Gestion d'accès concurrents

Auteurs: Arthur Junod et Edwin Haeffner

Description des fonctionnalités du logiciel

Ce logiciel simule une logistique entre des :

- Extracteurs de ressources
- Grossistes
- Usines

Chaque objet dispose d'un thread pour son execution.
Et le but de ce laboratoire est de bien identifier les sections critiques du programme et les protéger avec des mutex.

Choix d'implémentation

Comme conseillé par le professeur, nous avons commencé par implémenter la fin de la simulation en ne mettant que des `stopRequest()` sur tous les threads.

Ensuite, nous nous sommes concentrés sur l'identification des sections critiques dans le programme, qui concernent évidemment les endroits où l'argent et les stocks de marchandises sont modifiés. Ces sections doivent donc être protégées.

Initialement, nous avions implémenté plusieurs mutex distincts dans les différentes fonctions de chaque classe, un pour le trading, un pour le building, etc, mais nous nous sommes rendu compte que cette approche pouvait conduire à des accès concurrents sur les mêmes ressources, provoquant potentiellement des problèmes de création/suppression d'argent ou de matériel et qu'elle ne suivait pas la logique demandée par le labo. Le problème est que nous n'avons pris conscience de cela que tardivement, car les accès concurrents sont assez rares ce qui rend dur le test des mutex.

En effet, pour qu'un tel problème survienne, il faut entre autre qu'un objet crée une ressource pendant qu'un autre essaie de trade avec lui. Et vu les nombreuses pauses simulant les interactions dans notre programme, ce cas de figure se produit très rarement, et donc il est très complexe pour nous de le voir lorsqu'on quitte le programme. Il nous est arrivé qu'une seule fois que le programme nous indique qu'il y avait eu une duplication de l'argent lors de tous nos essais...

Finalement, nous avons utilisé un seul mutex déclaré dans la classe parente Seller, dont toutes les autres classes héritent. C'est l'endroit le plus approprié pour le placer.

Une fois que nous avons décidé de n'utiliser qu'un seul mutex, nous nous sommes retrouvé face à un deadlock qui arrêtait les trade entre les usines et les grossistes et qui empêchait la fin de notre programme. Il s'est avéré que le deadlock apparaissait quand une usine et un grossiste essayaient d'échanger des ressources en meme temps. Comme les deux objets bloquaient respectivement leur propre mutex avant de rentrer dans la fonction `trade()` s'ils arrivaient au meme moment à celle-ci, ils restaient bloqué dans la fonction trade de l'autre objet (au `mutex.lock()` de la fonction).

Afin de régler ce problème, nous avons déplacé le `mutex.lock()` de l'objet `Wholesaler` juste après la fonction `trade()` car la seule vérification qui s'effectue avant est `price <= money`. Nous savons que `trade()` ne peut qu'augmenter l'argent donc la vérification ne peut que rester vraie. Cette solution n'est pas parfaite, mais selon l'assistant, elle est valide car une solution plus juste serait trop compliquée par rapport au labo.

Tests effectués

Lorsque l'on quitte le programme, on calcule l'argent et la valeur des ressources de chaque entité, et on vérifie que ce total est égal à l'argent de départ avec lequel la simulation a commencé. Ce test permet de s'assurer que l'accès concurrent aux ressources est bien protégé et qu'il n'y a pas de perte.

Mais comme dit précédemment, ce test est très faillible, car l'accès concurrent est très rare dans notre cas de figure.

On peut également vérifier visuellement pendant l'exécution que chaque usine achète et produit bien les ressources attendues, et que le programme se comporte correctement. C'est de cette façon, en nous aidant aussi du debugger, que nous avons vu le problème avec le mutex unique qui bloquait tout le code.

Il aurait été possible de faire des tests unitaires pour vérifier les accès concurrents, mais nous n'avons pas vraiment les connaissances nécessaires ni le temps pour les mettre en place.

Conclusion

Ce laboratoire nous a permis de bien mettre en pratique les connaissances du cours lié à l'utilisation des mutex. Nous avons pu voir les contraintes liées à celle-ci, notamment les deadlocks. Malheureusement, nous trouvons que le cas de l'usine comme implémenté ne permet pas d'avoir assez d'accès concurrent, car en enlevant tous les mutex, nous avons eu beaucoup de vérifications finales juste.