Programação Concorrente

Monitores

Prof. Rodrigo Campiolo

UTFPR - Universidade Tecnológica Federal do Paraná
DACOM - Departamento de Computação
BCC - Bacharelado Ciência da Computação

18 de setembro de 2018

Introdução

Monitor é um construtor para sincronização de programas concorrentes, orientado a objetos e de alto nível, que provê exclusão mútua e coordenação entre processos.

Monitores

- Um monitor pode ser visto como uma classe, composta por atributos, métodos que operam sobre esses atributos, sequência de inicialização.
- A ideia central é encapsular os dados compartilhados nessa classe e coordenar o acesso a esses dados.
- Os métodos do monitor garantem a exclusão mútua, logo apenas *uma thread* executa no monitor por vez.
- As threads que estão aguardando a entrada no monitor, se encontram em um fila.

Monitores

- Os monitores suportam a noção de *variáveis de condição*.
- As variáveis de condições possibilitam criar condições de sincronização.
- As variáveis de condições têm duas operações: *wait* e *notify* (*signal*)
- Seja x uma variável de condição,
  wait(x) faz com que uma thread seja bloqueada e inserida em um fila associada a x;
  notify(x) faz com que uma thread seja acordada e inserida na fila de execução;

Monitores

- Apenas uma thread pode estar no monitor, logo surge a seguinte pergunta:
  Qual thread deveria continuar após a operação de notify?
- Há duas possíveis respostas:
  1. Uma das threads que estava aguardando a variável de condição será executada (*Hoare Monitors*).
  2. A thread que executou o *notify* continua a execução e, após deixar o monitor, as outras disputam o acesso (*Java*).

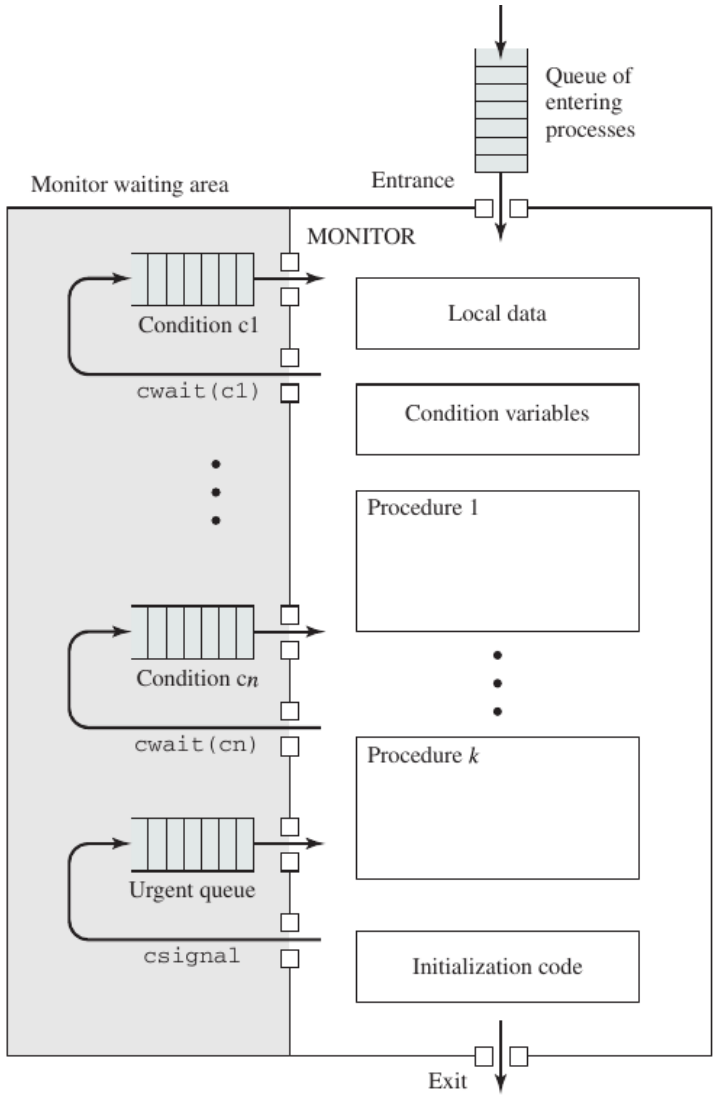

Figura: Estrutura de um monitor. (Fonte: Stallings)

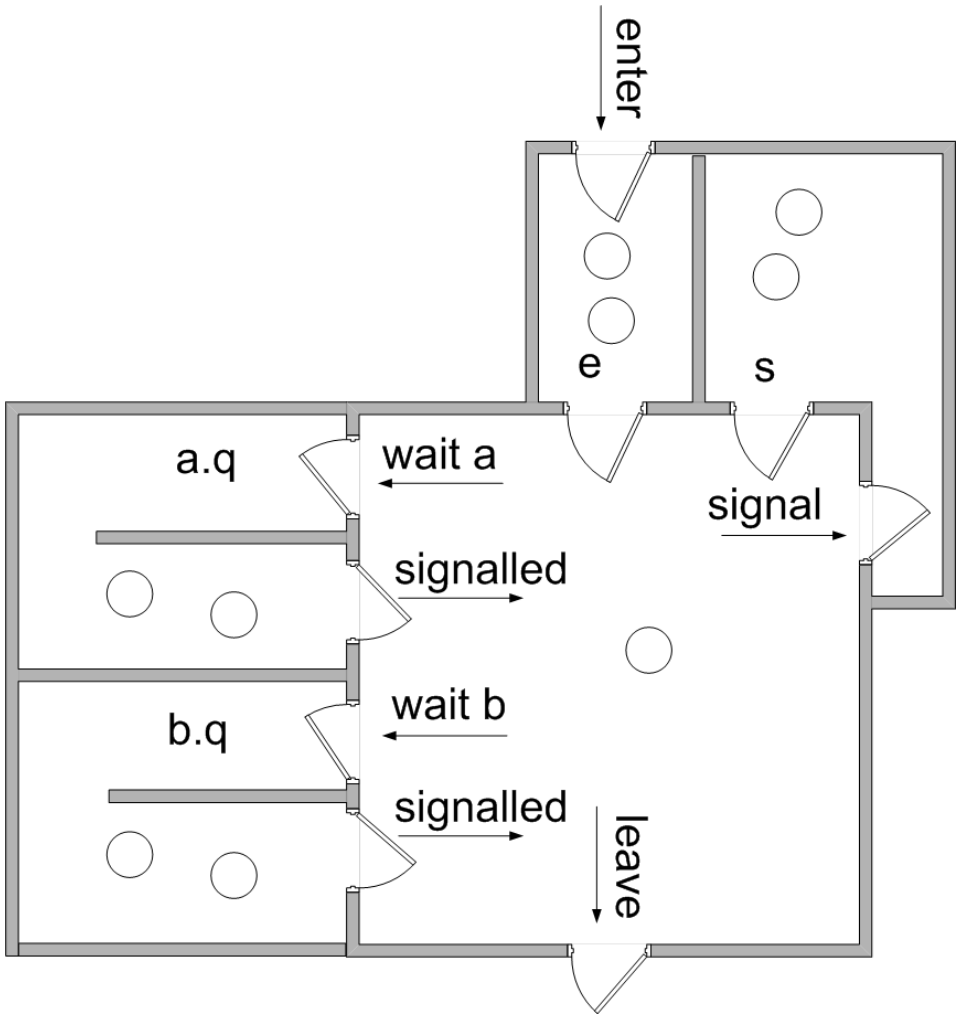

Figura: Estrutura de um monitor de Hoare. (Fonte: Wikipedia)

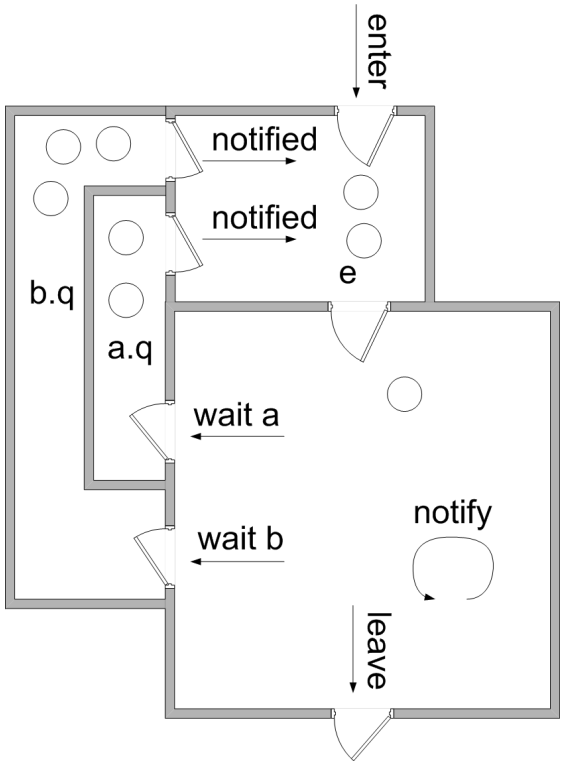

Figura: Estrutura de um monitor de Mesa. (Fonte: Wikipedia)

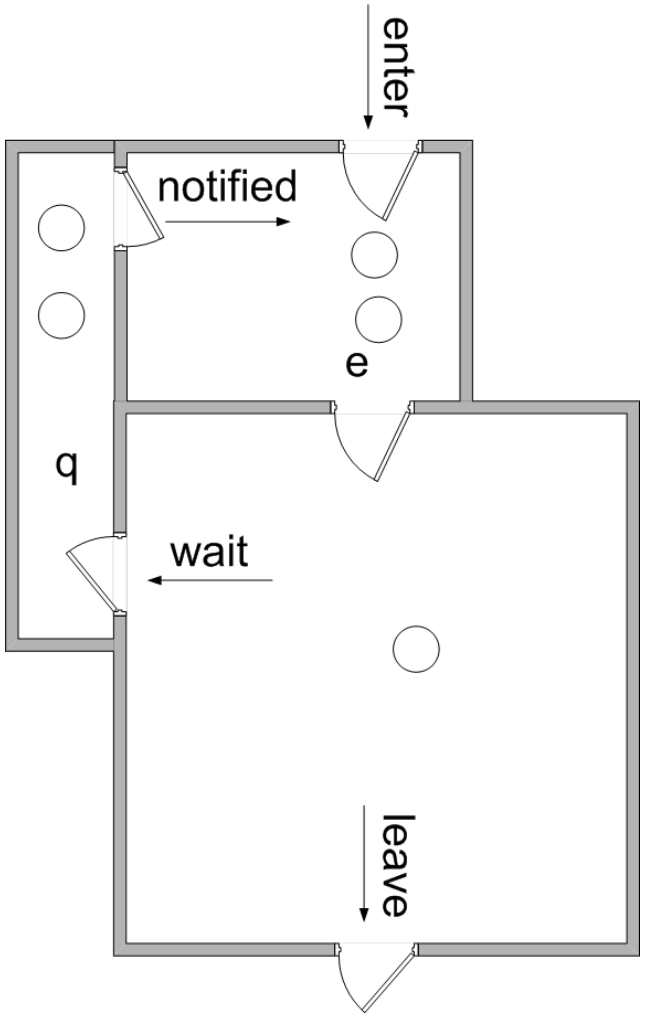

Figura: Estrutura de um monitor de Java. (Fonte: Wikipedia)

Produtor-Consumidor (Buffer Simples) usando Monitor

```
public class ResourceMonitor { //versao 1
    private String buffer;
    private boolean empty = true;

    private final Object objMonitor = new Object();

    public String take() {
        synchronized (objMonitor) {
            while (empty) {
                try {
                    objMonitor.wait();
                } catch (InterruptedException e) {}
            }
            empty = true;
            objMonitor.notifyAll();
            return buffer;
        }
    }

    public void put(String message) {
        synchronized (objMonitor) {
            while (!empty) {
                try {
                    objMonitor.wait();
                } catch (InterruptedException e) {}
            }
            empty = false;
            this.buffer = message;
            objMonitor.notifyAll();
        }
    }
}
```

```
public class ResourceMonitor { //versao 2
    private String message;
    private boolean empty = true;

    public synchronized String take() {
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = true;
        notifyAll();
        return message;
    }

    public synchronized void put(String message) {
        while (!empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = false;
        this.message = message;
        notifyAll();
    }
}
```

Atividades

- Implemente uma solução com monitor para o problema do Produtor-Consumidor usando um buffer circular.
- Escreva uma monitor Counter que possibilita um processo dormir até o contador alcançar um valor. A classe Counter permite duas operações: *increment()* e *sleepUntil(int x)*.
- Escreva um monitor BoundedCounter que possui um valor mínimo e máximo. A classe possui dois métodos: *increment()* e *decrement()*. Ao alcançar os limites mínimo ou máximo, a thread que alcançou deve ser bloqueada.
- Implemente uma solução para o problema do Barbeiro Dorminhoco usando monitores.

Referências

- Monitor (synchronization). Wikipedia. `https://en.wikipedia.org/wiki/Monitor_(synchronization)`. Acessado em 09/2018.
- Garg, K. Vijay. **Concurrent and Distributed Computing in Java**. Capítulo 3: Syncronization Primitives. 2004.