HLIN603 - TD/TP pour 3 séances de 3h
Threads en C++

L'objectif de cette série d'exercices est d'apprendre à manipuler des threads en C++11 pour mettre en oeuvre des calculs parallèles. Nous commençons par des applications simples et sans accès concurrents à des données, avant de mettre en oeuvre des accès concurrents et la synchronisation.

Vous êtes vivement encouragés à adopter les bonnes habitudes de programmation en C++ apprises dans le cadre globale de toute l'UE.

# 1 Débuter avec la notion de thread

1. Créer deux fonctions :
   - **afficheO()** qui affiche une suite de caractère 'O', de taille 100.
   - **afficheK()** qui affiche une suite de caractère 'K', de taille 100.
2. Ecrire un programme principal qui appelle ces deux fonctions (afficheO() puis afficheK()). Exécuter ce programme. Ce programme est-il multi-tâche ?
3. Modifier maintenant votre programme principal pour qu'il crée deux tâches (threads), exécutant les deux fonctions précédentes (chaque tâche exécutera une fonction). Exécuter ce programme. Est-il multi-tâche ?
4. Copier votre programme dans un nouveau fichier. Modifier le code de manière à ce que la taille de la suite de caractères à afficher soit un argument des deux tâches. Tester votre programme (la taille est à saisir au clavier).
5. Modifier le code de manière à ce que les fonctions exécutées par les tâches appartiennent à des classes. Pour répondre, utiliser les différentes approches vues en cours (pensez à copier vos fichiers avant toute modification). Tester vos programmes.
6. Enfin, modifier votre code pour qu'il n'y ai qu'une seule fonction d'affichage. Les arguments de cette nouvelle fonction seront donc, le caractère et le nombre de caractère à afficher.

# 2 Première parallélisation

1. Ecrire un programme qui calcule la somme des éléments d'un tableau d'entiers.
2. Modifier ce programme pour qu'il divise le tableau en 4 parties de même taille, calcule la somme de chaque partie et enfin, déduit la somme finale à partir des sommes partielles. Testez votre programme.
3. A partir de cette dernière version, paralléliser votre code. Dans votre réponse, éviter au mieux les variables globales et favoriser le passage de paramètres.

# 3 Futures

Reprendre l'exercice 2 et modifier le code pour faire en sorte d'utiliser des objets "future" pour récupérer les sommes partielles calculées par les tâches.

# 4 Accès concurrent : problème et solution

Cet exercice illustre l'importance des verrous à partir d'un exemple simple.

1. Soit une variable globale **total** initialisée à 0 et une fonction **void carre(int)** qui ajoute à **total** le carré de l'entier passé en paramètre. Ecrire un programme qui lance 20 tâches (threads) (i = 1, 2, ..., 20) ; où la tâche i exécute carre(i). Après l'exécution de ces tâches, le programme affiche le résultat final (**total**).
2. Lancer votre programme
   - une fois
   - 20 fois : en supposant que votre exécutable est appelé exo3, exécuter la commande : "for i in {1..40} ; do ./exo3 ; done"
   - 100, 500 et 1000 fois. Vous pouvez ajouter un tri des résultats à l'aide de la commande : "for i in {1..40} ; do ./exo3 ; done | sort | uniq -c"
3. Sachant que le résultat attendu est 2870, expliquer ce qui a pu se produire à l'exécution.
4. Corriger votre code pour éviter ce phénomène et tester votre code modifié à l'aide des commandes précédentes.

# 5 Mutex : Où est la baguette ?

Le problème du dîner des philosophes est un problème classique de partage des ressources en programmation concurrente. La situation est la suivante :

- 5 philosophes sont assis autour d'une table,
- chaque philosophe a devant lui une assiette de riz et pour manger, il a besoin de deux baguettes,
- Il y a en tout 5 baguettes, chacune se trouve à gauche de chaque assiette,
- Chaque philosophe a le comportement suivant : il pense, puis quand il a envie de manger, il prend la baguette de gauche, puis la baguette de droite et quand il termine, il pose les deux baguettes et il recommence à penser et ainsi de suite

Télécharger le fichier "philosophes.cc" depuis Moodle (cours HLIN603-concurrence). Il contient une tentative de modélisation de ce problème où chaque philosophe est représenté par un thread et chaque baguette par un verrou (mutex).

1. Exécuter ce programme. Que se passe-t-il ? Comment appelle-t-on ce phénomène ?
2. Proposer et implémenter une solution à ce problème. Si vous avez une solution mais avez des difficultés à l'implémenter, demandez à votre chargé(e) de TD.

# 6 Synchronisation : Producteur-Consommateur

Télécharger le fichier "prodcons.cc" depuis Moodle.

1. Exécuter ce programme plusieurs fois. Qu'observez vous ?
2. Utiliser les outils de synchronisation pour résoudre le problème.

# 7 De retour à table

Reprendre l'exercice 5 et proposer une implémentation dans laquelle :

- Les baguettes sont des objets.
- La synchronisation est réalisée en utilisant un seul verrou et une variable conditionnelle.

## 8 Cowabunga !

Les tortues Ninja raffolent des pizzas. Un jour, Donatello décide de manger autant de pizzas que possible. Toutefois, il ne peut en manger qu'une à la fois. Il a appelé 9 services de livraison de pizzas, pour être sûr de ne pas perdre un instant. Chaque livreur de pizza est capable d'apporter une nouvelle pizza en 60 millisecondes. Enfin, le séjour étant petit, il ne permet pas de stocker plus de 35 pizzas, dans ce cas, les livreurs attendent que Donatello fasse de la place en mangeant des pizzas.

Ecrire un programme implémentant ce scénario, avec un thread pour Donatello et 9 threads pour les 9 livreurs de pizzas, chacun livrant une pizza toutes les 60 millisecondes. On suppose que Donatello utilise 10 millisecondes pour respirer entre chaque pizza.

## 9 Optionnel : Threads en Java

Se documenter sur l'utilisation des threads en java et reprendre les exercices de votre choix pour les réaliser en Java.