Programação Concorrente

Semáforos

Prof. Rodrigo Campiolo

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

24 de setembro de 2018

Introdução

- Conceito proposto por Dijkstra para o problema da espera ocupada.
- Semáforos limitam o número de threads concorrentes que podem acessar um recurso.
- Um semáforo é composto por 2 campos, **valor** e **fila de processos bloqueados**, e duas operações, **P()** e **V()**.
- As operações P() e V() são de origem alemã: Probeer ('Try') e Verhoog ('Increment'). [1]
- Em geral, são também usados os pares: down() e up(), wait() e signal(), acquire() e release(), etc.

[1] https://cs.nyu.edu/~yap/classes/os/resources/origin_of_PV.html

Introdução

- O uso de semáforo pode ser comparado ao controle de entrada de um restaurante por meio de Fichas.
- Nesse restaurante, há um número de Fichas (Tokens) equivalente ao número de mesas.
- Quando o cliente chega, solicita a Ficha e, se houver Fichas disponíveis, o cliente entra e acomoda-se.
- Quando o cliente saí, devolve a Ficha e, dessa forma, libera uma mesa.
- Se não há Ficha, o cliente espera em uma Fila a devolução da Ficha pelo primeiro cliente a sair.

Semáforo Binário (*Binary Semaphore*)

- Em um **Semáforo Binário**, o campo **valor** pode assumir **true** ou **false**.
- A **fila de processos bloqueados** é inicialmente vazia.
- Ao executar o P(), se valor é true, torna-se false. O processo ganha a execução.
- Ao executar o P(), se valor é false, o processo é adicionado a fila de processos bloqueados.
- Ao executar o V(), valor é modificado para true e outro processo na fila pode adquirir a execução.

```
public class BinarySemaphore {
    private boolean value;

    public BinarySemaphore(boolean initValue) {
        value = initValue;
    }

    public synchronized void P() {
        while (value == false) {
            try {
                this.wait();
            } catch (InterruptedException ie) {}
        } //while
        value = false;
    }

    public synchronized void V() {
        value = true;
        this.notify();
    }
}
```

Exclusão Mútua com Semáforo

```
BinarySemaphore mutex = new BinarySemaphore(true);
mutex.P();
criticalSection();
mutex.V();
```

Semáforo por Contagem (*Counting Semaphore*)

- Em um **Semáforo por Contagem**, o campo **valor** assume valores inteiros entre 0 e MAX.
- MAX pode indicar o número máximo de recursos ou o número máximo de acessos simultâneos.
- Ao executar o P(), se valor é não zero, valor é decrementado e o processo ganha a execução.
- Ao executar o P(), se valor é zero, o processo é adicionado na fila de processos bloqueados.
- Ao executar o V(), valor é incrementado.

```
public class CountingSemaphore {
    private int value;

    /* 0 < initValue <= MAX */
    public CountingSemaphore(int initValue) {
        value = initValue;
    }

    public synchronized void P() {
        while (value == 0) {
            try {
                this.wait();
            } catch (InterruptedException ie) {}
        } //while
        value = value - 1;
    }

    public synchronized void V() {
        value = value + 1;
        this.notify();
    }
}
```

Atividades

1. Faça a implementação do problema do produtor-consumidor usando Semáforos.

Classe **Semaphore**

- A classe **Semaphore** do pacote **java.util.concurrent** implementa um semáforo por contagem.
- Possui duas opções de construtores:

```
/* Semáforo não justo (nonfair) */
   Semaphore(int permits)

/* Semáfor justo (fair) */
   Semaphore(int permits, boolean fair)
```

- As duas principais operações são:

```
/* Obtem uma permissão do semáforo ou bloqueia a thread */
void acquire()

/* Libera uma permissão devolvendo ao semáforo. */
void release()
```

- A thread é desbloqueada ao obter permissão do semáforo ou quando é interrompida.

Exemplo - classe **Semaphore**

```
public class SemaphoreExample {

    private static final int MAX_PERMITS = 5;
    /* fair semaphore */
    private final Semaphore semaphore = new Semaphore(MAX_PERMITS, true);

    public SemaphoreExample(int numberThreads) {
        for (int i = 0; i < numberThreads; i++) {
            new TestSemaphoreThread(semaphore).start();
        }
    } //construtor

    public static void main(String[] args) {
        new SemaphoreExample(10);
    } //main
}
```

```
class TestSemaphoreThread extends Thread {
    private final Semaphore semaphore;

    public TestSemaphoreThread(Semaphore semaphore) {
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        int count = 0;  // thread solicita o recurso 3 vezes
        while (cont < 3) {
            try {
                semaphore.acquire();

                /* doing some long process */
                Thread.sleep(Math.round(Math.random() * 10000));

            } catch (InterruptedException ie) {
                System.err.println("IE: " + ie.getMessage());
            } finally {
                semaphore.release();
            }

            count++;

        } //while
    }
}
```

classe Semaphore - Outros métodos

- **void acquire(int permits)**: bloqueia se número de permissões for menor, adquire se o número for maior ou igual.
- **void acquireUninterruptibly()**: bloqueia ou adquire uma permissão, se bloqueado, somente notificado após o método.
- **int availablePermits()**: devolve o número de permissões disponíveis.
- **int getQueueLength()**: devolve o número de threads aguardando na fila .
- **boolean tryAcquire()**: tenta adquirir uma permissão, devolve *true* se positivo.
- **boolean tryAcquire(long timeout, TimeUnit unit)**: tenta adquirir uma permissão dentro de um período, devolve *true* se positivo.

classe Semaphore - Outros métodos

- A documentação está disponível em:

  `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html`.

Referências

- GOETZ, Brian. **Java concurrency in practice**. Upper Saddle River, NJ.: Addison-Wesley, 2006.
- Jenkov. **Thread Safety and Shared Resources**. Disponível em `http://tutorials.jenkov.com/java-concurrency/thread-safety.html`
- Oracle. **The Java Tutorials - Concurrency**. Disponível em: `https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html`