TD : Parallélisme à base de thread

1 Hello World !

1. Ecrire un programme créant un nombre de threads spécifié en argument, chaque thread devant afficher une chaîne de caractères sur la sortie standard.
2. Modifier le programme précédent pour que chaque thread affiche egalement un entier qui lui est passé en paramètre égal à l'ordre dans lequel il a été créé (le premier thread affiche "1", le second "2", etc). La chaîne ainsi que l'identifiant du thread devront être passés en argument. Essayez de deux manières différentes :
   (a) En créant la chaîne ("`Hello World !`" + identifiant) **avant de la passer** au thread à créer (indice `man sprintf`).
   (b) En passant la chaîne et l'identifiant **séparément** au thread à créer.
3. Modifier le programme précédent pour que chaque thread se termine en retournant la valeur donnée en argument multipliée par deux (deux solutions). Le thread principal affiche les valeurs de retour.

2 Multithread unsafe

Tout programme parallèle doit manier avec précaution des ressources partagées. Dans le cas de la programmation "multithreadée", les variables globales sont des variables communes et visibles de tous les threads. Accéder en écriture de façon concurrente à ce type de variables sans protection peut rendre le programme faux.

Ouvrir le fichier `mt_unsafe/exemple.c` et étudier le code.

- Compilez le code une fois. Exécutez le plusieurs fois.
  Que remarquez-vous ? Comment expliquer le problème ?
- Rajouter le mot-clé `volatile` devant la déclaration de `count`.
  Que signifie ce mot-clé ?
  Est-il suffisant pour rendre le programme correct ?
- Comment résoudre le problème ? (indice `man pthread_mutex_lock`).

3 Recherche parallèlle du max d'un tableau

1. Ecrivez un programme qui effectue les opérations suivantes:
   (a) Remplit un tableau de 20 entiers avec des valeurs tirées aléatoirement entre 1 et 1000.
   (b) Recherche la valeur maximale dans le tableau à l'aide de **n** threads.
      - Chaque thread recherche la valeur maximale dans une sous-partie du tableau, et renvoie cette valeur.
      - Le thread principal recherche le maximum parmi les valeurs retournées.
2. Modifiez le programme précédent pour que la valeur maximale soit stockée et mise à jour dans une variable `max` partagée par tous les threads.

4 Barrière multithread

On désire implémenter un mécanisme de `barrière` pour la synchronisation de threads. Une barrière est un **point de rendez-vous** entre threads.

1. Ecrivez une fonction `barrier` qui implémente une barrière à l'aide de threads POSIX. Considérez que le nombre total de threads est stocké dans une variable globale `nthreads` (indice : utilisez les **variables de condition**).
2. Ecrivez un programme créant `n` threads. Chaque thread généré par le thread principal attend pendant une durée aléatoire (comprise entre 1 et 5 secondes), puis rejoint la barrière. Le thread principal appel la barrière après avoir créé tous les threads fils. (indice : la fonction `rand` n'est pas réentrante).

5 Les semaphores

Reprenez la recherche de maximum d'un tableau en remplaçant l'utilisation de `mutex` par l'utilisation d'un `semaphore`.

6 Bonus: Le problème du producteur/consommateur

Le problème du producteur/consommateur est un problème classique de synchronisation de tâches. Un ensemble d'acteurs accède de manière concurrente à une base d'éléments. Les producteurs accèdent à la base pour y déposer des éléments. Les consommateurs lisent la base pour en retirer un élément. Si un producteur ne peut pas accéder à la base (base pleine), il se met en attente jusqu'à ce qu'un élément soit lu par un lecteur. De même, si la base est vide, les lecteurs se mettent en attente jusqu'à ce qu'un producteur y dépose un élément.

1. Implémentez ce problème pour le cas simple d'un seul producteur et un seul consommateur se partageant une base à un seul emplacement :
   (a) Ecrivez la fonction `void produire (int donnee)` qui écrit un élément dans une base d'entiers.
   (b) Ecrivez la fonction `int consommer (void)` qui lit un élément dans cette base.
2. Généralisez le problème pour le cas d'une base contenant `n` éléments et un nombre quelconque de producteurs et consommateurs.