Concurrence – Master 1
TP 3 (séance 7)
Programmation concurrente en C — bibliothèque `Phtreads`

Un *thread* est un processus léger, c'est-à-dire un processus qui a ses propres compteur programme et pile d'exécution, mais aucun des données "lourdes" associées en général avec un processus (par exemple les tables des pages mémoire).

L'interface de la bibliothèque `Pthreads` a été proposée dans les années 1990 comme un standard POSIX (d'où le préfixe P) pour les routines de programmation multi-*thread* en C. Différentes implémentations de ce standard sont actuellement disponibles sur les systèmes de type UNIX. Cette bibliothèque contient une douzaine de fonctions pour le contrôle des *threads* et pour la synchronisation.

Ce TP a comme but de vous introduire les fonctions de cette bibliothèque utiles pour simuler les instructions `parbegin`...**parend** ainsi que la synchronisation par sémaphores.

Création et terminaison de *threads*

Pour utiliser la bibliothèque `Pthreads` dans un programme C il faut, premièrement, inclure son fichier de déclarations standard :

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

Deuxièmement, il faut déclarer des variables pour un descripteur d'attributs de *thread* et pour un ou plusieurs descripteurs de *threads* comme suit :

```
pthread_attr_t tattr; /* attribut de thread */
pthread_t pid, cid;   /* descripteurs de thread */
```

Troisièmement, il faut initialiser les attributs en exécutant :

```
pthread_attr_init(&tattr);
pthread_attr_setscope(&tattr, PTHREAD_SCOPE_SYSTEM);
```

La dernière instruction indique quel est l'ensemble de *threads* avec lequel il entre en compétition pour l'ordonnancement. La seule valeur possible pour les systèmes Linux est le système entier. Une autre valeur possible prévue par le standard est `PTHREAD_SCOPE_PROCESS`, pour ordonnancement local pour les *threads* d'un processus. On peut également initialiser d'autres attributs (par exemple la priorité), voir le manuel en ligne (**man**).

Finalement, on crée les *threads* comme suit :

```
pthread_create(&pid, &tattr, start_func, arg);
```

Le premier argument est l'adresse d'un descripteur de *thread*, suivi de l'adresse du descripteur d'attributs qui a été initialisé. Le code du *thread* est la fonction `start_func` qui doit avoir un seul argument `arg`. Si la création est effectuée correctement, la fonction renvoie 0, sinon un code d'erreur.

Un *thread* se termine en appelant :

```
pthread_exit(value);
```

où l'argument peut être NULL.

Un processus parent peut attendre la terminaison d'un fils en effectuant :

```
pthread_join(pid, value_ptr);
```

où le deuxième argument est une adresse pour le stockage de la valeur de retour du *thread* (avec `pthread_exit`).

Sémaphores

Les *threads* peuvent communiquer entre eux par les variables globales. Ils peuvent se synchroniser par attente active, sémaphores, verrous et variables conditionnelles. On considère ici que les sémaphores.

Pour utiliser les sémaphores il faut, premièrement, inclure le fichier de déclarations standard :

```
#include <semaphore.h>
```

Deuxièmement, déclarer un descripteur de sémaphore global aux fonctions exécutées par les *threads* qui l'utiliseront :

```
sem_t mutex;
```

Troisièmement, il faut initialiser les descripteurs comme suit :

```
sem_init(&mutex, SHARED, 1);
```

Si le paramètre `SHARED` est 1, le sémaphore peut être partagé entre processus ; sinon il peut être partagé que entre les *threads* du même processus.

Finalement, les opérations P et V sont implémentées par les fonctions `sem_wait` respectivement `sem_post`. Donc, une section critique peut être implémentée comme suit :

```
sem_wait(&mutex);        /* P(mutex) */
/* section critique */
sem_post(&mutex);        /* V(mutex) */
```

D'autres fonctions sont disponibles pour les sémaphores, consultez le **man**.

Exercices

Exercice 1 : Programmer en C le problème du producteur-consommateur avec un tampon unitaire.

- Le producteur produit une suite d'entiers terminée par 0.
- Le consommateur calcule et renvoie au parent la somme des ces entiers.
- Le processus parent attend la terminaison de ses fils et affiche la somme calculée.

En utilisant les attributs, changer de priorité pour chaque *thread* fils et observer le comportement de votre programme.

Exercice 2 : Programmer en C le problème 1 du TD précédent, c'est-à-dire la diffusion atomique.

Exercice 3 : Programmer en C le problème 4 du TD précédent, c'est-à-dire la salle de bain unisexe.