

## 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 L'alternance

Soit les 2 processus P1 et P2 suivants :



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-.....)

1.1 Concevez un moniteur permettant de reproduire ce comportement d'alternance Pour cela vous suivrez la procédure donnée en cours :

- (1) *variables d'états* : les ressources à protéger (ressources critiques et état du moniteur),
- (2) *points d'entrée* : nom des fonctions du moniteur (sections critiques),
- (3) *Conditions* : liste d'attente de processus supportant 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).

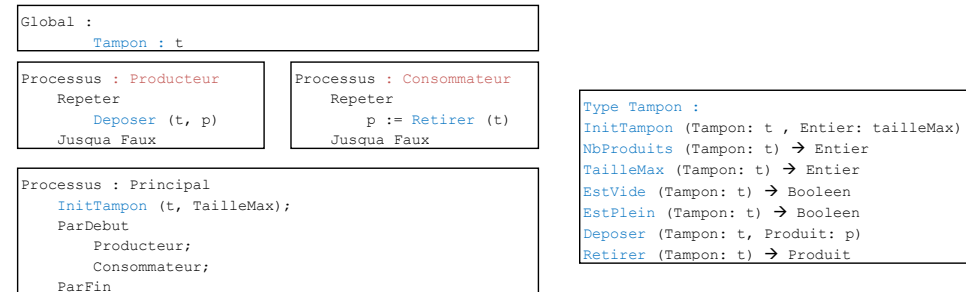
1.2 Donnez le *pseudo code* implémentant ce moniteur et utilisez-le pour résoudre le problème de synchronisation entre P1 et P2.

1.3 Donner le chronogramme de principe donnant l'évolution des processus, du moniteur et des conditions, dans un tableau dont les colonnes seront par exemple : P1, P2, M.P, M.F, C1, C2, commentaire

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

### 2 Le problème des Producteurs Consommateurs

Soit les 2 processus Producteur et Consommateur suivants :



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())

2.1 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*.

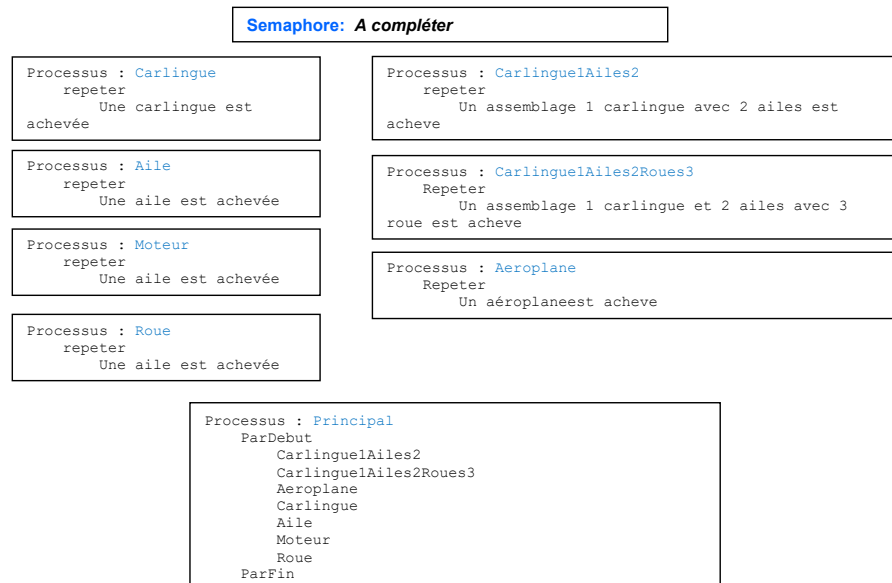
2.2 En dériver le pseudo-code de votre solution par moniteur. Vérifiez que les assertions logiques contrôlant les opération wait() et signal() sont correctes, que les configurations indésirables sont bien évitées et qu'il n'y a pas d'interblocage.

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

2.4 Donner le chronogramme de principe donnant l'évolution des processus, du moniteur et des conditions, dans un tableau dont les colonnes seront par exemple : C1, C2, P1, P2, M.P, M.F, CV, CP, T, commentaire

### 3 Usine d'assemblage d'aéroplanes

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



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

Cependant différentes contraintes de synchronisation doivent être respectées

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

- 3.1 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.
- 3.2 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.
- 3.3 Donner le chronogramme de principe donnant l'évolution des processus, du moniteur et des conditions, dans un tableau dont les colonnes seront par exemple : C, A, M, R, C2A, C2A3R, AERO, M.P, M.F, CC, CA, CM, ..., NC, NA, NM, ..., commentaire

### 4 Un Sémaphore de comptage défini par moniteur

4.1 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 même procédure que précédemment : (1) *variables d'états*, (2) *points d'entrée*, (3) *Conditions*, (4) *Assertion logiques*, (5) *Coder*.

4.2 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) :

