Université de Reims Champagne-Ardenne
Département de Mathématiques, Mécanique et Informatique

Pierre Delisle

Travaux pratiques 1

Premiers programmes multithreads

Exercice 1 – Bonjour le monde (de moi et mes amis)

Implémentez une version multi-threadée du fameux programme "Hello World" où chaque thread affiche un message, un identifiant entre 1 et le nombre total de threads, ainsi que le nombre total de threads. Votre programme ne doit pas contenir de variables globales : l'identifiant et le nombre total de threads doivent être fournis au thread lors de sa création. De plus, votre programme doit être fonctionnel avec un nombre variable de threads, ce dernier devant être spécifié au moment de l'exécution du programme (Exemple : ./hello_world 4). L'affichage devra être semblable au suivant :

```
Hello world ! du thread 2 de 4 !
Hello world ! du thread 3 de 4 !
Hello world ! du thread 1 de 4 !
Hello world ! du thread 4 de 4 !
```

Exercice 2 – Utilisation de variables partagées

Considérez le programme **compteur_seq.c** suivant (disponible sur la page Moodle du cours) :

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

#define SPIN	4000000

int compteur = 0;

int main(int argc, char *argv[]) {
	int i, diffTemps;
	time_t temps1, temps2;

	temps1 = time(NULL);
	for (i = 0; i < SPIN; i++)
		compteur++;
	temps2 = time(NULL);
	diffTemps = (int) temps2 - (int) temps1;
	printf("Valeur finale du compteur : %d\n", compteur);
	printf("Temps : %d secondes\n", diffTemps);

	return 0;
}
```

Ce programme très utile incrémente un compteur global SPIN fois.

1. Écrivez un programme qui créé 4 threads qui incrémentent la variable compteur SPIN / 4 fois chacun, sans la protéger et en la laissant globale. Testez votre programme sur différentes valeurs de SPIN. Obtenez-vous bien à chaque fois la valeur finale SPIN que vous auriez obtenu dans le programme séquentiel (si vous l'obtenez, c'est que vous n'avez pas spécifié une valeur de SPIN assez grande) ? Observez-vous un impact sur le temps d'exécution du programme ?
2. Ajoutez un mutex à votre programme afin de protéger la variable. Verrouillez le mutex juste avant l'incrémentation du compteur et déverrouillez-le juste après. Obtenez-vous maintenant une valeur finale correcte (normalement, vous devriez !) ? Quel est l'impact sur le temps d'exécution ?

Exercice 3 – Calcul de la suite de Fibonacci

Considérez le programme **fib_seq.c** suivant (disponible sur la page Moodle du cours.

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

unsigned int fib(int n) {
	unsigned int x, y;

	if (n <= 1)
		return n;
	else {
		x = fib(n - 1);
		y = fib(n - 2);
		return x + y;
	}
}
```

```
int main(int argc, char *argv[]) {
	unsigned int f;
	int n, diffTemps;
	time_t temps1, temps2;

	n = atoi(argv[1]);
	temps1 = time(NULL);
	f = fib(n);
	temps2 = time(NULL);
	diffTemps = (int) temps2 - (int) temps1;
	printf("Valeur du %d-ieme nombre de Fibonacci : %u\n", n, f);
	printf("Temps : %d secondes\n", diffTemps);

	return 0;
}
```

Ce programme permet de calculer récursivement (et de façon non efficace, mais nous ne nous en formaliserons pas ici...) le n-ème nombre de la suite de Fibonacci.

1. À partir de quelle valeur de *n* ce programme prend-t-il plusieurs secondes de calcul ?
2. Proposez une version multi-threadée de ce programme qui utilise 3 threads : un thread qui calcule fib_{n-1}, un thread qui calcule fib_{n-2} et un thread principal qui s'occupe de créer les deux autres threads, de récupérer les valeurs qu'ils ont calculées et d'afficher le résultat final. Testez son temps d'exécution pour différentes valeurs de n égales et supérieures à celle identifiée à la question précédente. Quelle accélération maximale arrivez-vous à obtenir ?
3. [Facultatif] Proposez une version multithreadée de ce programme qui utilise davantage de threads pour améliorer le temps d'exécution. Quelle accélération maximale arrivez-vous à obtenir cette fois-ci ?