# INF4033 Lab 6 :
# Pour aller plus loin avec les Pthreads

Enseignant : Alexandre BRIÈRE

Étudiant : Thierry KHAMHPOUSONE – TC44 IA&DATA

## 1 Introduction

### 1.1 Objectifs

Mettre en pratique le paradigme "divide and conquer" grâce aux Pthreads.
Vous n'utiliserez pour ce TP que les fonctions présentes dans la bibliothèque Pthread (fork, sémaphore, etc. ne sont pas autorises).

### 1.2 Pour commencer

Les codes sources sont fournis avec un makefile, tappez la commande suivante pour compiler :

$ make

### 1.3 Astuces

Pour connaitre le prototype exact des fonctions de la bibliothèque Pthread, pensez à la commande man.

$ man pthread_create

Pensez bien à vérifier toutes les valeurs de retour de la bibliothèque Pthread.

## 2 Exercice

Le but de l'exercice est de réaliser un algorithme de tri parallèle. L'application de tri a de nombreuses ressemblances avec l'application recherche max. Chaque thread trie individuellement sa partie de tableau. La fonction sort_partial permet de faire ce travail. Une fois les sous-tableaux triés, les threads se regroupent deux à deux (terminant et fusionnant). Chaque **thread terminant** retourne son tableau **trié au thread fusionnant** associé. Chaque **thread fusionnant attend la fin** du thread terminant associé, puis commence à fusionner son tableau avec celui du thread attendu.

Pour la première itération, les threads fusionnants sont ceux dont l'index est multiple de deux, les autres threads étant terminants. Lors de la deuxième itération, les fusionnants sont multiples de quatre, les autres sont terminants. La troisième itération, les fusionnants sont multiples de huit, etc.

## 2.1 Completez le code

En vous servant du code de recherche du max et en utilisant le code donné dans le dossier exo, faites en sorte que le programme trie le tableau avec plusieurs threads.

Pour compiler le code, utilisez la commande make. Le programme attend en paramètre le nombre de threads à lancer pour effectuer le tri.

## 2.2 Mesure de performance

Le programme écrit sur la sortie d'erreur le nombre de secondes passées à faire le tri.
Tracez le temps d'exécution en fonction du nombre de threads.
Que constatez-vous ? Comment expliquez-vous cela ?

### 1) Faisons d'abord un premier test sur un buffer de 100 valeurs :

```
❯ make
gcc -o main.o -c main.c
gcc -o sort main.o sort.o time.o -lpthread -lc -lm
❯ ./sort 10
display after shuffle
82 1 33 36 4 5 73 46 3 9 66 55 87 68 88 80 16 17 78 61 20 21 22 23 24 0 26 27 28 57 84 31 32 2 34 14
 8 10 41 39 40 53 42 18 25 85 44 70 37 91 50 51 96 52 54 97 7 69 74 59 60 19 62 63 64 65 67 48 35 56
 47 89 30 71 58 75 76 77 11 79 15 81 38 83 72 13 86 45 93 29 90 49 92 12 94 95 6 43 98 99
display after sort
Real time used = 0.000251
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
 ~/Desktop/Ressources LAB 6 - E44-20201103/exo                                    ✔ 16:27:22
```

On remarque que le tableau a été trié correctement.

2) **Augmentons maintenant le nombre de valeurs dans le buffer et testons pour plusieurs nombres de threads en arguments.**

```
❯ make
gcc -o main.o -c main.c
gcc -o sort main.o sort.o time.o -lpthread -lc -lm
❯ ./sort 1
Real time used = 14.119791
❯ ./sort 2
Real time used = 8.047194
❯ ./sort 3
Real time used = 5.823217
❯ ./sort 4
Real time used = 4.681897
❯ ./sort 5
Real time used = 4.612314
❯ ./sort 6
Real time used = 4.208358
```

On remarque que pour 1 thread, le temps d'exécution est de 14.12, et plus on augmente le nombre de threads, plus le tri est effectué rapidement. Ici, pour 6 threads, le temps d'exécution est descendu à 4.20.
Le temps d'exécution diminue lorsque le nombre de threads augmente car lors du lancement de la fonction de tri, on a découpé et réparti le tableau (buffer) sur plus de threads qui vont trier localement leurs tableaux avant de fusionner avec les autres threads.
Donc en augmentant le nombre de threads, le thread aura « moins de travail » à faire puisque la charge de travail sera répartie entre les autres threads.