Université des Sciences et de la Technologie
Houari Boumediene
USTHB

Outils de synchronisation évolués

Dr BOUYAKOUB F. M
bouyakoub.f.m@gmail.com

Novembre 2012

Les moniteurs

bouyakoub.f.m@gmail.com

Définition de *Hoare* (1974)

- Un moniteur est **un module** exportant des procédures (et éventuellement des constantes).
- Contrainte d'exclusion mutuelle pour l'exécution des procédures du moniteur: au plus une procédure en cours d'exécution.
- Mécanisme de synchronisation interne.
- Un moniteur est **passif**: ce sont les activités qui invoquent ses procédures.

Définition 2:

- Un moniteur est un outil de synchronisation entre processus. Il est composé de:
  - Un ensemble de variables de synchronisation;
  - Procédures pour la manipulation de ces variables:
    - Procédures internes au moniteur;
    - Procédures externes accessibles aux processus.
- Les ressources sur lesquelles les processus se synchronisent sont manipulées par les procédures externes (points d'entrée) du moniteur.

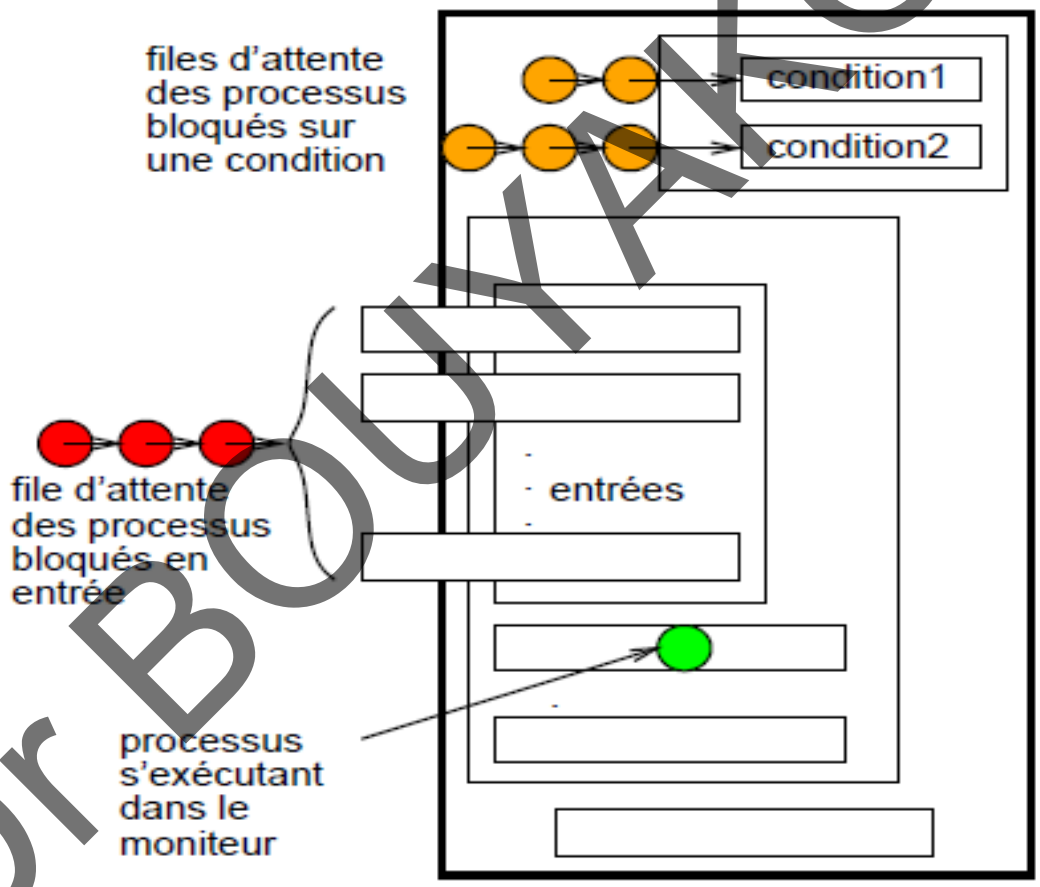
Remarques

- Un seul processus est actif à un instant donné dans le moniteur.
- Chaque entrée (opération) est accessible en exclusion mutuelle.
- Pas de variable exportée, seules les procédures du moniteur sont susceptibles de manipuler les variables du moniteur.
- Les sections critiques sont transformées en opérations (procédures) d'un moniteur.
- La gestion de la section critique est à la charge du moniteur et non pas de l'utilisateur. Le moniteur est implanté tout entier comme une section critique.
- Si le moniteur appelé par un processus pour manipuler une donnée partagée, est occupé, le processus est placé dans la file d'attente du moniteur. Dès qu'il est libéré, l'un des processus de la file est choisi et la procédure invoquée est exécutée.

Procédures internes au moniteur

- La synchronisation des processus s'exprime à l'aide de conditions;
- Le blocage et le réveil des processus se fait à l'aide de trois primitives: ***wait***, ***signal*** et ***empty***:
  - ***Wait:*** cette primitive bloque le processus appelant en dehors du moniteur et derrière une condition c (***c.wait***).
  - ***Signal:*** ***c.signal*** réveille l'un des processus bloqués derrière la condition c (FIFO) si la file est non vide, sinon le signal est perdu (pas de mémorisation).
  - ***Empty:*** ***c.empty*** retourne "vrai" s'il y a au moins un processus bloqué derrière la condition c.

Représentation du moniteur



Méthodologie d'implémentation du moniteur

- Déterminer l'interface du moniteur: **procédures externes**
- Définir les prédicats d'acceptation de chaque opérations: **conditions d'exécution des procédures externes**
- Déduire les variables d'état qui permettent d'écrire ces prédicats d'acceptation: **variables associées aux conditions**
- Programmer les opérations

Exemple 1: Gestion d'une ressource à N points d'accès

Allocation: moniteur Variable N:entier; dispo: condition	**Initialisation** N=K;
Entry procedure Demander **Début** **Si** (N==0) alors dispo.wait; **fsi** N--; **Fin**	**Entry procedure Libérer** **Début** N++; dispo.signal; **Fin**

Exemple 2: producteur/consommateur (1/2)

Allocation: moniteur
Constante N //nombre de cases du tampon
Variable
Tampon : tableau;
prod, cons: entier // indice du tableau;
cpt: entier;
non-vide, non-plein : condition;
	Initialisation
	cpt =0;
	prod=0;
	cons=0;

Entry procedure Déposer (x) **Début** **Si** (cpt ==N) alors non-plein. wait **fsi** Tampon[prod] = x; Prod = (prod + 1) modulo N; cpt++; non-vide.signal; **Fin**	**Entry procedure Prélever (x)** **Début** **Si** (cpt ==0) alors non-vide.wait **fsi** X= tampon[cons]; cons = (cons +1) modulo N; cpt--; non-plein.signal; **Fin**

Exemple 2: producteur/consommateur (2/2)

- Inconvénient de la solution proposée: Le tampon est utilisé en exclusion mutuelle (dans le moniteur)
- Solution: faire sortir le tampon du moniteur et associer aux processus producteur/consommateur deux procédures "classiques": Déposer(article) et prélever (article).

Processus Producteur

```
{.....
Demande-Dépôt()
Déposer(article);
Libérer-Dépôt()
.....}
```

Processus Consommateur

```
{.....
Demande-Prélèvement()
Prélever(article);
Libérer-Prélèvement()
.....}
```

Solution 2: producteur/consommateur

Entry procedure Demande-Dépôt **Début** **Si** (cpt==N) alors Non-plein.wait **fsi** **Fin**	**Entry procedure Demande-Prélèvement** **Début** **Si** (cpt ==0) alors non-vide.wait **fsi** **Fin**
Entry procedure Libérer-Dépôt **Début** cpt++; non-vide.signal; **Fin**	**Entry procedure Libérer-Prélèvement** **Début** cpt--; non-plein.signal; **Fin**

A faire !

Généraliser la solution précédente pour plusieurs producteurs et plusieurs consommateurs

Variantes des moniteurs

- Conditions de *Kessels*
- Condition avec priorité

La variante de *Kessels*

Condition de *Kessels*

- Cette variante du moniteur consiste en la redéfinition de la primitive *wait* en lui associant une condition booléenne: ***wait(c)***
- Un processus se bloque sur *wait* si la condition n'est pas vérifiée.
- Quand un processus sort du moniteur ou lors de son blocage par *wait* toutes les conditions figurant dans les primitives *wait* sont réévaluées automatiquement ➔ **suppression de *signal***

Cas du producteur/consommateur avec condition de *Kessels*

Entry procedure Demande-Dépôt **Début** wait(cpt<N); **Fin**	**Entry procedure Demande-Prélèvement** **Début** wait(cpt>0); **Fin**
Entry procedure Libérer-Dépôt **Début** cpt++; **Fin**	**Entry procedure Libérer-Prélèvement** **Début** cpt--; **Fin**

Moniteur avec priorité

Condition avec priorité

- Introduction d'un ordre de réveil ➔ priorité.
- On associe un entier avec *wait*: ***c.wait(i)***.
- i indique la priorité du processus.
- Quand signal est exécutée, le processus bloqué sur la condition de synchronisation correspondante et ayant la valeur minimal de priorité est réveillé.

Exemple

- Gestion d'une ressource dont l'allocation est faite en priorité à celui qui a le moins d'instances de la ressource

Entry procedure Demande(K) **Début** **Si** (Disp<K) alors Ressource.wait(k) **fsi** **Fin**	**Entry procedure Libérer(m)** **Début** Disp=Disp+m; Ressource.signal; **Fin**

Les régions critiques

Questions ?