TD 3 : Thread, Lock

Objectifs pédagogiques :

- **thread**
- **atomic, mutex**
- **section critique**

Introduction

Dans ce TD, nous allons aborder l'utilisation des thread en C++ pour faire de la programmation concurrente. L'exercice permet de voir plusieurs mécanismes que l'on peut employer pour protéger les données critiques des accès concurrent.

Pour être sûr de ne pas entrer en conflit avec d'autres applications, nous utiliserons le namespace `pr` pour notre implémentation.

1.1 Création de Thread

Question 1. Ecrire une fonction `void work(int id)` qui affiche son identifiant *id*, puis dort pendant une durée aléatoire comprise entre 0 et 1000 ms, puis affiche un deuxième message.

Question 2. Ecrire une fonction `void createAndWait(int N)` qui crée N threads exécutant `work` avec pour identifiant leur ordre de création compris entre 0 et $N-1$, puis attend qu'ils soient tous terminés. On souhaite maximiser la concurrence.

Question 3. Si l'on exécute le programme, quels sont les entrelacements possibles ?

1.2 Variables partagées.

On considère une classe Compte munie d'un attribut entier représentant le solde, d'un constructeur positionant le solde initial, d'une opération membre `void crediter (int val)` qui ajoute de l'argent au solde, et d'une opération membre `int getSolde() const`.

Question 4. Ecrivez cette classe Compte.

Question 5. Ecrivez une fonction `void jackpot (Compte * c)` qui crédite 10000 pièces d'or sur le compte fourni, mais une par une (ding, ding, ding...). Ensuite dans un main, instancier un compte, et créer N threads qui exécutent le code de la fonction Jackpot.

Question 6. Quel sera le solde du compte à la fin du programme pour N=10 ?

Question 7. Si le nombre de pièces d'or du jackpot est faible (disons 100), on n'observera pas de problème en général sur cet exemple, expliquez pourquoi.

Question 8. Comment utiliser un `atomic` pour corriger ces problèmes ? Expliquez l'effet au niveau des entrelacements possibles.

1.3 Section Critique.

On ajoute au Compte de la question précédente une opération : `bool debiter (int val)` qui doit controler que le solde du compte est suffisant (la banque ne prête pas), et si c'est le cas réduire le solde du montant indiqué. La fonction rend vrai si le débit a eu lieu.

Question 9. On reprend l'exemple du Jackpot, mais cette fois-ci en débit, le LosePot retire 1000 par paquets de 10. On lance N thread qui exécutent cette fonction ; le compte peut-il tomber en négatif ? Quelle garantie fournit `atomic` ici ?

Question 10. Introduire un mutex dans la classe Compte pour sécuriser son utilisation dans un contexte Multi-Thread.

Question 11. Utiliser un `lock_guard` plutôt que l'API lock/unlock. Comparer les syntaxes.

Question 12. Dans les versions avec un mutex, qu'apporte l'utilisation d'un atomic pour l'attribut solde ? Si l'on n'utilise pas `atomic` est-ce nécessaire de protéger la méthode `getSolde` avec le mutex ?

1.4 Transaction.

On considère à présent une Banque, possédant en attribut un ensemble de K comptes (un `vector<Compte>`) initialement avec un solde de $SOLDEINITIAL$ chacun. Elle est munie d'une opération `bool transfer(int idDebit, int idCredit, size_t val)` qui essaie de débiter le compte d'indice $idDebit$ de val, et si c'est un succès, crédite le compte $idCredit$ du même montant val. La fonction rend vrai si le transfert est un succès. On crée N threads de transaction, qui bouclent 1000 fois sur le comportement est suivant :

- Choisir i et j deux indices de comptes aléatoires, et un montant aléatoire m compris entre 1 et 100.
- Essayer de transférer le montant m de i à j.
- Dormir une durée aléatoire de 0 à 20 ms.

Question 13. L'utilisation de la classe Compte dans un `vector` pose un problème : le compilateur se plaint que la classe n'est pas copiable. Expliquez le problème et corrigez le.

Question 14. Le comportement sera-t-il correct (pas de *datarace*) avec les protections actuelles sur le Compte ?

On estime que découpler les débits des crédits est une faute, on souhaite au contraire que la mise à jour des deux comptes concernés soit atomique : soit le transfert a lieu et les deux comptes sont mis à jour (simultanément du point de vue d'un observateur), soit il n'a pas lieu. A aucun moment un observatuer ne doit pouvoir voir un des comptes débités et l'autre pas encore crédité.

Question 15. Proposez une stratégie de synchronisation pour permettre ce comportement transactionnel. On ajoute un accesseur `mutex & getMutex()` au compte pour permettre de l'écrire, ou l'on définit les trois méthode `void lock () const, void unlock() const, bool try_lock () const` par délégation sur le mutex stocké.

Question 16. Le programme se bloque immédiatement, même avec un seul thread qui fait des transferts. Pourquoi ?

Question 17. Après correction du problème précédent à l'aide de `recursive_mutex`, on introduit plusieurs thread faisant des transferts, mais de nouveau on observe parfois un interblocage, le programme entier se fige. Expliquez pourquoi et corriger le problème.

1.5 Comptabilité

La banque possède aussi un thread qu'elle detache à la création pour faire la comptabilité. Ce thread itère sur les comptes en sommant leur solde, et vérifie que la somme vaut bien $SOLDEINITIAL*K$. Sinon il lève une alerte sur la console.

Question 18. Le thread comptable sera-t-il satisfait avec les synchronisations actuelles ? Expliquez pourquoi.

Question 19. Ajoutez un mutex dans la banque, et les synchronisations utiles pour que le thread comptable obtienne les bons résultats.

Question 20. Avec un seul mutex dans la banque, la concurrence entre les thread de transfert n'est plus possible. De fait les mutex définis dans Compte ne servent plus à rien dans ce scenario. Proposez une autre approche qui réutilise les locks de Compte plutôt que de n'avoir qu'un seul lock.

TME 3 : Thread, mutex

Objectifs pédagogiques :

- **thread**
- **atomic simples**
- **mutex**

1.1 Synchronisations de la Banque

Question 1. En suivant l'énoncé du TD3, implanter les exemples de synchronisation. On n'hésitera pas à copier coller et changer de namespace au fil des questions. Pour chaque exemple, lancer plusieurs fois l'exemple en faisant varier le nombre de threads et les constantes de manière à observer des exécutions diverses.

Activer `thread` dans Eclipse CDT.
Par défaut le link n'active pas l'option adaptée `-pthread`.
Pour le régler le compilateur, on doit faire un réglage pour chaque projet séparément. En partant d'un clic droit sur le projet, accéder à ses propriétés:

- "Project properties -> C/C++ Build -> Settings"
- Sous le "GCC C++ linker" on trouve une rubrique "General"
- Cocher : "Support for pthread (-pthread)" dans la liste

Attention, il faut faire ce réglage dans tout nouveau projet utilisant des thread.