Système TD 6

EXERCICE 1

Écrire un programme qui crée un thread prenant en paramètre une structure contenant un tableau d'entiers et l'affiche dans le terminal (voir **rappel**).

EXERCICE 2

Écrire un programme qui utilise une structure contenant un entier initialisé à 10.
Ce programme va lancer 4 threads en **parallèle**.

- Le premier thread ajoute 4 à la variable.
- Le deuxième thread divise par 2.
- Le troisième thread multiplie par 4.
- Le quatrième thread retranche 5.

Lancer plusieurs fois le programme, vous devriez avoir des résultats différents.
(Ne pas oublier d'afficher les valeurs intermédiaires).

EXERCICE 3

Écrire un programme qui crée une structure contenant un tableau d'entiers, le nombre d'éléments du tableau.
Le programme demande à l'utilisateur la taille de tableau désirée et stocke cette information dans la structure.
Créer le tableau avec ***malloc*** et l'initialiser avec des 0.
Votre programme lancera le premier thread qui changera les valeurs du tableau par des valeurs aléatoires comprises entre 1 et 5, puis votre programme lancera un second thread pour afficher le contenu du tableau. Les deux threads doivent s'exécuter en **parallèle**.

Modifier votre code pour que l'affectation des cases du tableau attende une seconde entre chaque case affectée. Le second thread affiche le tableau toutes les 2 secondes.

EXERCICE 4 : protection du tableau

Modifier l'exercice 3 afin de vous assurer que chaque thread accède seul à une case du tableau (Il faut utiliser un **mutex,** voir le **rappel**).

EXERCICE 5 : protection d'accès

Écrire un programme qui remplit un tableau (dans une structure) avec des valeurs aléatoires et l'affiche. Votre programme lance *n* threads, chaque thread doit avoir un numéro unique (incrémenter un compteur).
Toutes les secondes chaque thread affiche son numéro, une position aléatoire dans le tableau et le contenu de la case.

RAPPELS

Lancer un thread

```
typedef struct {
    int variable;
} Donnees;

void* maFonction(void* argument) {
    Donnees* donnees = (Donnees*) argument
    printf("je suis un thread, variable = %d \n", donnees->variable);
    pthread_exit(NULL);
}

int main(int argc, char* argv[]) {
    pthread_t monThread;
    Donnees donnees;
    donnees.variable = 10;
    pthread_create(&monThread, NULL, maFonction, (void*) &donnees);
    pthread_joint(monThread, NULL);
}
```

Utiliser un mutex

```
pthread_mutex_t mutex; // le mutex en variable globale
pthread_mutex_init(&mutex,NULL); // initialisation du mutex dans le main
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
```