir outra máquina (provavelmente) à qual esse procedimento poderá ser invocado.