

Théorie et pratique de la concurrence – Master 1 II

TD 7 : Sémaphores

www.liafa.jussieu.fr/~sighirea/cours/concur/

Exercice 1 :

Basic

Considérez le programme suivant qui utilise deux sémaphores binaires.

```
int x=0;
bsem s1=1, s2=0;
active proctype P1(){
    wait(s2);
    wait(s1);
    x=x*2;
    signal(s2)
}

active proctype P2(){
    wait(s1);
    x=x*x;
    signal(s1)
}

active proctype P3(){
    wait(s1);
    x=x+3;
    signal(s2);
    signal(s1)
}
```

1. Quelles sont les valeurs finales possibles de x ?
2. Donnez une implementation des sémaphores binaires en Promela.

Exercice 2 :

Dîner des philosophes en Promela

Voilà une implementation du dîner des philosophes :

```
#define NOMBRE_PHILO 5
/* Les ressources = fourchettes */
#define FOURCHETTE_LIBRE 1
bsem fourchettes[NOMBRE_PHILO] = FOURCHETTE_LIBRE;
#define FOURCHETTE_GAUCHE(p) ((p == 0) -> 4 : p-1)
#define FOURCHETTE_DROITE(p) ((p == 4) -> 0 : p+1)
/* L'état des philosophes */
#define PENSE -1
#define FAIM 0
#define MANGE 1
int etat[NOMBRE_PHILO] = PENSE;
inline PENSER(p) {etat[p] = -1}
inline AFAIM(p) {etat[p] = 0}
inline MANGER(p) {etat[p] = 1}
inline PRENDRE(f) { /** a completer **/ }
inline LIBERER(f) { /** a completer **/ }
/* Les philosophes */
active [NOMBRE_PHILO] proctype Philosophe()
{ int fourchette_gauche = FOURCHETTE_GAUCHE(_pid);
  int fourchette_droite = FOURCHETTE_DROITE(_pid);
  do :: 1 ->
    PENSER(_pid);
```

```

    AFAIM(_pid);
    PRENDRE(fourchette_droite);
    PRENDRE(fourchette_gauche);
    MANGER(_pid);
    LIBERER(fourchette_droite);
    LIBERER(fourchette_gauche)
od
}

```

1. Implementez les définitions manquantes.
2. Montrez que cette solution ne satisfait pas l'absence d'interblocage.
3. En cours, vous avez vu une version asymétrique qui permet d'obtenir l'absence de famine. Implementez cette modification et vérifiez les propriétés.
4. Une autre solution utilise un sémaphore de plus. Les philosophes restent debout autour de la table, mais au plus 4 philosophes peuvent être assis et prendre leurs fourchettes à un moment donné. Implementez cette solution. Vérifiez et/ou prouvez que cette solution évite la famine.

Exercice 3 :

Producteur–consommateur : tampon fini

Un système de type *producteur–consommateur* est constitué de producteurs qui calculent et déposent au fur et à mesure la suite de leurs résultats dans un tampon et de consommateurs qui prennent la suite des résultats déposés pour les utiliser. C'est le cas d'un *pipe* Unix.

Ce problème a plusieurs variantes :

- le tampon est de taille unitaire (*1-place box*),
- le tampon a une taille finie,
- plusieurs consommateurs peuvent prendre un résultat dans le tampon,
- plusieurs producteurs peuvent déposer leurs résultats dans le tampon, etc.

Dans cet exercice, nous considérons les deux premières variantes ci-dessus.

1. Donnez une implémentation du problème pour le cas du tampon à une place. Combien de sémaphores vous avez utilisé ?
2. Si la taille du tampon est plus que 1 mais finie, il existe une solution pour ce problème qui utilise (que) deux sémaphores. Donnez cette solution pour le cas d'un tampon fini circulaire.

Exercice 4 :

Producteur–consommateur : diffusion atomique

Soit un système avec un processus producteur et n processus consommateurs qui communiquent via un tampon ayant b cases. Le producteur dépose des messages dans ce tampon et les consommateurs les prennent. Chaque message déposé par le producteur doit être pris par tous les n consommateurs. En plus, chaque consommateur doit prendre les messages dans l'ordre dans lequel ils ont été déposés. Toutefois, des consommateurs différents peuvent prendre les messages à des moments différents. Par exemple, un consommateur peut prendre jusqu'à b messages avant un autre, si ce dernier est lent. Donner une solution à ce problème en utilisant les sémaphores.