

# Computação Concorrente (DCC/UFRJ)

## Aula 9: Resolução de problemas clássicos de concorrência usando monitores

Prof. Silvana Rossetto

15 de maio de 2012

- 1 Implementando monitores com semáforos
  - Barreira usando monitor
- 2 Problemas de concorrência usando monitores
  - Produtor/Consumidor
  - Leitor/Escritor
  - Barbeiro dorminhoco
  - Gerente de recursos

## Interface de monitores

```
define Monitor {  
    void entra_monitor() //entra no monitor  
    void deixa_monitor() //sai do monitor  
    void wait() //bloqueia a thread  
    void notify() //acorda uma thread do monitor  
    void notifyAll() //acorda todas as threads do monitor  
}
```

# Implementando monitores com semáforos

```
define Monitor
  sem mutex=1; //semaforo para exclusao mutua no monitor
  sem cond=0;  //semaforo para bloqueio por condicao
  int n=0;     //numero de threads bloqueadas no monitor

  void entra_monitor() { // entra no monitor
    ???
  }

  void deixa_monitor() { // sai do monitor
    ???
  }
  (...)
```

# Implementando monitores com semáforos

```
define Monitor
    sem mutex=1; //semaforo para exclusao mutua no monitor
    sem cond=0;  //semaforo para bloqueio por condicao
    int n=0;     //numero de threads bloqueadas no monitor

    void entra_monitor() { // entra no monitor
        down(mutex); //decrementa o semaforo
    }

    void deixa_monitor() { // sai do monitor
        up(mutex) //incrementa o semaforo
    }
    (...)
```

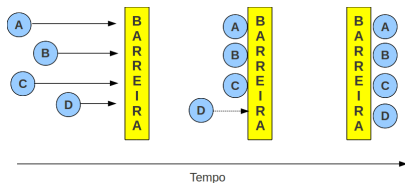
# Implementando monitores com semáforos

```
(...)  
void wait() { //executado dentro do monitor!  
    ???  
}  
  
void notify() { //executado dentro do monitor!  
    ???  
}  
  
void notifyAll() { //executado dentro do monitor!  
    ???  
}
```

# Implementando monitores com semáforos

```
(...)  
void wait() {  
    n++; up(mutex); //libera o monitor  
    down(cond); //bloqueia na variavel de condicao  
    down(mutex); //tenta alocar o monitor novamente  
}  
  
void notify() {  
    if(n > 0) { n--; up(cond); }  
}  
  
void notifyAll() {  
    for(int i=0; i++; i<n) up(cond);  
    n=0;  
}
```

## Barreira usando monitor (incompleto)



```
class Barreira {  
    private int numThreads;  
    private int cont;
```

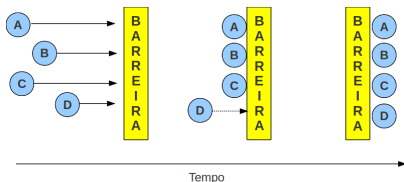
```
//...construtor
```

```
Barreira(int n) {  
    this.cont = 0;  
    this.numThreads = n;  
}
```

```
public synchronized void Chegada () {  
    _____  
    _____  
    _____  
    ...  
}
```



## Barreira usando monitor (completo)



```
class Barreira {  
    private int numThreads;  
    private int cont;
```

```
//...construtor
```

```
Barreira(int n) {  
    this.cont = 0;  
    this.numThreads = n;  
}
```

```
public synchronized void Chegada () {  
    cont++;  
    if (cont < numThreads) wait();  
    else {  
        cont = 0;  
        notifyAll();  
    }  
}
```

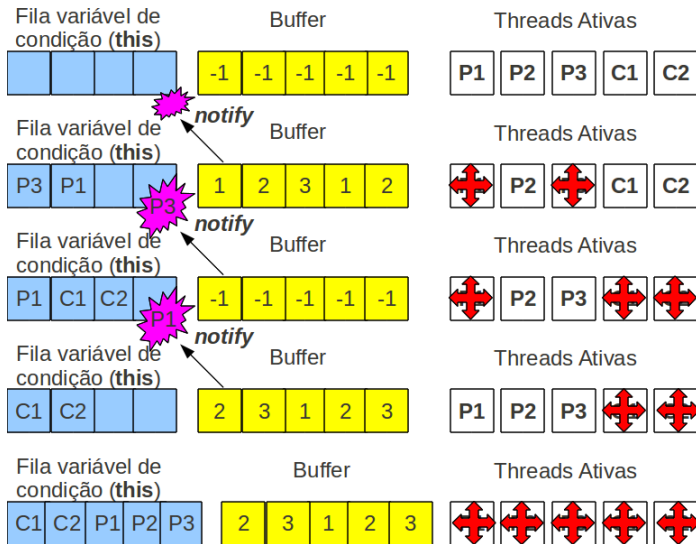
## P/C usando monitores (incorreto!)

```
int[] buffer = new int[N];  
int in = 0; out = 0; count = 0;
```

```
public synchronized void Inserir (int x){  
    while (count==N) wait();  
    buffer[in%N] = x;  
    in++; count++;  
    if (count == 1) notify();  
}
```

```
public synchronized int Remove () {  
    while (count==0) wait();  
    int aux = buffer[out%N];  
    buffer[out%N] = -1;  
    out++; count--;  
    if (count == (N-1)) notify();  
    return aux;  
}
```

# Exemplo de situação de deadlock



## L/E usando monitores (incompleto)

```
int leitores=0, escritor=0;


public synchronized void PreLeitura () {
    while ( _____ ) wait();
    leitores++;
}
public synchronized void PosLeitura () {
    leitores--;
    if ( _____ ) notify();
}
public synchronized void PreEscrita () {
    while ( _____ ) wait();
    escritor = 1;
}
public synchronized void PosEscrita () {
    escritor = 0; _____ ;
}
```

## L/E usando monitores (completo)




```
int leitores=0, escritor=0;

public synchronized void PreLeitura () {
    while (escritor != 0) wait();
    leitores++;
}

public synchronized void PosLeitura () {
    leitores--;
    if (leitores == 0) notify(); 
}

public synchronized void PreEscrita () {
    while ((leitores > 0) || (escritor != 0)) wait();
    escritor = 1;
}

public synchronized void PosEscrita () {
    escritor = 0; notifyAll(); 
}
```

# Barbeiro dorminhoco 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();  
}
```

# Barbeiro dorminhoco 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) notifyAll();  
}
```

# Barbeiro dorminhoco 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?**)



## Barbeiro dorminhoco 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();  
}
```

## Como funciona esse monitor?



```
pool = new int[N]; count = N;

public synchronized void A(int unidade) {
    if(count==N) return;
    pool[in%N] = unidade;
    in++; count++;
    if (count == 1) notify();
}

public synchronized int B(int threadid) {
    while (count==0) wait();
    int unidade = pool[out%N];
    pool[out%N] = 0;
    out++; count--; return unidade;
}
```

## Referências bibliográficas

- 1 *Sistemas Operacionais Modernos*, Tanenbaum, PHP, 1992
- 2 *Synchronization Algorithms and Concurrent Programming*, G. Taubenfeld, Pearson/Prentice Hall, 2006