Théorie et pratique de la concurrence – Master 1 Informatique
TP 3 : Programmation concurrente en C (suite)

Dans ce TP on utilisera deux autres moyens de synchronisation de threads en C : les *semaphores* et le *variables de condition*.

On rappelle (sans explication) quelle sont les fonctions concernées. Pour les variables de condition :

```
pthread_cond_init (&varcond,NULL);
pthread_cond_wait (&varcond,&verrou);
pthread_cond_signal(&varcond);
pthread_cond_broadcast(&verrou);
```

Pour les sémaphores :

```
sem_init(&sema,0, n);
sem_wait(&sema);          /* P(sema) */
sem_post(&sema);          /* V(sema) */
```

Exercices

Exercice 1: *Encore producteur-consommateur*

(Pour ceux qui n'ont pas fait programmation systèmes) On programmera encore un producteur et plusieurs consommateurs, cette fois à l'aide des variables de condition. Comme dans le tp précédent,un producteur ajoute un entier choisi au hasard à une file (FIFO). Chaque consommateur attend que la file ne soit pas vide, et ensuite il enlève l'entier de la tête de la file et il l'affiche, et il recommence du début.

Exercice 2: *Encore le pont a voie unique*

On programmera ici le pont à voie unique à l'aide des sémaphores. Des voitures arrivant du nord ou du sud doivent passer un pont à voie unique. Les voitures ayant la même direction peuvent passer le pont au même moment, mais les voitures ayant des directions opposées ne le peuvent pas. Chaque voiture sera un thread différent, qui, une fois passée sur le pont, recommence toujours dans la même direction.

1. Programmez le code des processus voiture, ainsi que le code nécessaire à leur synchronisation à l'aide des sémaphores.
2. Modifiez votre solution pour permettre au plus 4 voitures d'être sur le pont.
3. On souhaite maintenant garantir qu'il y aura une alternance entre voiture des deux sens (si il y a régulièrement des voitures qui veulent entrer dans les deux directions). Pour cela, développez une solution qui laisse au plus entrer 4 voitures sur le pont, si une voiture qui va dans la direction opposée veut entrer.

Exercice 3: *Programmation des semaphores avec les variables de condition*

Dans cet exercice il vous est demandé d'implémenter les sémaphores à l'aide des variables de condition. Vous allez créer un type sémaphore de la forme suivante :

```
typedef struct semaf
pthread_mutex_t verrou;
pthread_cond_t varcond;
int compteur;
g semaphore;
```

Intuitivement le verrou sert à protéger la valeur du compteur associée au sémaphore et la variable de condition est utilisée pour bloquer les *threads* qui tentent de prendre le sémaphore alors que la valeur du compteur est à 0.

1. Programmez les fonctions

   ```
   int semaphore_init(semaphore *sem,int val)
   int semaphore_post(seamphore *sem)
   int semaphore_wait(semaphore *sem)
   ```

   de ce sémaphore (qui correspondent aux fonction équivalentes pour les sémaphore de la libraire `Pthreads`).
2. Testez votre version des sémaphores en les utilisant pour implémenter le pont à voie unique.

Exercice 4: *Le drapeau hollandais*

Une collection de boules de couleur est réparti entre N processus. Il y a M N couleurs différentes de boules. L'objectif est pour les processus d'échanger des boules de sorte que quand l'algorithme termine, pour tout i, le processus i a toutes les boules de couleur i. Le nombre totale de boules est inconnu aux processus

Exercice 5: *Di usion 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.

Programmez en C une solution à ce problème en utilisant les sémaphores.