Thème 4 - TD

Objectifs

— Interface `Lock : lock(), unlock()`
— Interface `Condition : await(), signal()`

Exercices

Exercice 6 – Communication entre tâches, terminaison

Le but de cet exercice est de réfléchir sur l'une des principales difficultés de la programmation concurrente : qui fait quoi, et quand ? Il y a en général plusieurs réponses possibles à ces questions, nous proposons de réfléchir à la construction d'une solution, mais il existe des alternatives qui fonctionneront probablement très bien...

Nous allons appuyer notre réflexion sur une version simplifiée d'un problème de synchronisation classique : le problème du barbier.

Un salon de coiffure peut accueillir dans sa salle d'attente un nombre limité n de clients. Un client qui se présente à l'entrée du salon repart sans attendre si toute les chaises sont occupées. Sinon, il s'installe et attend que le coiffeur puisse s'occuper de lui. Le coiffeur a un comportement répétitif qui consiste, lorsqu'au moins un chaise est occupée, à faire entrer l'un des clients dans la pièce d'à côté pour le coiffer. La chaise devient disponible pour un nouveau client.

Question 1
Quelles sont les classes qui composent ce système ? Quelles sont celles qui devraient implémenter l'interface `Runnable` ?

Question 2
Quelles sont les variables qui définissent l'état de la salle d'attente ? Quelles sont les actions qui modifient ces variables ? Ces actions ont-elles des effets sur d'autres composants du système ?

Question 3
Dans quelle classe les actions identifiées précédemment doivent-elles être implémentées sous forme de méthode ? Les variables qu'elle manipulent doivent-elles être protégées par une section critique ?

Question 4
Quelles sont les conditions qui peuvent bloquer temporairement l'exécution du barbier ? Comment peut-on les implémenter ?

Question 5
Quelles sont les conditions qui peuvent bloquer temporairement l'exécution d'un client ? Comment peut-on les implémenter ?

Question 6
Comment peut-on gérer la terminaison du coiffeur ?

Exercice 7 – Producteur/Consommateur : cas optimisé

Nous reprenons le problème du producteur/consommateur et nous voulons maintenant programmer une solution optimisée qui améliore le parallélisme. La solution que nous avons proposée utilise directement le moniteur de l'instance de `Buffer` pour réaliser l'exclusion mutuelle qui garantit la cohérence de l'état du tampon.

Question 1
Quels sont les avantages et les inconvénients de cette solution ?

Nous souhaitons mettre en place un niveau de synchronisation plus fin pour permettre un meilleur parallélisme lors de l'accès au buffer. Pour cela, nous allons dissocier, dans les méthodes d'accès :
- les manipulations du (des) compteur(s) partagé(s) qui doivent se faire en exclusion mutuelle ;
- l'accès effectif à une case (lecture ou écriture), qui peut se faire en parallèle de l'accès à une autre case par un autre processus.

Question 2
L'exclusion mutuelle pour les accès au(x) compteur(s) est-elle suffisante pour garantir la cohérence des données ? Montrer qu'on peut avoir un producteur et un consommateur qui accèdent à la même case. Peut-on utiliser un mécanisme de synchronisation pour l'empêcher ?

Question 3
Peut-on protéger l'accès aux compteur partagés par un bloc d'instructions synchronisées et utiliser un verrou pour protéger l'accès à une case ?

Question 4
Comment faire pour gérer séparément le blocage des producteurs et celui des consommateurs ? Proposez une implémentation de cette solution.

Question 5
Comment pourrait-on envisager d'améliorer les performances ?