Semaine 8 – Questions sur la synchronisation (partie 2)

1. Expliquez ce qui peut arriver si la file d'attente d'un sémaphore est gérée selon la politique LIFO (Last In First Out).

   Risque de famine, car si un nouveau processus entre dans la file d'attente alors les anciens dans la file risque de ne pas pouvoir accéder à leur section critique.

2. Expliquez ce qu'est un moniteur et quel est le mécanisme utilisé par les moniteurs afin d'assurer l'exclusion mutuelle.

   Une structure de données composée d'attributs (variables partagées) et de méthodes (sections critiques) qui partagent en exclusion mutuelle ces attributs. C'est un mécanisme de synchronisation. Chaque moniteur a une file d'attente selon la politique FIFO.

   Pour assurer l'exclusion mutuelle, il suffit de regrouper toutes les sections critiques et toutes les variables partagées d'un problème dans un même moniteur.

3. Donnez un avantage de l'utilisation des moniteurs par rapport aux sémaphores pour la synchronisation de processus.

   Le moniteur regroupe, en une seule structure, toutes les sections critiques d'un problème donné. Le contrôle d'accès aux sections critiques est géré par le moniteur. Il facilite ainsi la compréhension et l'implémentation du code de synchronisation.

   Le programmeur n'a pas à se soucier de l'exclusion mutuelle. Elle est assurée par le moniteur.

4. Les moniteurs à eux seuls permettent d'implémenter le problème des producteurs et des consommateurs.
   a) Vrai
   b) Faux

Justifiez :

Le consommateur va rentrer dans une attente active infinie dans le moniteur, s'il accède en premier au moniteur (car le tampon est vide). Le producteur va se retrouver en attente infinie du moniteur. (ch. 5-suite, p. 6)

Pour pallier cette limitation, il suffit d'offrir la possibilité de se mettre en attente passive dans le moniteur tout en libérant l'accès au moniteur -> Variables de condition.

5. Quel énoncé est vrai ?
    a) Signal-and-wait est la seule politique de moniteur qui implique une file d'attente pour la variable de condition
    b) Signal-and-wait offre une file d'attente prioritaire pour les processus qui sont débloqués par le signalement d'une variable de condition
    c) Signal-and-wait offre une file d'attente prioritaire pour les processus interrompus par un processus débloqué par le signalement d'une variable de condition (ch. 5-suite, p.11, celle qui contient P5)
    d) Aucune de ces réponses

6. La sémantique Signal-and-wait utilisée dans les moniteurs implique une file d'attente prioritaire dans laquelle on place une tâche dès que celle-ci tombe en attente d'une variable de condition.
    a) Vrai
    b) Faux

    Justifiez :

    Ceci est vrai uniquement pour les processus interrompus par un processus débloqué par le signalement d'une variable de condition (ch. 5-suite, p.11, celle qui contient P5) et non pour les processus qui sont en attente d'une variable de condition (ch. 5-suite, p. 11, celle qui contient P1 et P2).

7. Il est possible d'implémenter un sémaphore avec un moniteur.
    a) Vrai (ch. 5-suite, p. 19)
    b) Faux

Si oui, comment?

```
Moniteur Semaphore
{
    const int N = 100;
    boolc car_file_vide;
    boolc car_file_pleine;
    int compteur = 0;
    void P()
    {
        if (!compteur)
        {
            wait(car_file_vide);
        }
        compteur--;
        if (compteur == N - 1)
        {
            signal(car_file_pleine);
        }
    }
    void V()
    {
        if (compteur == N)
        {
            wait(car_file_pleine);
        }
        compteur++;
        if (compteur == 1)
        {
            signal(car_file_vide);
        }
    }
}
```

8. On veut adapter le moniteur ProducteurConsommateur, vu en classe, au cas de deux producteurs P1 et P2 et d'un consommateur. Les producteurs doivent produire en alternance dans le tampon (une production de P1 suivie d'une production de P2, etc.). Chaque producteur Pi (i=1,2) est supposé appeler sa propre fonction de production produire_Pi.

   a) Complétez le code du moniteur ci-dessous afin de permettre aux producteurs P1 et P2 de produire en alternance dans le tampon.
   Attention : Comme mécanisme de synchronisation, vous devez vous limiter aux variables de condition pour les deux questions a) et b).

```
Moniteur ProducteurConsommateur1
{
    const N = 100;
    int tampon[N];
    boolc nplein, nvide; // variables de condition
    int compteur = 0, ic = 0, ip = 0, tour = 1; // pour chacun son tour
    boolc wtour;
    void produire_P1(int objet)
    {
        if (tour != 1)
            wait(wtour);
        if (compteur == N)
            wait(nplein);
        tampon[ip] = objet;
        ip = (ip + 1) % N;
        compteur++;
        if (compteur == 1)
            signal(nvide);
        tour = 2;
        signal(wtour);
    }
    void produire_P2(int objet)
    {
        if (tour != 2)
            wait(wtour);
        if (compteur == N)
            wait(nplein);
        tampon[ip] = objet;
        ip = (ip + 1) % N;
        compteur++;
        if (compteur == 1)
            signal(nvide);
        tour = 1;
        signal(wtour);
```

```
    }
    int consommer()
    {
        int objet;
        if (compteur == 0)
            wait(nvide);
        objet = tampon[ic];
        ic = (ic + 1) % N;
        compteur--;
        if (compteur == N - 1)
            signal(nplein);
        return objet;
    }
}
```

b) Supposez que la taille du tampon est égale à 1. Pour le cas d'un producteur et d'un consommateur, est-il possible de simplifier le code précédent (éliminer des variables de condition ou autres) ? Si oui, complétez le moniteur suivant. Sinon, justifiez votre réponse.

Oui, cela revient à forcer l'ordre suivant : une production, une consommation, une production, une consommation, etc.

```
Moniteur ProducteurConsommateur
{
    int tampon, tour = 1;
    boolc wtour;
    void produire(int objet)
    {
        if (tour != 1)
            wait(wtour);
        tampon = objet;
        tour = 2;
        signal(wtour);
    }
    int consommer()
    {
        int objet;
        if (tour != 2)
            wait(wtour);
        objet = tampon;
        tour = 1;
```

```
        signal(wtour);
        return objet;
    }
}
```