Auteurs: Auberson Kevin, Alexandre Shyshmarov

Description des fonctionnalités du logiciel

Le logiciel implémente un système de calcul différé où des requêtes de calcul sont soumises à un gestionnaire de calcul. Les requêtes sont de trois types différents (A, B, C), et elles sont stockées dans des files d'attente distinctes en fonction de leur type. Les calculs sont effectués par des threads de calcul, et les résultats sont stockés dans un vecteur dans l'ordre d'achèvement.

Choix d'implémentation

Managment des requêtes

Pour le stockage des requêtes, on utilise un vecteur de queue. Chaque queue est assignée à un type de requête. Ce qui nous permet avec l'enum class ComputationType de sélectionner la bonne queue dans le vector.

La même approche a été adopté pour les variables Condition pour la mise en attente de thread.

En revanche pour les résultats, on utilise un vecteur simple qui les stocks selon l'ordre d'arrivée. Même chose pour les requêtes avorté.

Pour la gestion des requêtes et de leurs id, nous avons décidé de faire avec un system de ticket qui nous semblait plus simple à faire. Un id unique est attribuée à chaque requête peu importe son type.

Etape 1

L'implémention de requestComputation ne change pas trop de ce qu'on a vue aux cours. Le haut de la fonction gère l'attente des threads si le buffer est plein. La fin gére la mise de la requet dans le buffer et on signal aux consommateurs pour le type précis de la requet qu'il peut se reveil et on sort de la fonction.
Le code en plus, c'est la gestion de l'arrêt du programme qui se fait juste par un réveil en cascade des thread.

L'implémentation de getWork suit le schéma classique du consommateur. On regarde si le buffer pour la requête est vide ou non. On prend la requête et on la sort du buffer. On signale au producteur du type donnée qu'on a consommé.

La gestion de l'arrêt du programme est la même que pour requestComputation.

Etape 2

La méthode provideResult est plutôt simple de conception. On push le resultat dans le vecteur. On fait un triage pour mettre devant le résultat avec l'id suivant(s'il est présent) et on signal au tread qui attend le résultat qu'il y a eu un nouvel ajout.

Pour la fonction getNextResult(). Le thread appelant se voit attribué le numéro de l'id suivant. Si cet id ne correspond pas à l'id du résultat tout devant du vecteur alors le résultat suivant n'est pas encore disponible. La boucle while est obligatoire même si on utilise un moniteur de Hoare, car à chaque fois qu'on fait un appel, dans a signal dans provideResult, on n'est pas sûr que le résultat avec l'id suivant soit mit.

Typiquement un scénario, ou on attend le résultat avec l'id 1 et que provideResult stock les résultats avec les id 2, 3, 4 ,5...

Etape 3

Dans abortComputation, l'identifiant de la requête annulée est ajouté à une liste. Puis on vérifie si la requête fait déjà partie des résultats si c'est le cas on le supprime du vector resultBuffer. Pour finir, la requête étant annulé, il y a de nouveau de la place pour une nouvelle requête et on réveille la file d'attente selon le type de computation grâce au vector computationResult. La méthode continueWork vérifie si le calcul associé doit se poursuivre en vérifiant si l'identifiant fait partie de la liste des requêtes avortées ou si le programme est arrêté. Nous avons ajouté une vérification des requêtes avortés dans la fonction getWork pour éviter qu'un calculateur prenne une requête avortée.

Etape 4

Dans stop, on fait passer la variable stopped à true pour empêcher l'ajout d'autre Computation. Stopped sert aussi aux threads reveillé à sortir de la condition d'attente. Sans lui, on ne pourrait pas court-circuiter la condition qui met en attente les threads.
Dans la méthode stop, on fait un signal à chaque variable de condition. Ces dernières par cascade vont réveiller toutes les autres. Cela permet de se passer de boucle.

Tests effectués

Tous les tests sont fournis avec le code passe sans problème.

Conclusion

Notre implémentation du système de calcul différé avec le moniteur de Hoare répond aux spécifications du laboratoire. Le code est bien conçu, les tests ont réussi, et le logiciel fonctionne comme prévu. Les différentes étapes de l'implémentation ont été abordées méthodiquement, garantissant une gestion efficace des requêtes, des résultats, et de l'arrêt du programme.
En somme, ce labo nous a permis d'utiliser les moniteurs de Hoare afin d'implementer un systeme de producteur-consommateur avec plusieurs producteurs et plusieurs consommateurs.