Algorithme de Dekker et algorithme de Peterson - Introduction à la synchronisation

Ensimag 2A

1 Introduction

Le but de ce TD est d'introduire la programmation concurrente en vous amenant à penser avec plusieurs flots d'exécution simultanés.

Le problème choisi ici est un problème qu'ont dû résoudre les utilisateurs des premières machines parallèles au début des années 60. Celles-ci n'avaient pas de dispositifs matériels pour gérer la concurrence, elles étaient juste composées par la juxtaposition de deux unités de calcul séquentiel manipulant la même mémoire partagée.

Nous chercherons dans ce TD à résoudre le problème de l' **exclusion mutuelle** sans support matériel autre que l'ordre des écritures en mémoire.

1.1 Le modèle de machine

La machine est composée de deux processeurs identiques mais exécutant chacun son propre flot d'instructions. Un flot d'exécution ne peut pas être déplacer d'un processeur à l'autre. Les deux processeurs accèdent directement à la même mémoire physique. Nous ne prendrons pas en compte ici les problèmes de caches, de protection d'accès ou de mémoire virtuelle.

1.2 La nécessité d'une isolation

Question 1 *Quel est le code* pseudo-assembleur *de l'instruction* $i++$ *?*

Dans le cas général, si l'un des processeurs fait $i++$ la valeur de la variable est incrémentée de 1. Si chacun des deux processeurs fait $i++$, nous nous attendrions à ce que la variable soit incrémentée de 2.

Question 2 *Quel scénario d'exécution de* $i++$ *produit un résultat incohérent ?*

Question 3 *Quelle solution envisager pour éviter ce problème ?*

On nomme une zone de code qui ne doit être exécutée que par un seul processeur à la fois une **section critique** et on parle alors d'une exécution en **exclusion mutuelle**.

Dans la suite de ce TD, nous allons chercher à produire une exécution en exclusion mutuelle pour une section critique (par exemple $i++$). Nous nous intéressons aux deux fonctions $entreeSC()$ et $sortieSC()$ telles que :

```
entreeSC();
...
-- Section critique
...
sortieSC();
```

Question 4 *Quel est l'équivalent d'une structure while en GOTO ?*

Nous utiliserons uniquement *GOTO* dans la suite de ce TD.

Question 5 *Pour cette seule question, nous utiliserons une fonction atomique "bool* $scUsed()$*" qui renvoie* $false$ *pour le premier processeur appelant. La fonction* $resetScUsed()$ *permet de réinitialiser l'état interne de la fonction* $scUsed()$ *afin que celle-ci renvoie à nouveau* $false$ *lors du prochain appel. Ecrire* $entreeSC()$ *et* $sortieSC()$ *en utilisant* $scUsed()$ *et* $resetScUsed()$*.*

Question 6 *Quel est le nom de la stratégie que vous avez utilisée dans* $entreeSC()$ *de la question précédente ?*

2 L'exclusion mutuelle "software"

2.1 Introduction

Nous allons maintenant implémenter l'exclusion mutuelle en utilisant différentes stratégies. Pour chacune d'entre elles, vous devez suivre le cahiers des charges concernant les fonctions `entreeSC` et `sortieSC`. Nous étudierons les avantages, les inconvénients, ainsi que les erreurs de chaque stratégie.

2.2 Utilisation d'un booléen d'occupation

Un processeur qui rentre dans la section critique va modifier un *bool* nommé *occ* pour indiquer qu'il l'utilise.

Question 7 *Ecrire entreeSC et sortieSC.*

Question 8 *Ce cahier des charges permet-il d'avoir une exclusion mutuelle ? Justifier.*

2.3 Chacun son tour

Nous allons maintenant avoir besoin de différencier les processeurs lors de l'exécution du code. Pour cela, nous utiliserons dans le pseudo-code les fonctions $myID()$ et $otherID()$ renvoyant respectivement l'identifiant du processeur qui exécute le code et l'identifiant de l'autre processeur.

Un processeur qui veut rentrer dans la section critique doit attendre que ce soit son tour. Une fois la section critique exécutée, il donne la main à l'autre.

Question 9 *Ecrire entreeSC et sortieSC.*

Question 10 *Ce cahier des charges permet-il d'avoir une exclusion mutuelle ? Existe-t-il d'autres problèmes dans cette stratégie ? Justifier.*

2.4 Annonce de sa volonté d'entrer en SC

Dans les stratégies précédentes un processeur indique son entrée dans la section critique. Dans cette stratégie, les processeurs signalent seulement leur souhait d'entrer en section critique. Pour cela, nous utiliserons un tableau de 2 booléens indexé par $myID()$ et $otherID()$. Un processeur souhaitant entrer en section critique attend que l'autre processeur n'ait plus la volonté d'entrer avant d'y entrer vraiment.

Question 11 *Ecrire entreeSC et sortieSC.*

Question 12 *Ce cahier des charges permet-il d'avoir une exclusion mutuelle ? Existe-t-il d'autres problèmes dans cette stratégie ? Justifier.*

2.5 Annonce de sa volonté, avec renonce

Comme pour le cahier des charges précédent, un processeur annonce sa volonté de rentrer dans la section critique. Cependant, pour cette section, un processeur n'ayant pas réussi à rentrer car l'autre a également fait l'annonce de sa volonté de rentrer annule son annonce et recommence.

Question 13 *Ecrire entreeSC et sortieSC.*

Question 14 *Ce cahier des charges permet-il d'avoir une exclusion mutuelle ? Existe-t-il d'autres problèmes dans cette stratégie ? Justifier.*

2.6 Demande et renonce si ce n'est pas mon tour

Dans les sections suivantes, nous utiliserons à la fois la notion de demande via un tableau de booléens et la notion de tour.

Le processeur fait une demande pour entrer en section critique et renonce si l'autre processeur a également fait la demande et qu'il ne s'agit pas de son tour. Sinon, le processeur rentre.

Question 15 *Ecrire entreeSC et sortieSC.*

Question 16 *Ce cahier des charges permet-il d'avoir une exclusion mutuelle ? Existe-t-il d'autres problèmes dans cette stratégie ? Justifier.*

2.7 Dekker 1966

Les deux points importants de la solution proposée par Dekker en 1966 sont :

- si un processeur renonce, il attend son tour avant de redemander
- si c'est son tour, le processeur attend que l'autre ait renoncé avant d'entrer

Question 17 *Ecrire entreeSC et sortieSC.*

2.8 Peterson 1981

En 1981, Peterson a proposé une solution avec un code plus "simple". L'idée est d'utiliser une variable `dernier` contenant l'identifiant du dernier processeur à vouloir entrer dans la section critique. C'est alors à lui d'attendre. Nous nous détachons de la notion de tour.

Question 18 *Ecrire entreeSC et sortieSC.*

3 Implantation

Question 19 *D'un point de vue implantation (produire un code C ou C++ qui fait le pseudo code vu plus tôt), remarquez-vous des éléments qui peuvent être problématiques ?*