TD 4 : Sur l'utilisation de primitives de synchronisation

Exercice 1 : Conditions de concurrence et exclusion mutuelle

Nous nous intéressons dans ce TD au problème d'allocation de ressources banalisées (par exemple allocation de pages de mémoire dans un système paginé). Nous supposons disposer de plusieurs exemplaires d'une même ressource, dont un nombre quelconque peut être demandé par un processus à un moment donné et nous nous plaçons dans un contexte de multiprogrammation : plusieurs processus ou tâches peuvent exécuter « simultanément » des codes partagés. Tout processus ou tâche qui demande des exemplaires de cette ressource doit solliciter les services d'un allocateur via deux procédures :

- *request(m)* : pour demander l'allocation de *m* exemplaires de la ressource
- *release(m)* : pour libérer les *m* exemplaires de la ressource

L'allocateur maintient une variable qui compte le nombre d'exemplaires disponibles afin de satisfaire éventuellement de nouvelles demandes.

Comportement de l'allocateur :

```
entier NbRessDisponibles := Max ;

Procédure request (entier m) {
   Tant que (NbRessDisponibles<m) faire attente le processus appelant ;
   NbRessDisponibles = NbRessDisponibles – m ;
}

Procédure release(entier m) {
   NbRessDisponibles = NbRessDisponibles + m ;
 }
```

Comportement d'une tâche :

```
Tant que (true) {
   request(m) ;
   utiliser les m ressources
   release(m) ;
}
```

Question 1 : *Identifier un exemple d'incohérence dans ce pseudo-programme.*

Pour éviter le problème précédent, on suppose qu'on dispose d'un mécanisme **d'exclusion mutuelle** qui n'autorise qu'un seul processus à la fois à exécuter un code partagé, la phase d'utilisation des codes est appelée **section critique**.

Question 2 : *Discuter les solutions suivantes (avantages et inconvénients) et donner le pseudo code correspondant à chacune de ces solutions.*

1. *Chaque processus intègre la section critique suivante : {request(m), utilisation de m ressources, release(m)},*

2. *Chaque processus fait appel à deux sections critiques séparées : {request(m)} et {release(m)}.*

Question 2 : *Implémenter la solution retenue avec des moniteurs Posix.*

Exercice 2 : le paradigme du producteur consommateur

- Une tâche (appelée producteur) produit des messages (pour simplifier une chaîne de caractères « Bonjour1 », « Bonjour2 », …, « Bonjour100 ») dans un tampon de taille N (par exemple 10)

- Une autre tâche (appelée consommateur) consomme ces messages et les affiche à l'écran (l'ordre de production doit être respecté).

  Proposez une solution en Pseudo langage que vous implémenterez dans un second temps en C – POSIX. Cette solution doit garantir un fonctionnement correct quel que soit la vitesse d'exécution relative des tâches ; vous pourrez la tester en « ralentissant » le producteur et /ou le consommateur (par exemple en leur faisant faire des « sleep(x) » après chaque production ou chaque consommation avec une valeur x différente pour les deux tâches).