Outil Théorique de synchronisation des processus : Sémaphores

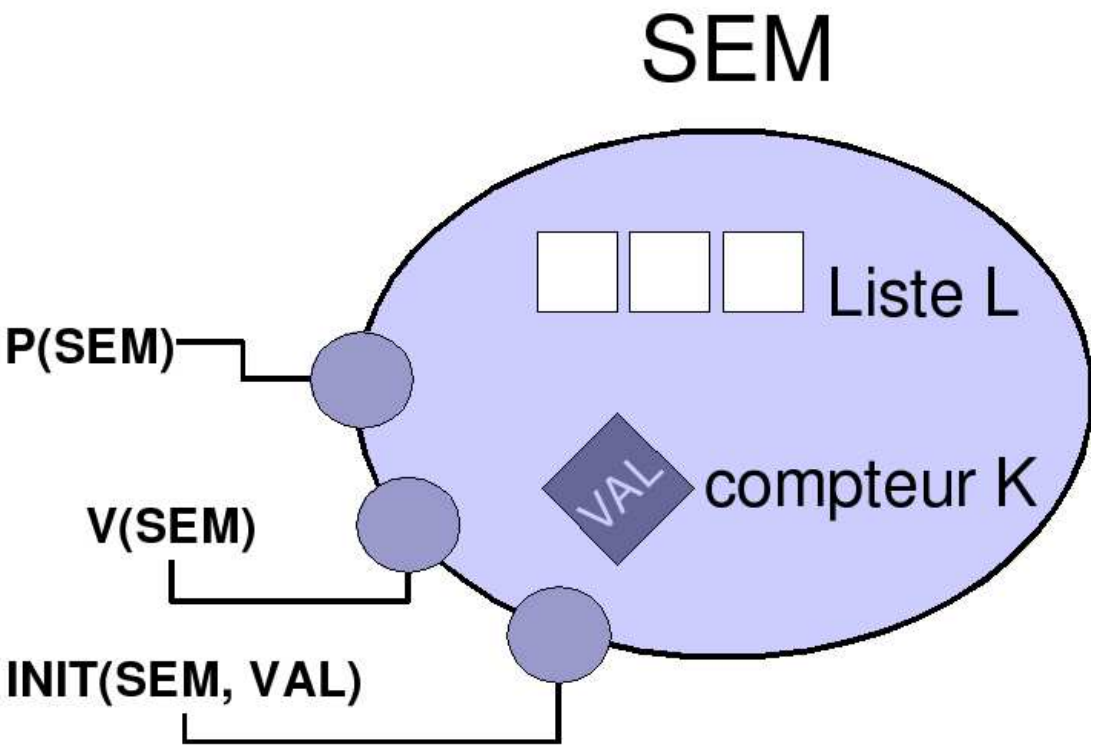
Sémaphore

Outil théorique d'aide à la synchronisation des processus conçus par Dijkstra 65

Un sémaphore est un objet de type Sémaphore.
Sur un sémaphore ***S*** 4 opérations existent

- L'initialisation du sémaphore : ***Init(S,v)***
- *Proberen* (en français "**P**uis-je? ») : ***P(S)***
- *Verhogen*, (en français "**V**as-y!"): ***V(S)***
- destruction

Eléments constituant un sémaphore



En accès privé, un sémaphore a

- une liste de processus en attente
- un compteur ayant une valeur entière

Opération Init

```
function Init(semaphore Sem, int val) {
  masquer_interruption;
  Sem.K = val;
  Sem.L = NULL;
  autoriser_interruption; }
```

// Init : opération atomique ou indivisible
// (qui ne peut être interrompu)
// Init: initialisation du sémaphore.
// Seule la valeur initial du compteur peut varier

Opération P(Sem)

```
function P(semaphore Sem) {
  masquer_interruption;
  if (Sem.K < 1) {
   L.ajouter(processus_appelant);
   processus_appelant.state = bloque;
  }
  Sem.K = Sem.K-1;
  autoriser_interruption; }
```

// P : opération atomique ou indivisible
// (qui ne peut être interrompue)
// Opération P peut être bloquante

Opération V(Sem)

```
function V(semaphore Sem) {
   masquer_interruption;
  Sem.K = Sem.K+1;
  if (sem.K > 0) {
     pr = L.enlever()
     pr.state = prêt;
     }
   autoriser_interruption; }
```

// V : opération atomique ou indivisible
// (qui ne peut être interrompu)

Exemple

Proc1	Proc2	Proc3	Proc4	État de S	temps
Init(S,2)				K= L =	
P(S)					
	P(S)				
		P(S)			
	V(S)				
			P(S)		
V(S)					
P(S)					
	P(S)				
		V(S)			

Opération P(Sem, dmd)

```
function P(semaphore Sem, entier dmd) {
  masquer_interruption;
  if (Sem.K < dmd) {
   L.ajouter(processus_appelant);
   processus_appelant.state = bloque;     }
  Sem.K = Sem.K-dmd;
  autoriser_interruption; }
```

// P : opération atomique ou indivisible
// (qui ne peut être interrompue)
// Opération P peut être bloquante

Opération V(Sem,val)

```
function V(semaphore Sem, entier val) {
   masquer_interruption;
  Sem.K = Sem.K+val;
  if (Sem.K > 0) {
     pr = L.enlever(dmd)
  /* pr est un processus dans L dont la demande (valeur
  de dmd) est inférieure ou égale à sem.K */
    pr.state = prêt;
      }
    autoriser_interruption; }
```

// V : opération atomique ou indivisible
// (qui ne peut être interrompue)

Exemple

Proc1	Proc2	Proc3	Proc4	État de Sem
Int(S,2)				
P(S,1)				
	P(S,2)			
		P(S,3)		
V(S,2)				
			P(S,2)	
P(S,1)				
	V(S,3)			
		V(S,4)		
	P(S,6)			

temps

Outil de synchronisation des processus légers en Java 1.5 : Classe Semaphore

java.util.concurrent.*

Implémentation d'outils de haut niveau pour la synchronisation de processus

- **Sémaphore: Semaphore**
- Loquet: CountdownLatch
- Point de rendez-vous pour échanger des données : Exchanger
- barrière réutilisable: CyclicBarrier

Implémentation des verrous : Lock

Classe **Semaphore**

`Semaphore (int valeur)`

`Semaphore (int valeur, boolean fairness)`

Si **« `fairness == true` »** alors la file d'attente est gère d'après la politique FIFO - premier arrivé premier servi -

`Semaphore` – action « Vas-y! »

```
void release()
void release(int valeur)
```

Semaphore – action « Puis-je? »

Implémenter via des méthodes bloquantes… ces méthodes génèrent « **InterruptedException** »

- **void acquire()**
- **void acquire(int valeur)**
- **boolean tryAcquire(long timeout, TimeUnit unit)**
- **boolean tryAcquire(int valeur, long timeout, TimeUnit unit)**

- **void acquireUninterruptibly()**
- **void acquireUninterruptibly(int valeur)**

Semaphore – action « Puis-je? »

Implémenter via des méthodes non bloquante

- **`boolean tryAcquire()`**
- **`boolean tryAcquire(int valeur)`**

Outils de mise au point sur un « **Semaphore** »

Protected void reducePermits (int r)

// diminue la valeur du compteur du sémaphore de **r**

public int drainPermits()

// fixe la valeur du compteur du sémaphore à **0**

int availablePermits()

// retourne le valeur du compteur du sémaphore

Public boolean isFair()

// retourne **true** si la file d'attente est du type FIFO

Outils de mise au point sur un « **Semaphore** »

```
public final int getQueueLenght()
```

// retourne la longueur de la file d'attente du sémaphore

```
Boolean  hasQueuedThreads()
```

// retourne **true** si un « thread » est bloqué

//dans la liste d'attente du sémaphore

```
Protected collection<Thread> getQueuedThreads()
```

// retourne les threads dans la liste d'attente du sémaphore

// dans un ordre quelconque