TD 2
Concurrence et synchronisation – INFO3 S5
Moniteurs

Objectifs du TD : Reprendre les éléments théoriques vus en cours sur les moniteurs, et les appliquer à la conception de solutions aux problèmes posés par la concurrence de processus s'exécutant en parallèle

1 Un Sémaphore de comptage défini par moniteur

1.1 Un sémaphore de comptage est défini par les éléments suivants :

- Un compteur entier
- Une liste d'attente fifo
- 2 opérations atomiques P() et V() (en exclusion mutuelle)

Concevez un moniteur permettant de reproduire le comportement d'un sémaphore de comptage Pour cela vous suivrez la procédure donnée en cours :

(1) *variables d'états* : les ressources à protéger,
(2) *points d'entrée* : nom des fonctions du moniteur (sections critiques),
(3) *Conditions* : liste d'attente de processus supprotant les opérations wait(), signal(),
(4) *Assertion logiques* : définir les expressions logiques contrôlant les opérations wait() et signal() sur chaque condition,
(5) *Coder* le moniteur et ses points d'entrée

NB : les *points d'entrée* sont par définition des fonctions exécutées en exclusion mutuelle au sein du moniteur. Le plus souvent les langages de programmation offrent des pseudo-moniteurs où l'exclusion mutuelle (EM) n'est pas implicite et doit être gérée manuellement avec un sémaphore binaire (mutex).

1.2 Donnez le *pseudo code* implémentant un sémaphore de comptage à l'aide de moniteur. Utilisez-le pour résoudre le problème de concurrence (EM – exercice 1, TD 1) entre 2 processus P1 et P2, partageant une variable globale commune RC (ressource critique) :

```
Processus : P1
    Global : RC
    Repeter
        a := RC
        RC := a + 1
    Jusqua Faux
    .....
```

```
Processus : P2
    Global : RC
    Repeter
        b := RC
        RC := b + 2
    Jusqua Faux
    .....
```

NB : pour les conditions, on utilisera les notations en pseudo-code : `Condition : nomCond, wait(), signal()`

2 L'alternance

Soit les 2 processus P1 et P2 suivants :

```
Processus : P1
    Repeter
        Ping()
    Jusqua Faux
    .....
```

```
Processus : P2
    Repeter
        Pong()
    Jusqua Faux
    .....
```

On souhaite que les 2 opérations `Ping()` et `Pong()` respectent les contraintes suivantes :

- Exclusion mutuelle
- On débute par `Ping()`
- `Ping()`et `Pong()` doivent alterner strictement (`Ping-Pong-Ping-Pong-Ping-Pong-…..`)

2.1 Concevez un moniteur permettant de reproduire ce comportement. Pour cela vous suivrez la même procédure que précédemment : (1) *variables d'états*, (2) *points d'entrée*, (3) *Conditions*, (4) *Assertion logiques*, (5) *Coder*.

2.2 Donnez le *pseudo code* implémentant ce moniteur et utilisez-le pour résoudre le problème de synchronisation entre P1 et P2.

3 Le problème des Producteurs Consommateurs

Soit les 2 processus Producteur et Consommateur suivants :

```
Global :
        Tampon : t
```

```
Processus : Producteur
    Repeter
        Deposer (t, p)
    Jusqua Faux
```

```
Processus : Consommateur
    Repeter
        p := Retirer (t)
    Jusqua Faux
```

```
Processus : Principal
    InitTampon (t, TailleMax);
    ParDebut
        Producteur;
        Consommateur;
    ParFin
```

```
Type Tampon :
InitTampon (Tampon: t , Entier: tailleMax)
NbProduits (Tampon: t) → Entier
TailleMax (Tampon: t) → Entier
EstVide (Tampon: t) → Booleen
EstPlein (Tampon: t) → Booleen
Deposer (Tampon: t, Produit: p)
Retirer (Tampon: t) → Produit
```

Le `Tampon t` est une ressource commune aux 2 processus, qui est munie d'un ensemble d'opérations de manipulation `Deposer`(),`Retirer`()….

On souhaite que les opérations `Deposer`() et/ou `Retirer`()r sur le tampon respectent les contraintes suivantes :

- C1 : Exclusion mutuelle des opérations `Deposer`() et/ou `Retirer`()
- C2 :Attente des consommateurs si le tampon est vide (`EstVide`())
- C3 :Attente des producteurs si le tampon est plein (`EstPlein`())

3.1 Concevez un moniteur permettant de reproduire la contrainte C1. Pour cela vous suivrez la même procédure que précédemment : (1) *variables d'états*, (2) *points d'entrée*, (3) *Conditions*, (4) *Assertion logiques*, (5) *Coder*.

3.2 En dériver le pseudo-code de votre solution par moniteur. Vérifiez que les assertions logiques contrôlant les opération wait() et signal() sont correctes, que les configurations indésirables sont bien évitées et qu'il n'y a pas d'interblocage.

3.3 Répéter les questions 3.1 et 3.2 pour résoudre les 2 contraintes C1 et C2, puis les 3 :C1 et C2 et C3

4 Usine d'assemblage d'aéroplanes

Soit les 8 programmes/processus suivants composant une usine d'assemblage d'aéroplanes :

Semaphore: ***A compléter***

```
Processus : Carlingue
    repeter
        Une carlingue est
achevée
```

```
Processus : Carlingue1Ailes2
    repeter
        Un assemblage 1 carlingue avec 2 ailes est
acheve
```

```
Processus : Aile
    repeter
        Une aile est achevée
```

```
Processus : Carlingue1Ailes2Roues3
    Repeter
        Un assemblage 1 carlingue et 2 ailes avec 3
roue est acheve
```

```
Processus : Moteur
    repeter
        Une aile est achevée
```

```
Processus : Aeroplane
    Repeter
        Un aéroplaneest acheve
```

```
Processus : Roue
    repeter
        Une aile est achevée
```

```
Processus : Principal
    ParDebut
        Carlingue1Ailes2
        Carlingue1Ailes2Roues3
        Aeroplane
        Carlingue
        Aile
        Moteur
        Roue
    ParFin
```

Les 7 processus de l'usine d'assemblage fonctionnent de manière autonome et en parallèle.

Cependant différentes contraintes de synchronisation doivent être respectées

- La chaine de montage `Carlingue1Ailes2` ne peut fonctionner que si au moins 2 ailes et une carlingue ont été produites
- La chaine de montage `Carlingue1Ailes2Roues3` peut fonctionner que si au moins 3 roues et un assemblage `Carlingue1Ailes2` ont été produits
- La chaine de montage `Aeroplane` peut fonctionner que si au moins 2 moteurs et un assemblage `Carlingue1Ailes2Roues3` ont été produits

4.1 Proposez une solution utilisant des tampons Producteurs Consommateurs afin de gérer les contraintes d'enchaînement entre les 7 processus. Pour cela il suffit de remarquer que `Carlingue` et `Aile` sont des processus producteurs pour le processus consommateur `Carlingue1Ailes2` … En dériver le pseudo code.

4.2 Proposez une solution à base de moniteur afin de gérer les contraintes d'enchaînement entre les 7 processus. Pour cela il suffit de remarquer que `Carlingue1Ailes2` devra être endormi (`wait`() sur condition) tant que au moins 1 carlingue produite par `Carlingue` et 2 ailes produites par `Aile` ne sont pas disponible … En dériver le pseudo code du moniteur et vérifier toutes les assertions logiques de contrôle des conditions du moniteur.