5 - SEMAPHORES

1. RAPPELS

Une **ressource critique** est une ressource partagée entre plusieurs processus et dont l'accès doit être contrôlé pour préserver la cohérence de son contenu (variable, fichier, etc…). En particulier, ce contrôle peut impliquer un accès en **exclusion mutuelle**, c'est-à-dire que la ressource n'est jamais accédée par plus d'un processus à la fois.

Une **section critique** est une séquence d'instructions au cours de laquelle un processus accède à une ressource critique et qui doit être exécutée de manière indivisible.

L'**indivisibilité** signifie que deux sections critiques concernant une même ressource ne sont jamais exécutées simultanément mais toujours séquentiellement (dans un ordre arbitraire). Il y a ainsi exclusivité entre ces séquences d'instructions.

L'exclusion mutuelle n'est qu'un cas particulier de synchronisation. Une **synchronisation** est une opération influant sur l'avancement d'un ensemble de processus:

- établissement d'un ordre d'occurrence pour certaines opérations (envoyer / recevoir).
- respect d'une condition (rendez-vous, exclusion mutuelle, etc…).

L'attente active est la mobilisation d'un processeur pour l'exécution répétitive d'une primitive de synchronisation jusqu'à ce que la condition de synchronisation permette la continuation normale du processus.

Un sémaphore est un objet permettant de contrôler l'accès à une ressource en évitant l'attente active. Il est constitué d'un compteur et d'une file d'attente. La valeur du compteur dénombre, lorsqu'elle est positive, le nombre de ressources disponibles, lorsqu'elle est négative le nombre de processus en attente de ressource. La politique de gestion de la file d'attente est définie par le concepteur du système. A la création d'un sémaphore, la file est vide. Un sémaphore est manipulé à l'aide des opérations indivisibles suivantes :

`SEM *CS(cpt)` : Création d'un sémaphore dont le compteur est initialisé à "cpt".

`DS(sem)` : Destruction d'un sémaphore. Si la file n'est pas vide un traitement d'erreur doit être effectué.

`P(sem)` : Demande d'acquisition d'une ressource. Si aucune ressource n'est disponible, le processus est bloqué.

`V(sem)` : Libération d'une ressource. Si la file d'attente n'est pas vide, un processus est débloqué.

1.1.

Donnez le code des primitives P et V en utilisant les opérations suivantes :

- TACH **courant*() : désigne la tâche en cours d'exécution (la tâche ELUE).
- void *insérer*(TACH tache, FILE *file) : insère une tâche dans la file

- TACH **extraire*(FILE *file) : extrait une tâche de la file.

1.2.

Expliquez pourquoi ces primitives doivent être rendues indivisibles et comment on peut réaliser cette indivisibilité.

1.3.

Pourquoi n'utilise-t-on pas le masquage / démasquage des interruptions pour réaliser une section critique en mode utilisateur ?

1.4.

Un processus en train d'exécuter une section critique peut-il être interrompu par un autre processus ? Expliquez ce qui se passe alors.

On considère les trois processus suivants qui s'exécutent de manière concurrente, et le sémaphore Mutex initialisé à 1.

Processus A	***Processus B***	***Processus C***
`(a) P(Mutex);`	`(d) P(Mutex);`	`(g) P(Mutex);`
`(b) x = x + 1;`	`(e) x = x * 2;`	`(h) x = x - 4;`
`(c) V(Mutex);`	`(f) V(Mutex);`	`(i) V(Mutex);`

1.5.

On considère le scénario suivant : a d b g c e f h i

Donnez après chaque opération sur le sémaphore Mutex

- la valeur du compteur,
- le contenu de la file du sémaphore,
- l'état de chacun des processus (élu, prêt, bloqué).

Le scénario suivant est-il possible ? Justifiez : a d e b c f g h i

On considère les trois processus suivants qui s'exécutent de manière concurrente sur une machine. La variable **a** est une variable partagée, et on a les initialisations suivantes :

int a = 6; S1 = CS(1); S2 = CS(0);

Processus A	***Processus B***	***Processus C***
`P(S1);`	`a = a - 5;`	`V(S1);`
`a = a + 7;`		`P(S2);`
`V(S2);`		`a = a * 3;`

1.6.

Quelles sont les valeurs finales possibles de la variable a ? Donner pour chaque valeur la (les) suite(s) d'instruction qui amènent à cette valeur.

1.7.

- Ajoutez dans les programmes des processus les sémaphores nécessaires (et ceux-là seulement), pour obtenir dans toute exécution a = 34. Donnez les initialisations des sémaphores ajoutés.
- Même question pour a = 24.

1.8.

Quel sémaphore pourrait être supprimé sans modifier l'exécution de l'ensemble des processus ? Justifiez.

1.9.

On modifie la synchronisation de la manière suivante :

Processus A	***Processus B***	***Processus C***
`P(S1);`	`P(S2);`	`P(S2);`
`P(S2);`	`a = a - 5;`	`P(S1);`
`a = a + 7;`	`V(S2);`	`a = a * 3;`
`V(S1);`	`V(S1);`	`V(S2);`
`V(S2);`		

S1 et S2 sont **tous les deux initialisés à 1**. Quelles sont les conséquences de cette modification ? Justifiez.

2. SYNCHRONISATION ET INTERBLOCAGE

Soient N processus utilisateurs U_1 à U_N partageant un ensemble de X serveurs et Y ressources (X > 0, Y > 0, N > X + Y). Ces processus sont synchronisés au moyen de 2 sémaphores S et R, représentant la disponibilité des serveurs et des ressources, selon le programme suivant.

```
Boucle
    Prologue:  P(S);  P(R);  V(S);
    Utilisation de ressource;
    Epilogue:  P(S);  V(R);  V(S);
Fin boucle
```

Le sémaphore S est initialisé à X et le sémaphore R est initialisé à Y.

2.1.

Quel est le nombre maximum n de processus pouvant se trouver simultanément en train d'utiliser une ressource ?

2.2.

Montrez, pour N = 2 et X = Y = 1 que l'exécution de ce système peut conduire à un interblocage. Pour X > 0, Y > 0 quelconques, quelle est la valeur minimum de N qui peut amener à un interblocage ?

2.3.

L'interblocage peut être évité en introduisant un nouveau sémaphore L, pour limiter le nombre de processus autorisés à entrer dans le prologue. Introduisez les opérations P(L) et V(L) nécessaires dans le prologue et l'épilogue. Précisez l'initialisation du sémaphore L.

3. GESTION D'UN SAS

On souhaite décrire le comportement de clients qui entrent dans une banque par l'intermédiaire d'un sas. Ce sas a deux portes qui ne doivent jamais être ouvertes simultanément.

Chaque client qui veut entrer exécute successivement les deux procédures suivantes :

ENTRER_SAS : bloque les clients dans le sas tant que la première porte est ouverte.

ENTRER_BANQUE : permet au client d'entrer lorsque la deuxième porte est ouverte.

NB : on ne s'intéresse pas à ce qui se passe à la sortie de la banque.

On suppose dans un premier temps que le sas d'entrée a une capacité de N clients. Les clients ne peuvent entrer dans la banque que lorsque le sas est plein.

3.1.

En 3 lignes maximum, que pensez-vous de cette gestion du sas ?

3.2.

Donnez la liste et la signification des sémaphores que vous utilisez pour programmer les procédures ENTRER_SAS et ENTRER_BANQUE, ainsi que la valeur d'initialisation de leur compteur.

3.3.

Programmez les procédures ENTRER_SAS et ENTRER_BANQUE.

Pour améliorer le fonctionnement du système, on suppose maintenant qu'un client qui veut entrer dans le sas se signale en levant le bras. Lorsqu'il arrive dans le sas (et que celui-ci n'est pas plein), il regarde derrière lui si d'autres clients veulent entrer. Si oui, il laisse la première porte ouverte, sinon il ouvre la deuxième.

3.4.

Reprogrammez les procédures ENTRER_SAS et ENTRER_BANQUE.