TP 3

Thread & Synchronisation

Le TP a pour but d'apprendre à manipuler les pthread d'UNIX ainsi que les synchronisations (mutex/sémaphore)

Question 1.1
Ecrire un programme qui crée trois threads qui exécutent toutes le même code « fct_thread » consistant à modifier de manière répétée une variable globale ainsi qu'une variable locale dans la fonction associée à chaque thread. Quelle différence y-a-t-il avec l'utilisation des processus vu dans le TP précédent ?

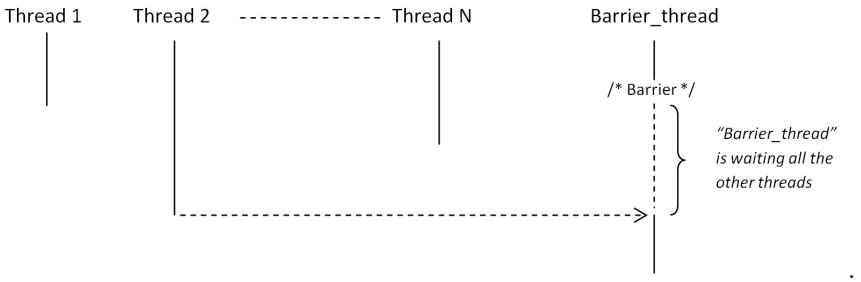
Question 1.2
Ecrivez un programme composé de deux threads (lancés par votre fonction main). Le premier thread devra faire deux affichages : "je "et "mes". Le deuxième fera également deux affichages : " synchronise " et "threads". Utilisez des sémaphores pour que l'exécution du programme produise l'affichage de "je synchronise mes threads" à l'écran quelque soit l'ordre de lancement des threads. Vous pourrez rajouter des attentes avant ou après les affichages pour bien vérifier que cela n'a aucune influence sur l'ordre d'exécution une fois que les synchronisations sont présentes.

Faire 2 versions du programme : l'une en utilisant des sémaphores et l'autre en utilisant des mutex.

Note :

- Utilisez la bibliothèque <pthread.h> pour créer les threads et la bibliothèque <semaphore.h> pour la gestion des sémaphores.
- Pour plus de lisibilité vous redéfinirez les primitives sem_wait() et sem_post() par des macro-fonctions P( ) et V( );
- Utilisez le flag -lpthread pour pouvoir compiler le programme.

Question 1.3
Ecrire un programme permettant de créer une barrière entre un groupe de N threads et un thread particulier qu'on nommera « barrier_thread »par la suite. Une fois que « barrier_thread » aura atteint cette barrière il devra attendre que chaque thread ait atteint un point bien particulier dans leur code (qui ne correspond pas forcément à la fin des threads). On pourra simuler dans ces N threads l'exécution d'instructions avant d'atteindre ce point par des attentes aléatoires (même si cela correspondra à une attente passive et non active).



Exemple d'exécution avec 3 threads + barrier_thread :

```
[thread 1] Point atteint
[b_thread] Barrière atteinte, en attente…
[thread 3] Point atteint
[thread 2] Point atteint
[b_thread] Je peux continuer !
```

Question 1.4

Ecrire un programme permettant de lire et écrire dans un tampon partagé. Le problème est composé de deux thread et d'un tampon partagé de taille N. Le thread "ecrivain" appelle une fonction ecrire_tampon(data) chargée d'écrire dans le tampon partagé. Le thread "lecteur" appelle une fonction lecture_tampon() chargée de lire une donnée précédemment écrite par la thread "ecrivain". Le tampon sera un simple tableau de N cases.

Note :

- Le tampon est circulaire pour traiter les données dans l'ordre de production;
- Une lecture et une écriture doivent pouvoir se faire en parallèle si c'est pour des cases différentes;
- Pour plus de lisibilité vous redéfinirez les primitives sem_wait() et sem_post() par des macro-fonctions P() et V();
- Utilisez le flag -lpthread pour pouvoir compiler le programme.