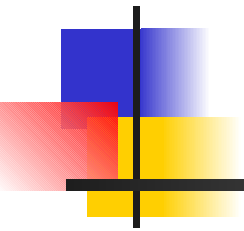
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
DISCIPLINA DE PROGRAMAÇÃO COM OBJETOS DISTRIBUÍDOS

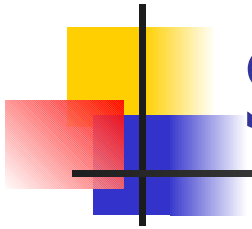


Semáforos:
Abordagem Clássica X Abordagem Java

Luiza Figueiredo Pagliari
Pablo Souza Grigoletti

{lfpagliari, psgrigoletti}@inf.ufrgs.br

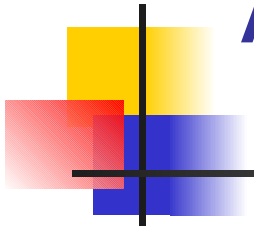
Professor Cláudio Geyer



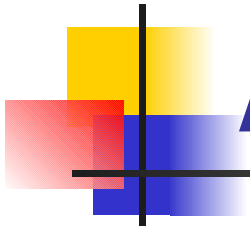
Sumário

- Abordagem Clássica
- Abordagem Java (classe Semaphore)
  - método construtor
  - método acquire() e semelhantes
  - método release()
  - outros métodos...
  - exemplo de utilização
- Comparações
- Bibliografia

Abordagem Clássica
Tanembaum/Silberschatz

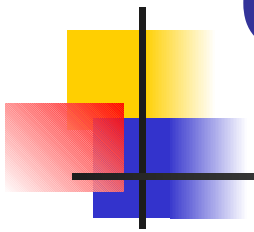


- Dijkstra (1965),
- Usados para restringir o acesso a um recurso ou para o problema da exclusão mútua,
- Consiste em uma variável inteira (valor ≥ 0),
- Inicializada com o nº de recursos disponíveis,
- Só pode ser acessada por 2 primitivas:
  - P (testar e decrementar),
  - V (incrementar),
- As operações devem ser implementadas de forma indivisível,
- Semáforo de contagem ou semáforo binário.



Abordagem Java

- Implementado pela classe Semaphore,
- Composto basicamente de:
  - variável inteira,
  - lista de espera,
- Restringir o número de threads que podem acessar determinado recurso,
- Semáforos binários são usados para o problema de exclusão mútua.

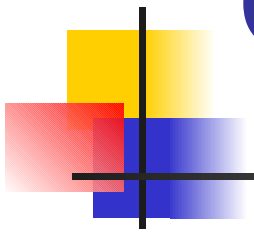


Classe Semaphore

Método Construtor

- `Semaphore(int permits)`
- `Semaphore(int permits, boolean fair)`

- Cria um semáforo com o número de permissões fornecido como parâmetro,
- Se o parâmetro "fair" for:
  - TRUE → a lista de espera é uma FIFO,
  - FALSE (ou não setado) → não é possível garantir a ordem da lista de espera.
- **OBS:** se o número de permissões for negativo as liberações devem ocorrer antes de qualquer permissão ser concedida.



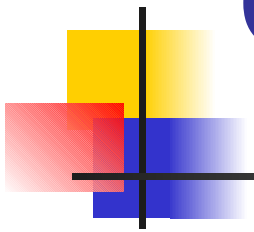
Classe Semaphore

Método acquire()

- **`acquire() // Adquire uma permissão`**
- **`acquire(int permits)`**

- Se existir ao menos uma (ou *n*) permissão disponível:
  - é liberada uma (ou *n*) permissão para a thread,
  - e é reduzido o número de permissões disponíveis no semáforo.

- Senão: a thread fica fora do escalonamento (em estado inativo) até que:
  - alguma outra thread realize uma liberação e esta thread seja a próxima a conseguir uma (ou *n*) permissão,
  - alguma outra thread interrompa a thread atual.



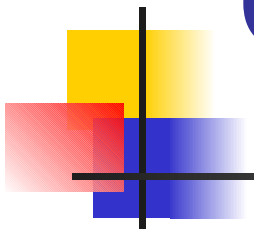
Classe Semaphore

Método acquireUninterruptibly()

- **`acquireUninterruptibly() // Semelhante ao acquire()`**
- **`acquireUninterruptibly(int permits)`**

- Se a thread atual for interrompida enquanto espera por uma permissão:
  - continua na espera,
  - quando a thread retornar deste método ela será interrompida.

Classe Semaphore
Método tryAcquire()



- **`tryAcquire() // Semelhante ao acquire()`**
- **`tryAcquire(int permits)`**

- Se uma (ou *n*) permissão estiver disponível no momento:
  - adquire uma (ou *n*) permissão do semáforo e retorna TRUE,
  - senão retorna FALSE.

- Mesmo se a política de requisição for justa, este método vai adquirir uma (ou *n*) permissão (se houver disponível), estando ou não outras threads esperando (**quebra da política de espera**).

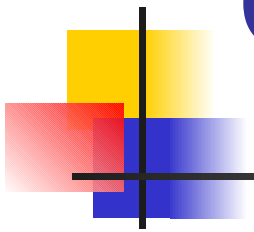
Classe Semaphore
Método tryAcquire()

- **`tryAcquire(long timeout, TimeUnit unit)`**
- **`tryAcquire(int permits, long timeout, TimeUnit unit)`**

- Adquire uma (ou *n*) permissão do semáforo se:
  - uma (ou *n*) ficar disponível no tempo de espera determinado,
  - e a thread atual não seja interrompida.

- Se não existe permissão disponível a thread fica inativa até:
  - alguma outra thread realizar uma liberação e esta thread conseguir uma (ou *n*) permissão,
  - ou o tempo de espera tenha acabado (retorna FALSE),
  - ou a thread atual seja interrompida.

- Para manter a política de espera usa-se:
  - **tryAcquire(0,TimeUnit.SECONDS)**.



Classe Semaphore

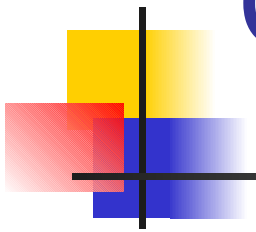
Método release()

- `release() // Libera uma permissão`
- `release(int permits)`

- Libera uma (ou *n*) permissão, a qual retorna para o semáforo,
- Número de permissões do semáforo é incrementado,
- Se algumas threads estavam na lista de espera, uma delas é escolhida para receber a permissão,
- A thread que receber a permissão retorna para o escalonamento.

OBS: Não existe nenhuma obrigação de que a thread que realiza a liberação deve ter adquirido a permissão.

Classe Semaphore
Outros métodos



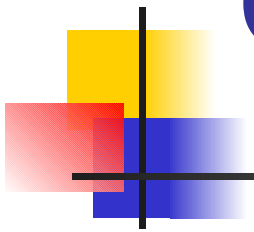
- **`draimPermits()`**
  - adquire todas as permissões que estão disponíveis no semáforo,
  - retorna o número de permissões adquiridas.
- **`reducePermits(int reduction)`**
  - reduz o número de permissões disponíveis,
  - gera uma exceção se o parâmetro "reduction" for negativo.
- **`getQueuedLenght()`**
  - retorna o número "estimado" de threads que estão esperando por permissão.
- **`availablePermits()`**
  - retorna o número de permissões disponíveis no semáforo.
- **`getQueuedThreads(), hasQueuedThreads(), isFair(), toString()`**

Classe Semaphore
Exemplo de Utilização

```
class Pool
   {
   private final Semaphore available = new Semaphore(10,true);

   public Object getItem() throws InterruptedException
       {
       available.acquire();
       return getNextAvailableItem();
       }

   public void putItem(Object x)
       {
       if (markAsUnused(x))
          available.release();
       }
   }
```



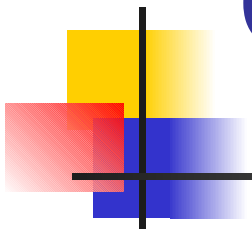
Comparações

Abordagem Clássica X Abordagem Java

- A abordagem clássica não faz referência à lista de espera,
- Os semáforos em Java possuem uma lista de espera, para a qual é possível escolher uma política de espera.

- Nas duas abordagens ainda existe a facilidade na criação de erros de programação.

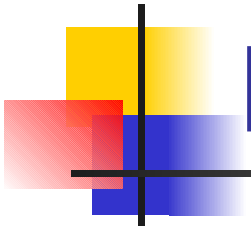
- Na abordagem clássica os valores da variável inteira devem ser sempre maiores ou iguais a zero,
- Em Java o semáforo pode assumir valor negativo na inicialização.



Comparações

Abordagem Clássica X Abordagem Java

- Na abordagem Java foram incorporadas diversas funcionalidades que não são referenciadas na abordagem clássica.



Bibliografia Consultada

- SILBERSCHATZ, A.; GALIN P.; GAGNE G. **Sistemas Operacionais: Conceitos e Aplicações**. Rio de Janeiro: Campus, 2000.

- TANENBAUM, A. S. **Sistemas Operacionais Modernos**. São Paulo: Prentice Hall, 2003.

- **Java 2 Plataform Standard Ed. 5.0 Documentation**. Disponível via WWW em: `http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Semaphore.html`