Sistemas Distribuídos

Universidade do Minho

2011/2012

lock/ monitor Classe

lock/ monitor Instância

```
public class Contador {
    private long i=0;
    private static long i2=0;

    public long get() return i;
    public synchronized void inc() i++;

    public synchronized static void incStatic() i2++;
    public static long getStatic() return i2;
}
```

```
C = new Contador();
Thread1(C);
Thread2(C);
```

```
public void run(){
        for(int i=0; i<1000000; i++){
            c.incStatic();
            this.c.inc();
        }
    }
```

Aula 3: Exclusão Mútua

Main

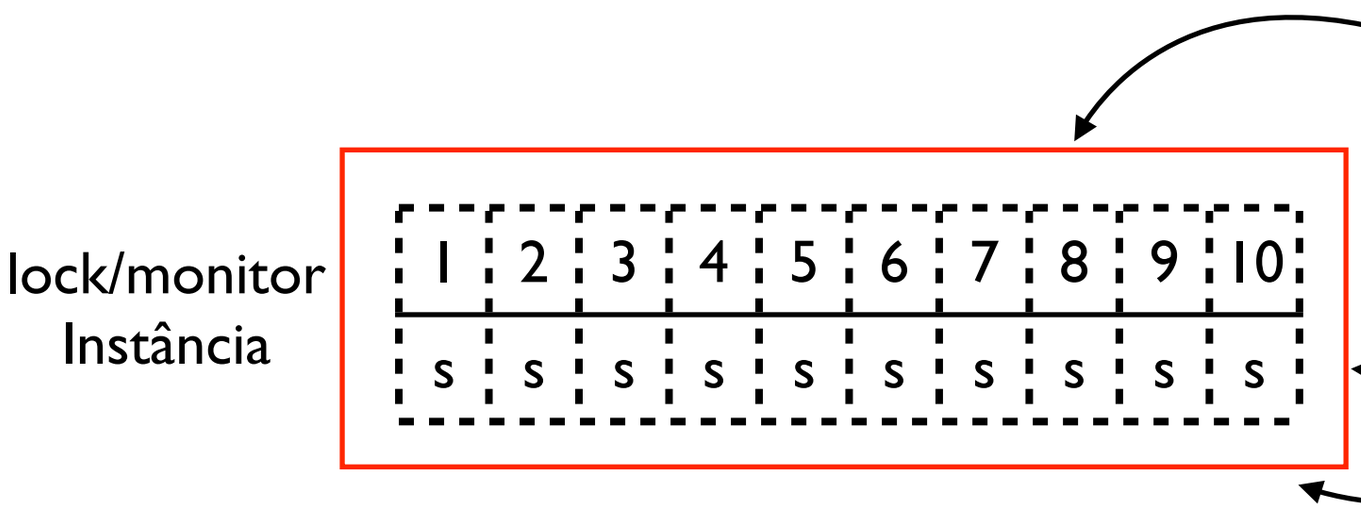
```
Banco banco = new Banco(10);
Thread t[] = new Thread[10];
        for(int i = 0; i < nThreads; i++)
        {
            t[i] = new Thread(new Acesso(banco));
            t[i].start();
        }
```

Acesso implements Runnable

```
public void run() {
        for(int i=0;i<nOps;i++)
        {
            for(int conta=0; conta<nContas; conta++)
                banco.credita(conta,10);
            for(int conta=0; conta<nContas; conta++)
                banco.debita(conta,5);
        }
```

Banco

```
private double[] contas
synchronized void credita(int conta,double
valor){
        contas[conta]+=valor; }
synchronized void debita(int conta,double
valor){
        contas[conta]-=valor; }
void transfere(int contaOrigem,int
contaDestino,double valor){
        this.debita(contaOrigem, valor);
        this.credita(contaDestino, valor);
    }
```



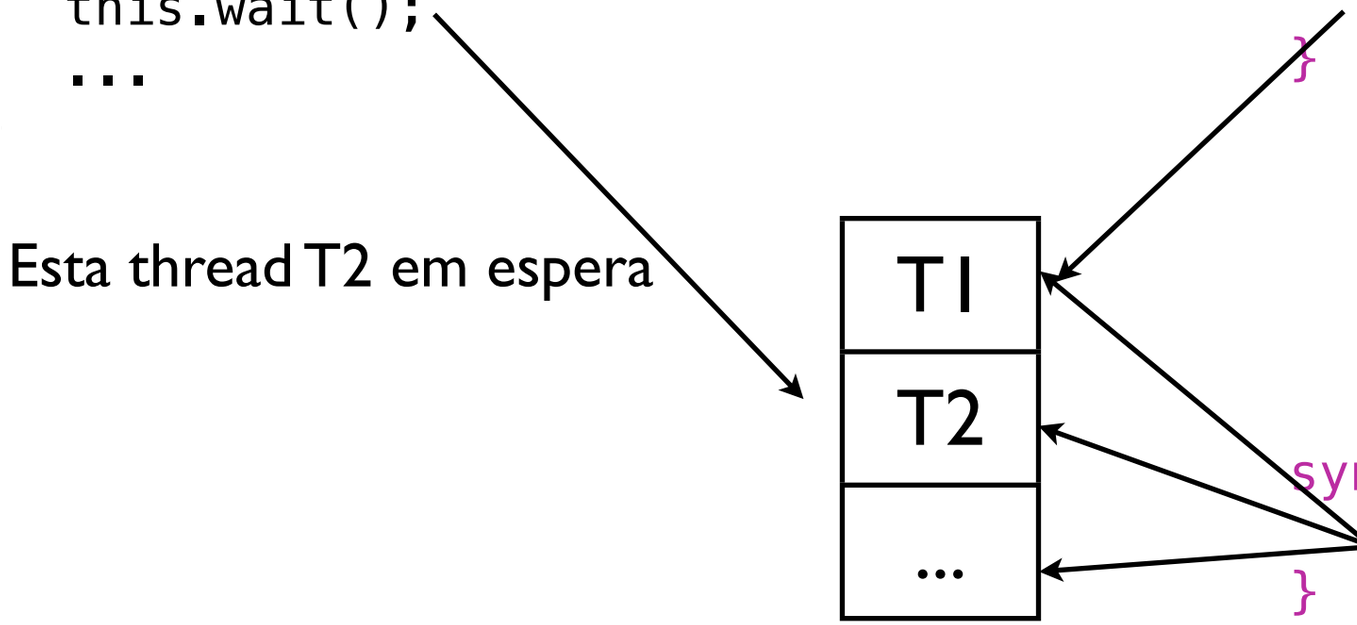
- Variáveis de condição:
  - suspensão/retoma de execução dentro de zona crítica;
  - mecanismo intrínseco de variável de condição em todos os objectos;
  - métodos wait(), notify(), notifyAll();

- Exemplo:

```
synchronized void metodo(){
...
while(condição){
   this.wait();
   ...
}
```

```
synchronized void metodo2(){
   this.notify();
}
```

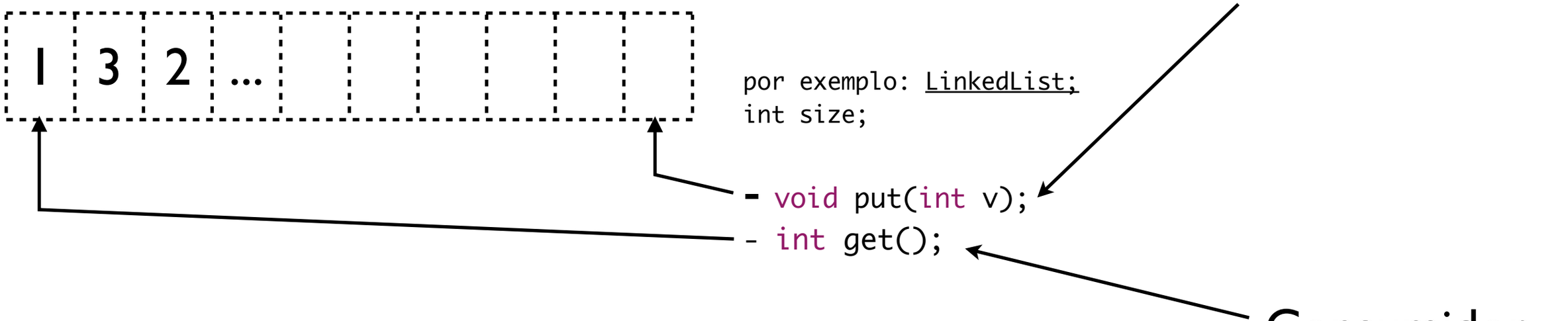
```
synchronized void metodo3(){
   this.notifyAll();
}
```



Aula 4: Exercicio 1

- Implemente uma classe BoundedBuffer que ofereça as operações void put(int v) e int get() sobre um array cujo tamanho é definido no momento da construção de uma instância. O método put() deverá bloquear enquanto o array estiver cheio e o método get() deverá bloquear enquanto o array estiver vazio. Os métodos oferecidos podem estar sujeitas a invocações de threads concorrentes sobre uma instância partilhada. A classe BoundedBuffer deverá garantir a correcta execução em cenário multi-thread.

Aula 4: Exercicio 1



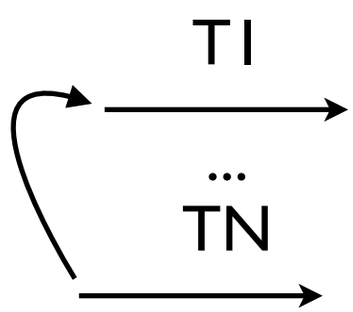
Condições a respeitar:

- put(v) -> Bloquear enquanto o array estiver cheio;
- get() -> Bloquear enquanto o array estiver vazio.

Usando variáveis de condição e exclusão mútua;

- Implemente uma classe Barreira que ofereça um método esperar() cujo objectivo é garantir que cada thread que o invoque se bloqueie até que o número de threads nesta situação tenha atingido o valor N passado ao construtor de uma sua instância.

Bloqueio



Barreira

```
int N;
- void esperar()
```

Usando variáveis de condição
e exclusão mútua;

Aula 4: Exercicio 3

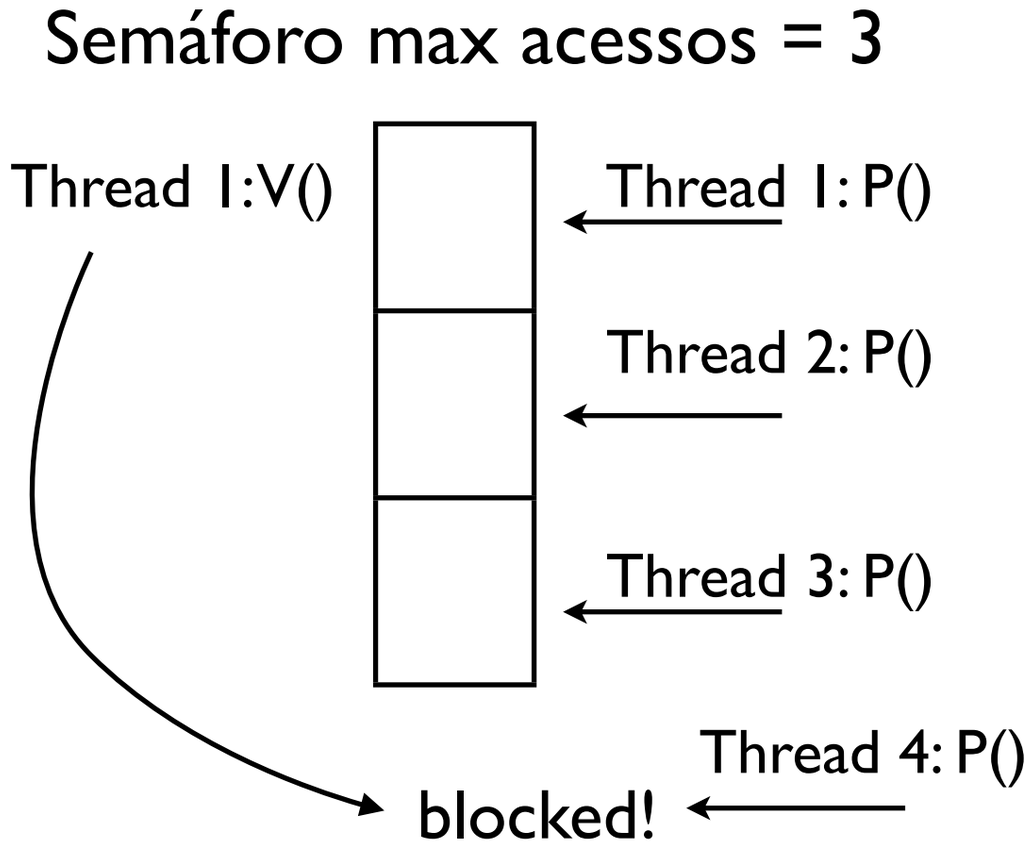
Semáforos:

- controlar o acesso a um recurso a um número limitado de processos/threads, por forma a tentar evitar *race conditions* e possíveis *deadlocks*.

- binários i.e. 0/1 mesma funcionalidade mutex mas no caso do mutex geralmente só quem o adquiriu é que o pode libertar;
- tamanho máximo arbitrário;

Operações:

- `void P();` -> adquire o recurso
  Se não conseguir adquirir o recurso espera por ele.
- `void V();` -> liberta o recurso e avisa outros que estejam à espera.



- Implemente o mecanismo tradicional de semáforos à custa do mecanismo de variáveis de condição.