

Computação Concorrente (DCC/UFRJ)

Aula 11: Exercícios com semáforos e monitores

Prof. Silvana Rossetto

19 de janeiro de 2016

Problema do “barbeiro dorminhoco”



Uma configuração do problema

- **1 barbeiro** e uma área de espera que pode acomodar **5 clientes** sentados
- Um cliente não pode entrar na barbearia se a sala estiver cheia
- Quando o barbeiro fica livre, um dos clientes esperando é atendido
- O barbeiro divide o tempo entre cortar cabelo e dormir esperando por clientes

Solução para o problema usando semáforos

```
int esperando = 0;  
sem_t em, barbeiro, clientes;  
sem_init(&em,0,1); sem_init(&barbeiro,0,0); sem_init(&clientes,0,0);
```

```
void Barbeiro() {  
    while(1) {  
        sem_wait(&clientes);  
        sem_wait(&em);  
        esperando = esperando - 1;  
        sem_post(&barbeiro);  
        sem_post(&em);  
        //corta o cabelo do cliente  
    }  
}
```

```
void Cliente() {  
    sem_wait(&em);  
    if (esperando < CADEIRAS) {  
        esperando = esperando + 1;  
        sem_post(&clientes);  
        sem_post(&em);  
        sem_wait(&barbeiro);  
        //senta na cadeira do barbeiro  
    } else sem_post(&em);  
}
```

¹Fonte: Stallings, 2009.

O problema da barbearia usando monitores (v1) (incompleto)

```
esperando = 0;    //max = 5  
ocupado = 0;      //0: cadeira livre; 1: cadeira ocupada
```

```
public synchronized boolean SentaCadeira () {  
    if(_____) return false;  
    esperando++;  
    while (_____) wait();  
    ocupado = 1;  
    _____;  
    return true;  
}
```

```
public synchronized void EsperaCliente () {  
    while ((_____) || (_____)) wait();  
    esperando--;  
}
```

```
public synchronized void TerminaCliente () {  
    ocupado = 0;  
    if (_____) notifyAll();  
}
```

O problema da barbearia usando monitores (v1) (completo)

```
esperando = 0;    //max = 5  
ocupado = 0;      //0: cadeira livre; 1: cadeira ocupada
```

```
public synchronized boolean SentaCadeira () {  
    if(esperando == 5) return false;  
    esperando++;  
    while (ocupado == 1) wait();  
    ocupado = 1;  
    notifyAll();  
    return true;  
}
```

```
public synchronized void EsperaCliente () {  
    while ((esperando == 0) || (ocupado == 0)) wait();  
    esperando--;  
}
```

```
public synchronized void TerminaCliente () {  
    ocupado = 0;  
    if (esperando > 0) notify();  
}
```

O problema da barbearia usando monitores (v1)

Problema dessa solução

Se um cliente chegar na barbearia no instante em que o barbeiro acabou de cortar o cabelo, ele pode conseguir passar a frente dos clientes que estão na fila de espera (**por que?**)

Altere o código anterior para corrigir o problema detectado

O problema da barbearia usando monitores (v2)

```
esperando = 0;    //max = 5
ocupado = 0;      //0: cadeira livre; 1: cadeira ocupada
proximoCliente = 0; //primeiro da fila
ultimoCliente = 0; //ultimo da fila
```

```
public synchronized boolean SentaCadeira () {
    if(esperando == 5) return false;
    int senha = ultimoCliente++; esperando++;
    while (proximoCliente != senha) wait();
    ocupado = 1; notifyAll(); return true;
}
```

```
public synchronized void EsperaCliente () {
    while ((esperando == 0) || (ocupado == 0)) wait();
    esperando--;
}
```

```
public synchronized void TerminaCliente () {
    ocupado = 0; proximoCliente++;
    if (esperando > 0) notifyAll();
}
```

Leitores e escritores com monitores

```
class Leitor extends Thread {  
    //objeto monitor para coordenar a lógica de execução  
    LE monitor;  
    Leitor (LE m) {  
        this.monitor = m;  
    }  
    public void run () {  
        try {  
            for (;;) {  
                this.monitor.EntraLeitor();  
                // !!! parte do código que faz a leitura !!!  
                this.monitor.SaiLeitor();  
            }  
        } catch (InterruptedException e) { return; }  
    }  
}
```

Leitores e escritores com monitores

```
class Escriitor extends Thread {
    //objeto monitor para coordenar a lógica de execução
    LE monitor;
    Escriitor (LE m) {
        this.monitor = m;
    }
    public void run () {
        try {
            for (;;) {
                this.monitor.EntraEscriitor();
                // !!! parte do codigo que faz a escrita !!!
                this.monitor.SaiEscriitor();
            }
        } catch (InterruptedException e) { return; }
    }
}
```

Leitores e escritores com monitores

```
class LeitorEscritor {
    static final int L = 4;
    static final int E = 3;
    public static void main (String[] args) {
        int i;
        LE monitor = new LE(); // monitor
        Leitor[] l = new Leitor[L]; // leitores
        Escritor[] e = new Escritor[E]; // escritores
        for (i=0; i<L; i++) {
            l[i] = new Leitor(monitor); l[i].start();
        }
        for (i=0; i<E; i++) {
            e[i] = new Escritor(monitor); e[i].start();
        }
    }
}
```

Implemente o código da classe **LE** com os métodos:

- ① **EntraLeitor**
- ② **SaiLeitor**
- ③ **EntraEscritor**
- ④ **SaiEscritor**

Leitores e escritores com monitores

```
class LE {  
    private int leit, escr;  
  
    // Construtor  
    LE() {  
        this.leit = 0;  
        this.escr = 0;  
    }  
    ...  
}
```

```
public synchronized void EntraLeitor() {  
    try {  
        while (this.escr > 0) {  
            wait();  
        }  
        this.leit++;  
    } catch (InterruptedException e) { }  
}
```

```
public synchronized void SaiLeitor() {  
    this.leit--;  
    if (this.leit == 0)  
        notify(); //libera escritores  
}
```

Leitores e escritores com monitores

```
public synchronized void EntraEscritor () {
    try {
        while ((this.leit > 0) || (this.escr > 0)) {
            wait();
        }
        this.escr++;
    } catch (InterruptedException e) { }
}

public synchronized void SaiEscritor () {
    this.escr--;
    notifyAll(); //libera leitores
}
```


Exercício: o problema do “banheiro unissex”

Um escritório contém um banheiro que deve ser usado por homens e mulheres, mas não por ambos ao mesmo tempo. Se um (ou mais) homem está no banheiro, outros homens podem entrar, as mulheres devem esperar até o banheiro ficar vazio. Se uma (ou mais) mulher está no banheiro, outras mulheres podem entrar, os homens devem esperar até o banheiro ficar vazio.

- 1 Implemente uma classe (Monitor) Java para gerenciar o acesso ao banheiro oferecendo 4 métodos: **EntraHomem()**, **SaiHomem()**, **EntraMulher()**, **SaiMulher()**.
- 2 A solução deve permitir qualquer número de homens ou qualquer número de mulheres (mas não ambos) no banheiro ao mesmo tempo, e garantir ausência de **deadlock**.

Exercício: o problema do “banheiro unissex”

- 1 Verifique se a sua solução para o problema do “banheiro unissex” garante ausência de **starvation**, se não, o que precisaria ser alterado na sua solução?
- 2 Reimplemente o problema se necessário.

Referências bibliográficas

- ① *Programming Language Pragmatics*, Scott, Morgan-Kaufmann, ed. 2, 2006
- ② *Operating Systems – Internals and Design Principles*, Stallings, Pearson, ed. 6, 2009