Programmation Concurrente

Threads POSIX

Série 1

Exercice 1

Soit le programme multi-threadé ci-dessous créant 16 threads où chaque thread affiche son numéro :

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

#define NUM_THREADS 16

void *thread(void *thread_id) {
    int id = *((int *) thread_id);
    printf("Hello from thread %d\n", id);
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    for (int i = 0; i < NUM_THREADS; i++) {
        int code = pthread_create(&threads[i], NULL, thread, &i);
        if (code != 0) {
            fprintf(stderr, "pthread_create failed!\n");
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}
```

Questions

- A votre avis, l'implémentation de ce programme est-elle correcte ? Vérifiez votre réponse en exécutant le programme **plusieurs fois**.
    - Si ce n'est pas le cas :
        - Expliquez pourquoi le comportement est incorrect.
        - Modifiez le code afin qu'il produise le comportement attendu.

Exercice 2

On aimerait implémenter un programme créant un thread pour calculer la somme d'un tableau d'entiers. Ce thread doit calculer la somme des entiers du tableau et le programme principal (`main`) s'occupera d'afficher le résultat.

Les points suivants sont à respecter :

- Aucun argument n'est passé au thread.
- Le thread ne retourne aucune valeur.
- Les entiers sont entrés sur la ligne de commande (chacun séparé par un espace).
- La somme est stockée dans une variable globale.
- N'oubliez pas de vérifier qu'aucune des fonctions utilisées n'échoue ; en cas d'erreur, veuillez afficher le résultat sur le canal d'erreur (`stderr`).
- Ecrivez un makefile pour compiler votre code et faire le ménage (cible `clean`).

Exercice 3

Nous aimerions implémenter une variante du programme de l'excercice 2. Celui-ci sera fonctionnellement identique, mais ne devra utiliser **aucune variable globale**.

Vous pouvez par contre passer un argument à votre thread et celui peut retourner une valeur.

Exercice 4

Nous souhaitons écrire un programme calculant la somme des entiers de 1 à N à l'aide de M threads. Chaque thread calculera la somme d'un sous-ensemble de ces entiers et la somme globale sera obtenue en calculant la somme des résultats intermédiaires de chaque thread.

Les entiers sont répartis uniformément entre les threads comme suit (exemple avec 3 threads) :

- Thread 1 : 1, 4, 7, . . .
- Thread 2 : 2, 5, 8, . . .
- Thread 3 : 3, 6, 9, . . .

Le programme doit lancer M threads, attendre qu'ils se terminent, faire la somme des résultats intermédiaires et afficher le résultat. Les valeurs N et M seront passées en ligne de commande.

Il est important que le programme respecte les points suivants :

- L'implémentation ne doit utiliser **aucune variable globale**.
- Le travail à effectuer pour chaque thread créé doit être aussi équitable que possible, quel que soit les valeurs N et M choisies par l'utilisateur (ex : N = 20, M = 8).
- Evitez d'utiliser un tableau pour contenir les valeurs à additionner.
- Réaliser un test de validation automatiquement du résultat obtenu avec une version séquentielle de l'algorithme.