[TD-2] Familiarisation avec l'API multitâches **pthread**

a) Exécution sur plusieurs tâches sans mutex

On reprend la fonction `void incr(unsigned int nLoops, double* pCounter)` définie au **TD-1b** ainsi que la fonction `main` correspondante permettant de spécifier la valeur initiale de `nLoops` en ligne de commande. On rajoute un 2e argument au programme correspondant au nombre de tâches à exécuter ; `argv[2]` est donc une chaine de caractères représentant la valeur décimale du nombre de tâches que la fonction `main` doit créer et exécuter ; soit `nTasks` la variable de type `unsigned` recevant la valeur binaire de `argv[2]`. Les actions supplémentaires que la fonction `main` doit donc effectuer sont les suivantes :

– lancer dans autant de tâches qu'indiqué par `nTasks` l'exécution d'une fonction `call_incr` appelant `incr` ;

– attendre la fin de l'exécution de toutes les tâches à l'aide de la fonction `pthread_join` avant d'imprimer à l'écran la valeur du compteur et le temps total d'exécution.

> Modifiez la fonction `main` et implémentez la fonction `call_incr` conformément aux indications ci-dessus.
> Exécutez le programme plusieurs fois et notez à chaque fois la valeur finale du compteur `counter`.
> Qu'en concluez-vous ?

b) Mesure de temps d'exécution

Nous allons maintenant exécuter le code développé au (a) avec un ordonnancement temps réel. Pour cela il faut rajouter un paramètre en ligne de commande spécifiant l'ordonnancement (`SCHED_RR`, `SCHED_FIFO` ou `SCHED_OTHER` ce dernier étant la valeur par défaut) ; le choix devra fixer la valeur d'une variable `schedPolicy` de type `int`.
Pour appliquer l'ordonnancement choisi à la fonction `main`, vous devrez utiliser les fonctions `pthread_setschedparam()` et `pthread_self()` vues dans le cours (Multitâches-1-Bases-v7, slide 22).

> Rajoutez le code pour ce nouveau paramètre d'ordonnancement en ligne de commande ;
> Appliquez à la fonction `main` l'ordonnancement choisi ; donnez la priorité maximale à `main` dans le cas ou l'ordonnancement est de type temps réel.
> Lancez les tâches avec un attribut Posix spécifiant une priorité inférieure à la priorité du `main` (0 si l'ordonnancement choisi est `SCHED_OTHER`).

> Exécutez le programme en choisissant un l'ordonnancement `SCHED_RR`. En utilisant la fonction `clock_gettime` vue au **TD-1b**, exécutez le programme pour les valeurs suivantes et faites un graphique des temps d'exécution (24 cas en tout) :
> `nLoops` $\in \{10^7, 2\times10^7, 3\times10^7, 4\times10^7\}$ et `nTasks` $\in \{1, 2, 3, 4, 5, 6\}$
> Que concluez-vous sur l'architecture processeur de votre cible ?

c) Exécution sur plusieurs tâches avec mutex

Reportez-vous au cours pour l'utilisation des mutex Posix.
On rajoute un 3e argument `protected` au programme correspondant à l'utilisation d'un mutex ou pas : si

l'argument `protected` est spécifié en ligne de commande, alors l'incrémentation de `counter` doit être protégée par un mutex, sinon, elle doit se faire sans protection (équivalent de la section précédente).

> Exécutez le programme plusieurs fois avec et sans la protection du mutex et notez à chaque fois :
> – la valeur finale de `counter` ;
> – le temps d'exécution.
> Qu'en concluez-vous ?