Théorie et pratique de la concurrence – Master 1 Informatique
TP 2 : Les verrous en C

Les mutexs

Les *mutexs* sont des verrous proposés par la bibliothèque `Pthreads`. Pour les utiliser, il faut inclure la bibliothèque :

```
#include <pthread.h>
```

La déclaration d'un *mutex* se fait de la façon suivante :

```
pthread_mutex_t verrou;
```

Pour initialiser un *mutex*, on procède comme suit :

```
pthread_mutex_init (&verrou , NULL);
```

Le premier argument correspond au verrou que l'on souhaite initialiser, quand au second, qui est mis à `NULL`, il précise les attributs du verrou.
Pour "prendre un verrou" (i.e. faire P sur le verrou), on fait :

```
pthread_mutex_lock (&verrou);
```

Et pour le libérer :

```
pthread_mutex_unlock (&verrou);
```

Par conséquent, une section critique pourra être faite de la façon suivante :

```
pthread_mutex_lock (&verrou);
/*section critique*/
pthread_mutex_unlock(&verrou);
```

Exercices

Exercice 1: *La mémoire partagée*

Reprenez l'exercice 4 du Tp 1 et proposer une implémentation dans laquelle chaque donnée produite par un producteur n'est lue que par au plus un consommateur.

Exercice 2: *Mauvaise manipulation de mutex*

1. Proposez un programme avec deux threads et deux verrous et de telle sorte que les deux threads soient bloqués dans l'attente de verrous pris par l'autre thread.
2. Inventez un programme où un thread tente de libérer un verrou qui a été pris par un autre thread. Qu'observez-vous ?

Exercice 3: *Encore des producteurs-consommateurs*

Proposez une implémentation d'un système de producteurs-consommateurs dans lequel la communication se fait par file FIFO. Les règles sont les suivantes. Les producteurs ajoutent des entiers dans la file qui sont ensuite lus/retirés par les consommateurs. Pour qu'un consommateur retire un entier de la file, il doit être sûr que cet entier a été lu par tous les consommateurs.

La file FIFO devrait être représentée par une liste simplement chaînée avec un pointeur `top` sur le début de la liste (utilisé par les consommateurs pour commencer à lire des entiers de la file) et un pointeur `tail` sur la fin de la liste (utilisé par les producteurs pour ajouter des entiers à la file). La liste contiendra toujours un noeud sentinelle (qui n'est pas utilisé pour stocker des entiers supposes à être lus par les consommateurs) pointé au début par les deux pointeurs `top` et `tail`. De cette manière, la file est vide quand `top = tail`. Les consommateurs lisent des entiers en parcourant la liste à partir de `top` (notez que les consommateurs n'avancent pas nécessairement au même rythme). Ainsi, les entiers sont lus dans l'ordre qui sont ajoutés par les producteurs. Pour tester qu'un entier de la file a été lu par tous les consommateurs il faut ajouter un ou plusieurs champs aux noeuds de la liste. **Faites attention aux modifications concurrentes de la file.**

Exercice 4: *Le dîner des philosophes*

Le problème du dîner des philosophes est un problème classique de partage des ressources en programmation concurrente. Ce problème se résume ainsi. Il y a `n` philosophes assis autour d'une table. Chaque philosophe a devant lui une assiette de riz et pour manger il a besoin de deux baguettes. Il y a en tout `n` baguettes qui se trouve à la gauche de chaque assiette. Un philosophe se comporte de la façon suivante : il pense, puis si il a envie de manger, il prend sa baguette gauche ensuite il prend sa baguette droite et quand il termine de manger, il rend les deux baguettes et il recommence à penser et ainsi de suite.
Dans cet exercice on souhaite modéliser ce problème du dîner des philosophes par un programme en C dans lequel chaque baguette sera représentée par un verrou et chaque philosophe sera implémenté par un thread.

1. Programmez votre solution et testez-la pour 5 philosophes.
2. Normalement votre solution devrait aboutir à un interblocage des philosophes. Essayez d'exhiber cet interblocage.
3. Proposez des solutions (et programmez-les) pour résoudre cet interblocage ? (*Indication : Il faut éliminer les entrelacements qui conduisent à l'interblocage, soit en ajoutant de la coordination entre les philosophes soit en modifiant la stratégie utilisée par les philosophes pour prendre les deux baguettes.*) Vos solutions permettent-elles de garantir l'absence de famine pour un philosophe ?