TD8 : Threads

Exercice 1 :

En utilisant la fonction `pthread_create( )`, écrire le programme où le processus principal crée un thread pour chacun des comportements suivants :

1. Affichage de “Hello World!” ;
2. Affichage d’un entier aléatoire généré par le processus principal ;
3. Affichage d’un entier aléatoire généré par le thread qui sera aussi affiché par le processus principal ;
4. Affichage de la moyenne d’un tableau de N entiers générés aléatoirement par le processus principal.

Exercice 2 :

L’objectif de cet exercice est d’offrir à l’utilisateur une bibliothèque de fonctions et structures permettant le traitement d’un grand tableau dans un environnement *multi-threadé*. L’exercice se découpe en plusieurs parties.

1. Le programme principal : l’exécution du programme se fait par la ligne de commande suivante, où `m` définit le nombre de threads générés par le programme principal, `n` la taille du tableau et `opcode` l’opération à réaliser sur le tableau :

   ```
   $ ./reduction m n opcode
   ```

   Le programme doit dans un premier temps créer le tableau et générer les entiers le composant (entre 1 et 100), puis créer les threads. Une fois l’exécution des threads terminée, il affiche le résultat final obtenu et uniquement ce résultat.

2. La structure message : l’argument de la fonction d’un thread est une structure comportant quatre (4) champs : le tableau d’entiers (dans sa totalité), les indices de début et de fin de traitement (partie du tableau que le thread doit traiter) et le résultat (renseigné par le thread à la fin de son exécution).

3. L’opération : l’analyse de l’`opcode` par le programme principal lui permet d’identifier et de sélectionner l’opération à effectuer. Le tableau ci-dessous contient les fonctions disponibles et leur `opcode` respectif.

Nom	Code
somme	+
moyenne	/
min	m
max	M

4. Exécution des threads : chaque thread doit appliquer la fonction indiquée par l'**opcode** sur la partie du tableau qu'il doit traiter, puis retourner le résultat de cette opération.

Exercice 3 :

La fonction **getopt( )** permet l'analyse et le décodage des arguments d'un programme en utilisant des options (voir sa définition ci-dessous). Il est alors possible de gérer des arguments obligatoires et/ou optionnels, quelque soit l'ordre dans lequel ils sont donnés. L'exécution du programme se fera par la ligne de commande suivante :

```
$ ./reduction -t m -s n -o opcode
```

On supposera que l'argument '**-t**' pour le nombre de threads est optionnel et les deux autres obligatoires. Si le nombre de threads n'est pas donné, alors par défaut il vaudra 4.

---

getopt( )

```
#include <unistd.h>

int getopt ( int argc,  char *const argv[ ],  const  char  *optstring )

extern char *optarg ;
extern int optind, opter ;
```

La fonction **getopt** place dans la variable **optind** l'indice dans **arvg** du prochain argument à traiter. Quand toutes les options sont traitées, la fonction **getopt** renvoie **EOF**.

La chaîne **optstring** décrit les options valides : une lettre seule décrit une option sans argument, une lettre suivie d'un caractère est une option qui admet un argument, séparé ou non par des espaces. La variable **optarg** pointe alors vers sur le texte de l'argument.