Sistemas Distribuídos
Exame de Recurso

9 de Fevereiro de 2015

07/01/2018

II

```
public enum OPSJogo{

    disparo;

}

public class Jogo{
    int [n][m] matriz= inicializar();
    HashMap<String, Integer> pontuacoes;
    HashMap<String, Integer> jogadas;
    ReentrantLock l=new ReentrantLock();
    Condition c = l.newCondition();
    int vez=0;
public Jogo (jogadores) {

   for (String j: jogadores){
       pontuacoes.put(j, 0);
       jogadas.put(j, 0);
   }
}

public String vencedor(){
    int max=-1; String jog="";
    for (Map.Entry<String, Int> l: pontuacoes.entrySet()){
        int h= l.getValue();
        if(h> max){
            jog = l.getKey();
            max = h;
        }
    }
    return jog;
}
```

```
String
public disparo (String jog, int i, int j) {
    l.lock();
    int pont=0;
    for(int k=j-4; k<j+5; k++){
        if(matriz[i][k]==1){
            pont++;
            matriz[i][k]=0;
        }
    }
    int aux = pontuacoes.get(jog);
    aux += pont;
    pontuacoes.put(jog, aux);
    int jogadaJ = jogadas.get(jog);
    jogadaJ++;
    jogadaJ.put(jog, jogadaJ);
    int minhaVez=vez;
    if (jogadaJ==3) {
        while (!acabou()){
            l.unlock();
            e.await();
            l.lock();
        }
    }
    if(vez==minhaVez){
        vez++;
        e.SignalAll();
    }
    String res= vencedor();
    l.unlock();
    return res;
}
l.unlock();
return null;
```

aqui o vencedor poderia ser visto sem lock() pois nesta fase ninguém altera nada.

```
public boolean acabou(){
boolean b= true

    for (int j : jogadas.values()){
        if (j<3){
            b=false;
            break;
        }
    }
    return b;
}

public TrataCliente {

    private Socket s
    private Jogo j;
    public TrataCliente(s, j){
        this.s=s;
        this.j= j;
    }

    public void run(){
        String vencau =null;
        ObjectInputStream in= new OIS(s.getIS());
        // Output //   out = new OOS(s.getOS());
        OpsJogo op;
        while(vencau == null){
            op= (OpsJogo) in.readObject();

            try{
                switch(op){
                case disparo:
                    String jog=in.read();
```

```
int i= in.readInt();
int K= in.readInt();

venceu= j.dispara(jog, i, K);
break;

default:
    out.writeUTF("Erro, opção invalida!\n");
    out.flush();
}
}catch(Exception e){}
}
out.writeUTF("Vencedor: " + venceu+"!\n");

try{
    s.shutdownOutput();
    s.shutdownInput();
    s.close();
}catch(Exception e){}
}
}

public class Servidor{

public static void main(){

ServerSocket ss= new ServerSocket(9999);
Jogo j= new Jogo(jogadores);
while(true){

Socket st= ss.accept();
TrabCliente tc= new TrabCliente(j, st);
(new Thread(tc)).start();
}
}
}
```

Aqui poderia terminar o servidor, caso o jogo tivesse acabado!

I

1. A operação de lock() basicamente é um adquirir lock, ou seja se puder adquire prossegue, caso contrário passa (o processo) ao estado bloqueado não consumindo tempo de processador, geralmente colocado num "fila" afeta aquele lock.
A operação de unlock(), seleciona 1 processo da lista de bloqueados afeta aquele lock (caso exista), mudando o seu estado para pronto a executar.

2. São os sistemas P2P, no qual a gestão e utilização é baseada numa rede lógica. Existem os sistemas estruturados, os quais obedecem a uma estrutura específica para a distribuição dos dados, estrutura que é gerida através de uma DHT. Exemplos são o Chord, estrutura em anel e o Can uma estrutura baseada no eixo cartesiano*. Existem ainda os não estruturados, nos quais a interação entre pares é feita de forma aleatória, onde o funcionamento se segue por cada nó ter uma vista parcial da rede com um conjunto razoavelmente pequeno de nós no qual periodicamente cada nó P escolhe 1 nó Q (da sua vista parcial) e estes trocam nós da sua vista parcial, sendo a aleatoriedade com que as vistas parciais são mantidas um fator crucial para o funcionamento e robustez do sistema.

3. Assegurar exclusão mútua entre processos a executar numa única máquina é assegurado pelo SO com o auxílio de primitivas. No caso dum sistema distribuído pode ser feito de uma forma centralizada na qual existe 1 coordenador e sempre que 1 processo pretenda aceder à sua zona crítica deverá enviar uma mensagem ao coordenador, sendo que este apenas responde quando o processo o possa fazer. Também pode ser realizado de uma forma descentralizada com etiquetas temporais enviadas a todos os processos apenas podendo aceder ao receber as respostas de todos esses processos. Existe ainda uma forma com uma estrutura em anel e passagem de tokens. Num sistema distribuído existem problemas de falha do coordenador, falha de um processo ou "perda" do token em cada caso, respetivamente.

* → baseada no routing da Internet!