TP 5 : Programmation concurrente

Exercice 1 : Affichage

1) En s'aidant de l'antisèche sur la programmation concurrente en C, traduire le programme en pseudo-code suivant en C :

```
fonction f(index)
  Afficher "Le fil $index a démarré"
    pour i de 1 à p × n faire
      si i mod n = 0 alors
        Afficher "Le fil $index a atteint $i."

fonction Principale
  Afficher "Avant"
  fils ← [CréerFil(f, 0),. . ., CréerFil(f, N - 1)]
  Afficher "Pendant"
  pour i = 0 à N − 1 faire
    JoindreFil(fils[i])
  Afficher "Après"
```

2) Exécuter votre programme plusieurs fois. Que constate-t-on ?

Exercice 2 : Incrémentation

3) Écrire un programme qui crée deux fils d'exécution qui incrémentent chacun la même variable globale 100000 fois.

L'exécuter en affichant la valeur de la variable à la fin de l'exécution.

Exercice 3 : Somme d'un tableau

4) Écrire une fonction qui calcule (sans programmation concurrente) la somme des nombres entiers d'un tableau donné en paramètre (avec sa taille).

Attention cette question est très difficile.

5) Écrire la même fonction en utilisant la programmation concurrente. On ajoute un paramètre qui indique le nombre de fils à créer par la fonction.

6) Exécuter les deux fonctions sur un tableau de taille 100000 contenant des entiers aléatoires.

Vérifier que la valeur renvoyée par les deux fonctions est bien la même.

7) Mesurer le temps nécessaire aux deux fonctions pour calculer cette somme.

Avec combien de fils d'exécution la deuxième fonction est-elle la plus rapide ?

Exercice 4 : Première utilisation des mutex

8) Modifier le programme de la question 3 pour que la variable globale atteigne la valeur 200000 à coup sûr.

Exercice 5 : File concurrente en OCaml

Le module `Queue` de OCaml fournit une implémentation des files avec les fonctions suivantes :

- `Queue.create : unit -> 'a Queue.t` qui crée une file vide ;
- `Queue.push : 'a -> 'a Queue.t -> unit` qui ajoute un élément à la file ;
- `Queue.take_opt : 'a Queue.t -> 'a option` qui retire un élément de la file et renvoie `None` si la file est vide.

> **9)** Écrire un programme qui crée trois fils d'exécution ajoutant chacun les nombres de 1 à 100 (dans l'ordre) dans une file partagée (sans utiliser de mutex car on cherche à voir ce qu'il va se passer).

> **10)** Exécutez ce programme en affichant la file obtenue à la fin.

On choisit de rendre cette structure de données concurrente avec le type suivant :

```
type 'a concurrent_queue = {
  lock : Mutex.t;
  queue : 'a Queue.t
}
```

> **11)** Définir les fonctions `create`, `push` et `take_opt` pour les `concurrent_queue`.

> **12)** Réécrire le programme de la question 9 pour utiliser cette nouvelle structure de données. L'exécuter.

Exercice 6 : Programme à corriger

On considère le programme suivant que vous pouvez retrouver dans le fichier `faux.c`.

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

#define NB_THREADS 3

void *f(void *arg){
  int index = *(int*)arg;
  for (int i = 0; i < 10; i++) {
    printf("Thread %d : %d\n", index, i);
  }
  return NULL;
}

int main(void){
  pthread_t threads[NB_THREADS];
  printf("Before creating the threads.\n");
  for (int i = 0; i < NB_THREADS; i++) {
    int thread_index = i;
    pthread_create(&threads[i], NULL, f, &thread_index);
  }
  printf("While the other threads are running.\n");
  for (int i = 0; i < NB_THREADS; i++) {
     pthread_join(threads[i], NULL);
  }
  printf("After the other threads have stopped running.\n");
  return 0;
}
```

13) Compiler et exécuter ce programme. Que constate-t-on ?

14) Expliquer d'où provient le problème.

15) Proposer une version correcte de ce programme.

Exercice 7 : Jeu de la vie

Le jeu de la vie se joue sur une grille (généralement infinie mais on se contentera d'une grille $m \times n$) dans laquelle les cases, appelées « cellules », peuvent être soit dans l'état « vivante » soit dans l'état « morte ».

Au temps 0, on choisit quelles cellules sont vivantes et quelles cellules sont mortes. Puis, au temps t :

- si une cellule morte possède exactement trois cellules voisines vivantes (parmi les huit cellules adjacente orthogonalement et diagonalement) alors elle devient vivante, sinon elle reste morte.
- si une cellule vivante possède deux ou trois cellules voisines vivantes elle le reste, sinon elle meurt.

16) Écrire un programme en C (pour l'instant sans programmation concurrente) qui permet d'afficher l'évolution d'une grille donnée de taille $m \times n$ jusqu'au temps t.

17) Quel schéma de synchronisation vu en cours peut ici être utile pour synchroniser plusieurs fils ?

18) Écrire une nouvelle version de votre programme utilisant la concurrence.

19) Comparer les performances entre vos deux versions (et vérifier qu'elles produisent les mêmes résultats sur des exemples).