Travaux Pratique 1

Ce premier sujet est en lien avec le *premier chapitre du cours*, et concerne la ***programmation concurrente***. L'idée est de manipuler des threads, et de gérer correctement les sections critiques soit directement avec des sémaphores, soit via un ou des moniteurs. Le problème du grain (nombre de tâches résultant du découpage du problème) est à considérer, n'hésitez pas à tester différentes tailles.

Votre travail est à rendre (il le sera à chaque fois) sous la forme du code (et uniquement la partie « student ») accompagnée d'un rapport au format PDF, le tout dans une archive compressée au format ZIP. *Ne respecter pas ces contraintes, et votre note en sera diminuée de quelques points*.

Vous ne devez modifier que ce qui se trouve dans le répertoire « `./student/` » !

Notez que chaque exercice vient avec un squelette, qui s'occupe de la ligne de commande, du lancement de votre code (défini dans une classe particulière), et d'une vérification sommaire du résultat (lorsque c'est possible). Utilisez l'option « `-h` » ou « `--help` » pour connaître le fonctionnement de la ligne de commande …

Deux scripts sont fournis pour compiler : sous Windows, il s'agit de « `./build.bat` » qui génère les exécutables dans le répertoire « `./win32/Release` » ; pour les autres systèmes, utilisez « `./build.sh` » qui génère les exécutables dans « `./linux/` ». Ces scripts utilisent « `cmake` », et soit `GCC` sous linux/mac, soit `Visual C++ 19` sous Windows. Leur comportement a été testé sous Windows, WSL/Debian, et Linux Mint.

Exercice 1

Reprenez le premier exemple vu en cours sur les threads C++ 11. Compilez puis testez votre programme. Dans le rapport, répondez (encore) à la question posée en cours (combien de threads au total). N'hésitez pas à consulter la documentation en ligne …

Exercice 2

Reprenez le second exemple sur les threads vu en cours (calcul de π avec la formule de Liebniz). Que se passe-t-il avec 2 threads ? Avec 4 ? Avec 8 ? Avec 16 ? Notez la variabilité des temps de calcul, qui impose de faire des statistiques (moyenne, écart-type) dans le rapport.

Exercice 3

Dans l'exercice précédent, chaque thread utilise une case mémoire unique par thread pour écrire son résultat. Modifiez ce fonctionnement pour n'utiliser qu'une seule variable globale.

Autrement dit, la somme finale des calculs partiels effectués par chaque thread et qui était réalisée dans le thread principal doit disparaître. A la place, chaque thread ajoute son résultat dans la variable globale.

Attention : cette variable globale étant manipulée par tous les threads, il convient de la protéger. Son accès en écriture doit donc être effectué en **section critique**. Pour rappel, une section critique doit être encadrée par une étape d'entrée en zone d'exclusion mutuelle et une étape de sortie de zone d'exclusion mutuelle. Pour cela, utilisez un sémaphore booléen, plus connu comme « mutex » (les sémaphores à compteur sont une nouveauté de C++ 20).

Exercice 4

Reprenez l'exercice précédent, en ajoutant une instance d'une classe de type **moniteur** pour le stockage et la manipulation d'un résultat de calcul partagé. Votre classe stocke le résultat dans un membre donnée privé, dont l'accès est toujours effectué en section critique.

Est-ce que deux méthodes d'accès en lecture et en écriture suffisent ? Remarquez le lien avec la notion de lecteur/rédacteur …

Exercice 5

Cet exercice concerne les nombres premiers « jumeaux ». Deux nombres (n, m) sont premiers jumeaux si et seulement si n et m sont premiers, et $m = n + 2$. Par exemple les nombres (3,5) sont premiers jumeaux. De même pour (5,7), (11,13), ou encore (1000139,1000141).

L'objectif de cet exercice est de calculer tous les couples de nombres premiers jumeaux dans un intervalle donné par l'utilisateur (options `-s=xxx` et `-e=yyy`).

Le calcul est effectué par n threads. Ils produisent des ressources (nombres premiers), et ils testent des nombres. L'intervalle à tester est géré par un moniteur, afin de garantir l'équilibrage de charge. Le thread principal consomme les données (il les renvoie dans l'ordre une fois le calcul terminé). Donc, il doit récupérer les nombres premiers jumeaux produits, les trier et les retourner. Le tri entre deux couples peut se faire en ne considérant que leur première valeur (l'écart étant toujours égal à 2).

Utilisez un premier moniteur pour le stockage des nombres premiers générés, et un second pour la consommation de l'intervalle.