Sémaphores

— notations, sémantique et implémentation —

Introduction

Motivations pour des primitives de synchronisation de plus haut niveau :

- L'attente active est inefficace.
- Protocoles d'entrée et sortie de section critique assez complexes.
- Mélange des variables utilisées par les protocoles d'entrée-sortie de section critique avec celles du programme.

Historique

- Dijkstra & co. 1968 avec le système d'exploitation THE
- la première primitive de synchronisation évoluée
- concept inspiré de la synchronisation ferroviaire

Notations et sémantique

Type abstrait

```
public interface Semaphore {
    public void P(); /* passeren */
    public void V(); /* vrygeven */

    public void init(int);
}
```

Invariant du Sémaphore : si nP est le nombre d'opérations P effectuées complètement et nV est le nombre d'opérations V effectuées complètement, alors dans tout état observable du programme on a :

$$nP \leq nV + init$$

Implémentation en utilisant la synchronisation fine : on utilise une variable entière $s = init + nV - nP$ et l'invariant devient :

$$SEM \;:\; s \geq 0$$

Les opérations sont implémentées par :

```
#define P(s) atomic { (s>0) -> s=s-1 }
#define V(s) atomic { s=s+1 }
```

et donc leur règles d'inférence sont :

$$\frac{(P \wedge s>0) \Rightarrow Q^s_{s-1}}{\{P\} \;\; P(s) \;\; \{Q\}}$$

$$\frac{P \Rightarrow Q^s_{s+1}}{\{P\} \;\; V(s) \;\; \{Q\}}$$

Exercice : Quels sont les changements si on veut implémenter un sémaphore binaire, c'est-à-dire $0 \leq s \leq 1$?

Exemple d'utilisation – section critique

Rappel solution n-aire non-primitive

```
int in[1:n] = ([n] 0);
{ CS : in[1]+ ...+in[n] <= 1 }
proctype  P() {
  do :: true -> { CS && in[_pid]=0 }
      atomic { (in[1]+...+in[n]=0) -> in[_pid]=1 }
      { CS && in[_pid]=1 }
      section_critique
      in[_pid] = 0
      { CS && in[_pid]=0 }
      section_non_critique
  od
}
```

Solution avec sémaphore binaire

```
bsem mutex = 1; /* mutex = 1-(in[1]+...+in[n]) */
proctype  P() {
  do :: true ->
      P(mutex) /* in[_pid] = 1 */
      section_critique
      V(mutex) /* in[_pid] = 0 */
      section_non_critique
  od
}
```


Exemple d'utilisation — Producteurs et consommateurs

Solution pour un tampon à une place

```
T buf;
bsem empty = 1; /* semaphone binaire, buf vide */
bsem full = 0;  /* semaphore binaire, buf plein */
{ PC : 0 <= empty + full <= 1 }
proctype  Prod(int i) {
  do :: true ->
         produit message m
  depot: P(empty);
         buf = m;
         V(full)
  od
}
proctype  Cons(int j) {
  do :: true ->
  prise: P(full);
         m = buf;
         V(empty)
         consomme message m
  od
}
```

Implémentation dans un noyau mono-processeur

Primitives à implémenter :

- création avec initialisation de sémaphore
- P
- V

Introduit un nouveau état pour les processus : bloqué pour une opération P. Il faut donc gérer une file de processus bloqués dans chaque descripteur de sémaphore.

On suppose qu'il existe :

- un type descripteur de sémaphore
- un tableau avec les descripteurs de sémaphores utilisés
- une primitive de création de sémaphore par sémaphore déclaré


```
create_sem(int init, int& nom) {
  obtient un descripteur de sémaphore libre
  initialise avec init sa valeur
  initialise nom avec la position du descripteur
  dispatcher()
}

P(int nom) {
  trouve le descripteur de nom
  if (sa valeur > 0) valeur--
  else {
    insère exec_proc dans la file de nom
    exec_proc = 0
  }
  dispatcher()
}

V(int nom) {
  trouve le descripteur de nom
  if (sa file est vide) valeur++
  else {
    enlève le processus en tête de liste
    insère ce processus à la fin de ready_proc
  }
  dispatcher()
}
```