The background features a collage of various vintage postage stamps and postmarks. Visible text includes "PAR AVION", "02293", "MADRID", "COSTA RICA", and "CHITRAWA". There are also numerical values like "100.00" and "100.00".

Monitores: conceito, exemplo e implementação

Alcides Calsavara

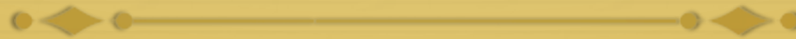
Referências Bibliográficas



Foundations of Multithreaded, Parallel, and Distributed Programming. Gregory R. Andrews. Addison Wesley Longman, Inc., 2000. (Capítulo 5)

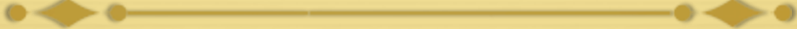
Principles of Concurrent and Distributed Programming. M. Ben-Ari. Prentice Hall, 1990.

Conceito



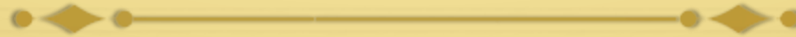
Monitores

Visão Geral



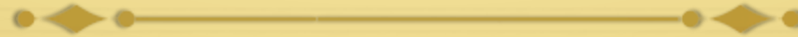
- ✦ Mecanismo de mais alto nível de abstração que semáforos
 - ✦ melhora a escrita, a legibilidade e a manutenção do código
 - ✦ reduz a chance de erros de programação
- ✦ Pode ser implementado de forma tão eficiente quanto semáforos
- ✦ Mecanismo de abstração de dados: encapsulamento
 - ✦ um monitor mantém o *estado de um objeto*, e esse estado só é acessível através de *operações* (procedimentos, métodos) do monitor
 - ✦ um processo acessa as variáveis mantidas por um monitor somente através de chamadas de operações desse monitor
- ✦ Exclusão mútua é implícita: há somente uma execução de operação do monitor por vez

Uso



- ✦ Um monitor é compartilhado por processos concorrentes.
- ✦ Toda a comunicação entre os processos ocorre através do monitor.
- ✦ Os processos que executam em um monitor o fazem com garantia de *exclusão mútua*: isso evita interferência entre processos.
 - ✦ as operações de um monitor são implicitamente exclusivas
- ✦ Os processos que executam em um monitor podem requerer *sincronização condicional*: suspender a execução até que o estado do monitor fique adequado aos requisitos da aplicação.

Variável Condicional



- ✧ Usada para suspender um processo que não pode continuar executando com garantia da semântica da aplicação até que o estado do monitor satisfaça alguma condição booleana.
 - ✧ Também é usada para acordar um processo que esteja suspenso, quando a condição torna-se verdadeira.
- ✧ Sintaxe (pseudocódigo): **cond** *cv*
- ✧ Semântica: uma fila de processos suspensos (política FIFO)
- ✧ Operações:
 - ✧ **empty**(*cv*) : verifica se a fila está vazia
 - ✧ **wait**(*cv*) : bloqueia o processo (insere o processo na fila)
 - ✧ **signal**(*cv*) : acorda o *primeiro* processo da fila
 - ✧ **signal_all**(*cv*) : acorda *todos* os processos da fila

Sintaxe

DEFINIÇÃO DE UM MONITOR: (como um objeto estático)

```
monitor nome_do_monitor {  
    declaração de variáveis permanentes  
    comando de iniciação  
    procedimentos  
}
```

CHAMADA DE OPERAÇÃO POR UM PROCESSO:

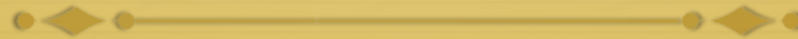
```
chame nome_do_monitor.nome_da_operação(argumentos)
```

Propriedades Semânticas



1. Somente os nomes de operações são visíveis externamente ao monitor.
2. Comandos (em iniciação e em operações) dentro do monitor não podem acessar variáveis externas ao monitor.
3. Variáveis permanentes são iniciadas antes de qualquer chamada de operação.

Problema do Buffer Limitado



Exemplo

```

monitor BufferLimitado {
    int buf[n];
    int front = 0, rear = 0, count = 0;
    ## rear == (front + count) % n;

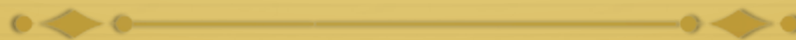
    cond not_full,    # sinalizado quando count < n
        not_empty; # sinalizado quando count > 0

    procedimento depositar(int dado)
    {
        enquanto ( count == n ) wait( not_full );
        buf[rear] = dado; rear = (rear+1) % n; count++;
        signal( not_empty );
    }

    procedimento retirar(int& dado)
    {
        enquanto ( count == 0 ) wait( not_empty );
        dado = buf[front]; front = (front+1) % n; count--;
        signal( not_full );
    }
}

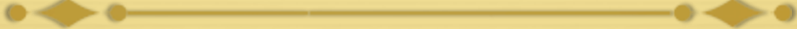
```

Implementação



Monitores

Monitor em Java



- ✧ Um monitor é implementado como uma classe
 - ✧ cada instância da classe é um monitor
- ✧ Abordagens:
 1. **Sem** variáveis condicionais
 2. **Com** variáveis condicionais

Monitor em Java

sem variáveis condicionais

- ✧ Há uma **fila única** de threads suspensas associada ao monitor
- ✧ Métodos do monitor são qualificados como **synchronized**
 - ✧ exclusão mútua implícita
- ✧ Métodos herdados da classe **Object**
 - ✧ **wait**: suspende a thread corrente e a insere na fila do monitor
 - ✧ **notify**: remove e acorda a primeira thread da fila do monitor (equivalente ao **signal**)
 - ✧ **notifyAll**: remove e acorda todas as threads da fila do monitor (equivalente ao **signal_all**)

Monitor em Java

com variáveis condicionais

- ✧ Há uma **fila** de threads suspensas para cada variável condicional
- ✧ Métodos do monitor são programados para exclusão mútua com uso de uma instância da classe **ReentrantLock**
 - ✧ variável do monitor: **Lock** *mutex* = **new ReentrantLock();**
 - ✧ início de cada método: *mutex.lock();*
 - ✧ fim de cada método: *mutex.unlock();*
- ✧ Definição de uma variável condicional: interface **Condition**
Condition *condicao* = *mutex.newCondition();*
- ✧ Métodos da interface **Condition**:
 - ✧ *condicao.await*: suspende a thread corrente e a insere na fila da variável condicional
 - ✧ *condicao.signal*: remove e acorda a primeira thread da fila da variável condicional
 - ✧ *condicao.signalAll*: remove e acorda todas as threads da fila da variável condicional