Semaine 5 – Questions sur la synchronisation (solutions)

1. Dans quel cas est-il intéressant de masquer une interruption?

   Réponse : Lors du fonctionnement de certaines parties du système d'exploitation, il peut être nécessaire d'interdire (masquer) les interruptions, soit parce que celles-ci perturberaient un compte serré du temps, soit parce que des structures de données sont en cours de modification (on réalise ainsi une sorte de verrou d'exclusion mutuelle dans un système mono-processeur).

2. Pourquoi le partage de données pose des problèmes dans un système multiprogrammé (utilisation de plusieurs fils d'exécution) en temps partagé? De quels types de problème parle-t-on? Nommez-en deux.

   Réponse : Un autre processus peut accéder aux données partagées avant qu'un processus n'ait fini de les utiliser (modifier). Problème de cohérence des données. Données lues trop tôt ou trop tard (conditions de concurrence).

3. Le système UNIX permet-il de contrôler les accès aux données partagées? Comment?

   Réponse : Oui, par exemple en utilisant des mécanismes de synchronisation comme les sémaphores, mutex, verrous, …

4. Qu'est-ce qu'une section critique? Que permet-elle de faire ou d'éviter?

   Réponse : En programmation concurrente, une section critique est une portion de code dans laquelle il doit être garanti qu'il n'y aura jamais plus d'un fil d'exécution (*thread*) simultanément. Il est nécessaire d'utiliser des sections critiques lorsqu'il y a accès à des ressources partagées par plusieurs fils d'exécution.

5. Qu'est-ce qu'une impasse (ou interblocage)? Donnez un exemple.

   Réponse : Un ensemble de processus est en interblocage si et seulement si **tout** processus de l'ensemble est en attente d'un évènement qui ne peut être réalisé que par un autre processus de l'ensemble.

   Par exemple, dans le problème des philosophes, si tous les philosophes prennent en même temps chacun une baguette, aucun d'entre eux ne pourrait prendre une seconde.

6. Qu'est-ce qu'une famine? Donnez un exemple.

   Réponse : La famine est une situation dans laquelle **un ou plusieurs** processus/threads n'ont jamais l'occasion d'accéder à une ressource partagée protégée par un sémaphore ou tout autre mécanisme assurant un contrôle d'accès à la ressource.

   Par exemple, dans le problème des philosophes, il se pourrait qu'un philosophe n'arrive jamais à obtenir les deux baguettes en même temps.

7. Dans **laquelle** de ces situations un processus peut-il se bloquer?

   a) lorsqu'il crée un sémaphore

      Justification : en cas de problème lors de la création d'un sémaphore, une valeur de retour sera envoyée pour des fins de gestion, donc pas de blocage.

   b) lorsqu'il tente de décrémenter un sémaphore dont le compteur à une valeur égale à zéro.

      Justification : ceci correspond à l'opération P() section SINON, chapitre 5, p. 22, qui cause un blocage.

   c) lorsqu'il incrémente le compteur d'un sémaphore dont la file d'attente n'est pas vide

      Justification : ceci correspond à l'opération V() section SINON, chapitre 5, p. 23, encore là pas de blocage.

   d) lorsqu'il tente de décrémenter un sémaphore dont le compteur à une valeur supérieure à zéro.

      Justification : ceci correspond à l'opération P() section SI, chapitre 5, p. 22, qui cause un blocage.

8. Dans le problème des lecteurs-rédacteurs :

   a) deux lecteurs peuvent travailler en même temps

      Justification : ceci correspond à une règle de fonctionnement, chapitre 5, p. 54. 3^{e} puce.

   b) deux rédacteurs peuvent travailler en même temps

      Justification : ceci correspond à une règle interdite, chapitre 5, p. 54, 2^{e} puce.

   c) un lecteur et un rédacteur peuvent travailler en même temps

      Justification : ceci correspond à une règle interdite, chapitre 5, p. 54, 2^{e} puce.

   d) un seul processus peut travailler à un moment donné

      Justification : effectivement si ce processus est un rédacteur, il sera le seul à travailler, chapitre 5, p. 54, 2^{e} puce. De plus, il se peut qu'à un moment donné qu'il n'y ait qu'un seul lecteur.

9. Dans le problème des producteurs-consommateurs :
   a) deux consommateurs peuvent travailler en même temps
      Justification : L'accès au tampon est exclusif, chapitre 5, p. 51 (texte dans l'encadré jaune)
   b) deux producteurs peuvent travailler en même temps
      Justification : L'accès au tampon est exclusif, chapitre 5, p. 51 (texte dans l'encadré jaune)
   c) un producteur et un consommateur peuvent travailler en même temps
      Justification : L'accès au tampon est exclusif, chapitre 5, p. 51 (texte dans l'encadré jaune)
   d) un seul processus peut travailler à un moment donné
      Justification : chapitre 5, p. 51

10. Dans le problème des philosophes :
   a) deux voisins peuvent penser en même temps
      Justification : Aucun problème car cette fonction se trouve à l'extérieur de la section critique, chapitre 5, p. 62
   b) deux voisins peuvent manger en même temps)
      Justification : impossible car il partage une ressource commune (fourchette), chapitre 5, p. 60
   c) deux voisins peuvent avoir faim en même temps
      Justification : la notion d'avoir faim représente la tentative (avec succès ou non) de décrémenter le sémaphore mutex, chapitre 5, p. 62
   d) tous les philosophes peuvent avoir faim en même temps
      Justification : la notion d'avoir faim représente la tentative (avec succès ou non) de décrémenter le sémaphore mutex, chapitre 5, p. 62

11. Soient trois processus concurrents P1, P2 et P3 qui partagent les variables n et out. Pour contrôler les accès aux variables partagées, votre coéquipier propose les codes suivants :

```
Semaphore mutex1 = 1;
Semaphore mutex2 = 1;

//Code du processus P1 :
P(mutex1);
P(mutex2);
out = out + 1;
n = n - 1;
V(mutex2);
V(mutex1);

//Code du processus P2 :
P(mutex2);
out = out - 1;
V(mutex2);

//Code du processus P3 :
P(mutex1);
n = n + 1;
V(mutex1);
```

Cette proposition est-elle correcte? Sinon, indiquez parmi les 4 conditions requises pour réaliser une exclusion mutuelle correcte, celle(s) qui n'est(sont) pas satisfaite(s)? Proposez une solution correcte.

Réponse : Non, car si P2 est dans la section critique et P1 a exécuté P(mutex1) alors P1 est bloqué et empêche P3 d'entrer dans la section critique. Conditions non vérifiées :
Un processus en dehors de sa section critique bloque un autre processus (qui ne devrait pas être le cas). Voir chapitre 5, p. 6 (condition 3).

Le code ci-dessous pour le processus P1 est une solution possible :

```
//Code du processus P1 :
P(mutex1);
n = n - 1;
V(mutex1);
P(mutex2);
out = out + 1;
V(mutex2);
```