Licence 3ème année
Année universitaire 2012-2013

UE Systèmes

TD sur les processus

1 Le dîner des philosophes

Le dîner des philosophes est un problème classique de programmation distribué, inventé par Dijkstra en 1971.

Cinq philosophes sont assis autour d'une table circulaire, et ont commandé un plat (supposé infini) de spaghettis. Cinq fourchettes sont disposées autour de la table, une entre chaque assiette. Chaque philosophe passe alternativement dans trois états : *en train de penser*, *affamé*, *en train de manger*. Tout d'abord, il pense pendant un temps aléatoire. Puis, il est affamé et attend que la fourchette à sa gauche et que la fourchette à sa droite se libèrent. Ensuite, il prend les deux fourchettes et commence à manger pendant un temps aléatoire. Quand il a fini de manger, le philosophe pense (jusqu'à ce qu'il soit à nouveau affamé).

Question 1 *Pourquoi chaque philosophe est-il modélisé par un processus ? Est-ce qu'autre chose est modélisé par un processus ?*

Exercice 1 *Décrivez le pseudo-code d'un programme qui lance cinq processus fils.*

1.1 Solution avec interblocage

Question 2 *Quelles sont les ressources critiques ?*

Question 3 *Quand un philosophe doit-il demander la resource ?*

Question 4 *Quand un philosophe doit-il libérer la resource ?*

Question 5 *Quelle est la section critique ?*

Question 6 *Pour protéger l'utilisation des fourchettes dans la section critique, nous allons implémenter une sémaphore par fourchette. Décrivez le pseudo-code d'un programme qui implémente le dîner des philosophes de la manière suivante :*

- *chaque philosophe pense pendant un temps aléatoire,*
- *chaque philosophe mange pendant un temps aléatoire,*
- *l'accès aux ressources critiques est protégé.*

Question 7 *Qu'est-ce qu'un interblocage ?*

Question 8 *Est-ce que votre solution est susceptible de faire apparaître un interblocage ?*

Exercice 2 *Est-ce que l'interblocage est susceptible d'arriver en pratique ?*

Question 9 *D'où vient l'interblocage ?*

Question 10 *Comment faire en sorte que le problème soit résolu sans interblocage ?*

1.2 Une solution sans interblocage

Dans cette partie, nous allons considérer que le nombre de philosophes qui mangent simultanément est limité.

Question 11 *Chaque philosophe ne peut prendre que la fourchette à sa gauche et à sa droite. Combien de philosophes sont autorisés à manger simultanément ? Écrivez le pseudo-code correspondant.*

Question 12 *Chaque philosophe peut à présent prendre n'importe quelle fourchette sur la table. Combien de philosophes sont autorisés à manger simultanément ? Écrivez le pseudo-code correspondant.*

2 Communications entre processus

Question 13 *Il existe plusieurs mécanismes de communications entre processus : les fichiers, les signaux, les sockets, les tubes nommés, les tubes anonymes, ou encore les segments de mémoire partagée. Pour chacun de ces mécanismes, indiquez :*

- *si les processus communiquants peuvent être sur des machines différentes,*
- *si les processus communiquants doivent avoir une relation de parenté,*
- *si les opérations de lecture et d'écriture doivent être réalisées en exclusion mutuelle.*

3 Utilisation des sémaphores

Un sémaphore est une variable munie de trois opérations :

- l'initialisation,
- P (pour *proberen* en néerlandais, signifiant tester (moyen mnémotechnique : prendre)),
- V (pour *verhogen* en néerlandais signifiant incrémenter (moyen mnémotechnique : vendre)).

Les sémaphores permettent de réaliser plusieurs comportements classiques en programmation multi-processus, comme nous allons le voir.

Question 14 *L'exclusion mutuelle consiste à empêcher plusieurs processus d'exécuter une certaine partie d'un programme. Par exemple, l'accès à une ressource partagée (comme une imprimante, un fichier, une variable, etc) est généralement réalisé en exclusion mutuelle. Implémentez l'exclusion mutuelle avec des sémaphores.*

Question 15 *La barrière de synchronisation consiste à forcer tous les processus à atteindre un certain point du programme avant de continuer leur exécution. Donnez un exemple d'utilisation d'une barrière de synchronisation.*

Question 16 *On considère une barrière de synchronisation particulière : les processus A et B s'exécutent indépendemment, mais une certaine partie du code de B doit obligatoirement s'exécuter après une certaine partie du code de A. Implémentez cette barrière particulière.*

Question 17 *Considérez l'algorithme 1. En supposant que cet algorithme n'est utilisé qu'une fois dans le programme, montrez qu'il réalise une barrière de synchronisation. Vous pourrez utiliser la démarche suivante :*

- *À quoi sert le sémaphore mutex ?*
- *Que font les $N-1$ premiers processus ? Comment garantit-on qu'ils attendent ?*
- *Que fait le N-ème processus ?*

Algorithm 1 Algorithme réalisant une unique barrière de synchronisation (source : wikipedia, article sur les barrières de synchronisation).

Require: *mutex* initialisé à 1, *wait* initialisé à 0, nb initialisée à 0, N le nombre de processus

```
P(mutex)
nb ← nb + 1
if nb = N then
    for i de 1 à N − 1 do
        V(wait)
    end for
    nb ← 0
    V(mutex)
else
    V(mutex)
    P(wait)
end if
```

Question 18 *Quand il est utilisé plusieurs fois dans un programme (pour réaliser plusieurs barrières successives), l'algorithme 1 peut causer des erreurs. Expliquez pourquoi.*

Question 19 *Considérez l'algorithme 2. Montrez qu'il réalise une barrière de synchronisation correcte (même lorsqu'il est utilisé plusieurs fois successivement). Quel était le rôle du sémaphore end ajouté ?*

Algorithm 2 Algorithme réalisant une barrière de synchronisation (source : wikipedia, article sur les barrières de synchronisation).

Require: *mutex* initialisé à 1, *wait* initialisé à 0, *end* initialisé à 0, *nb* initialisé à 0

```
P(mutex)
nb ← nb + 1
if nb = N then
    for i de 1 à N − 1 do
        V(wait)
    end for
    for i de 1 à N − 1 do
        P(end)
    end for
    nb ← 0
    V(mutex)
else
    V(mutex)
    P(wait)
    V(end)
end if
```