Université Mohammed V- Rabat
École Mohammadia d'Ingénieurs
Département Génie Informatique

Filière Génie Informatique et Digitalisation

SYSTÈMES D'EXPLOITATION 2

Mr N. EL Faddouli

elfaddouli@emi.ac.ma, nfaddouli@gmail.com

2024-2025

1

Plan du cours

- Introduction et Rappels
- Gestion des processus: Synchronisation avec attente active
  - Définitions et rappels
  - Algorithmes avec attente active
- Gestion des processus: Synchronisation sans attente active
  - Les sémaphores
  - Les moniteurs
  - L'interblocage (Dead-lock)
- Gestion de la mémoire
  - Définitions et rappels
  - La mémoire virtuelle
- Les entrées/Sorties

2

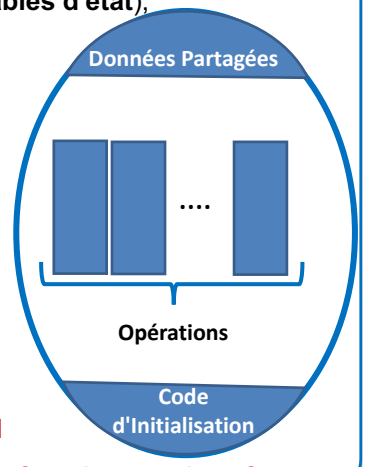
Les moniteurs

Les Moniteurs (Monitor)- Hoare (1974) / Hansen (1975)

- Les sémaphores fournissent un mécanisme pratique et efficace pour la synchronisation de processus.
- Le programmeur développe le code d'entrée et de sortie d'une section critique en utilisant les opérations P et V sur des sémaphores bien initialisés.
- Leur **utilisation incorrecte** peut conduire à des erreurs de synchronisation:
  - difficiles à détecter
  - surviennent si des **séquences d'exécution particulières** ont lieu.
- Comment décharger le programmeur du développement du code d'entrée et sortie d'une section critique ? ⇒⇒ **Les moniteurs**

Les Moniteurs (Monitor)- Hoare (1974) / Hansen (1975)

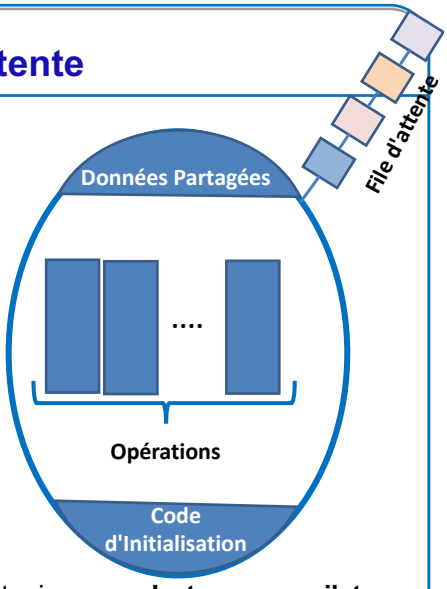
- Un moniteur est un module de programme constitué de:
  - un ensemble de variables partagées (**variables d'état**),
  - un ensemble de **procédures** (*méthodes*) permettant d'accéder à ces variables.
  - un corps d'initialisation des variables.
- Seules certaines procédures, appelées entrées (**entry**) du moniteur sont **visibles**: seules ces procédures peuvent être appelées de l'extérieur.
- **Les procédures du moniteur sont exécutées en exclusion mutuelle: Un seul processus peut être actif dans le moniteur à un instant donné.**



Les Moniteurs: La file d'attente

- Quand un processus **P** appelle une procédure d'un moniteur qui n'est pas disponible (*le moniteur est utilisé par un autre processus*),

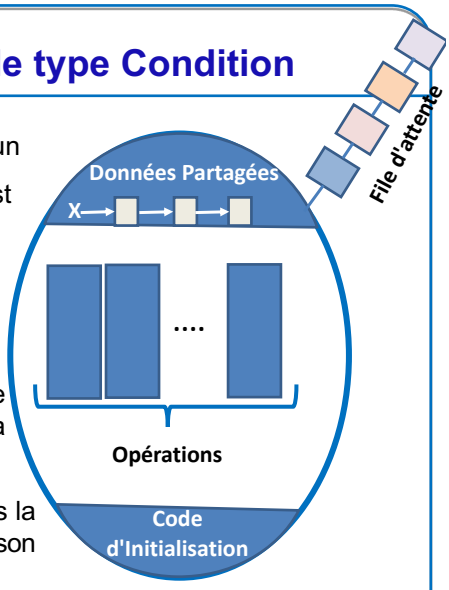
→ le **PCB** du processus P est placé dans **une file d'attente associée au moniteur**. Dès que ce dernier est libéré, un processus est choisi de la file et la procédure appelée est exécutée.



- Un moniteur est une primitive de haut niveau : **c'est au compilateur d'assurer l'exclusion mutuelle pour l'accès au procédures du moniteur.**

Les Moniteurs: Variable de type Condition

- Une variable X de type ***condition*** a un rôle particulier dans un moniteur: Elle est représentée par une ***file d'attente*** sur laquelle on peut agir à l'aide de deux primitives spéciales:



wait : suspend le processus appelant et le met en attente dans la file de la variable (***X.wait()***)

signal: permet à un processus bloqué dans la file de la variable de reprendre son exécution (***X.signal()***)

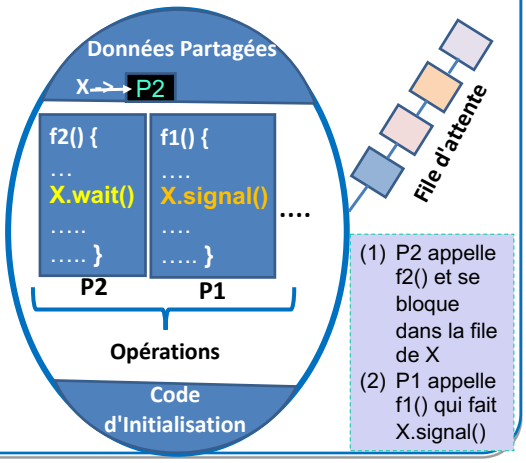
Si aucun processus n'est suspendu, la primitive **signal** n'a pas d'effet (*l'état de la variable condition ne change pas*).

Les Moniteurs: Variable de type Condition

- Il y a plusieurs comportements pour l'opération **Signal**:
  - **Signal & Continue (SC):** Le processus signaleur (appelant) garde l'exclusion mutuelle et le processus signalé (réveillé) attendra à ce que le signaleur quitte le moniteur.

P2 ne pourra reprendre son exécution que lorsque P1 aura quitté le moniteur (*fin de l'appel de f1()*)

Moniteur de type Mesa
En Java par exemple



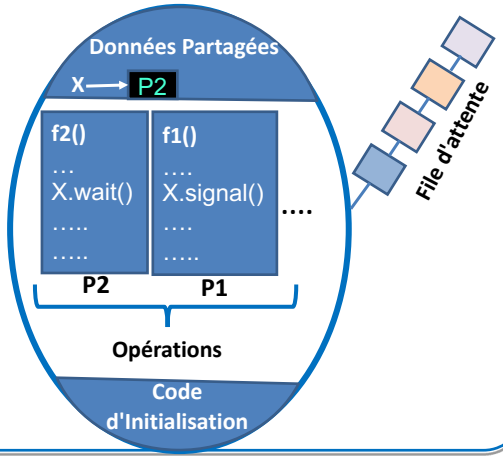
Les Moniteurs: Variable de type Condition

- **Signal & Wait (SW):** Le processus signaleur est bloqué en attendant l'exclusion mutuelle et le processus signalé peut acquérir le moniteur et poursuivre son exécution.

P1 sera bloqué et P2 pourra poursuivre son exécution de f2().

P1 pourra poursuivre son exécution de f1() quand il aura le tour pour entrer dans le moniteur.

Moniteur de type Hoare en C++ par exemple

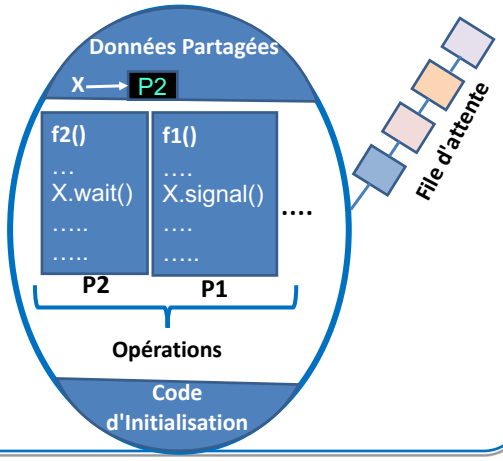


Les Moniteurs: Variable de type Condition

- **Signal & Urgent Wait (SU):** Comme SW mais le processus signaleur a la garantie qu'il irait juste après le processus réveillé.

P1 sera bloqué et P2 pourra poursuivre son exécution de f2().

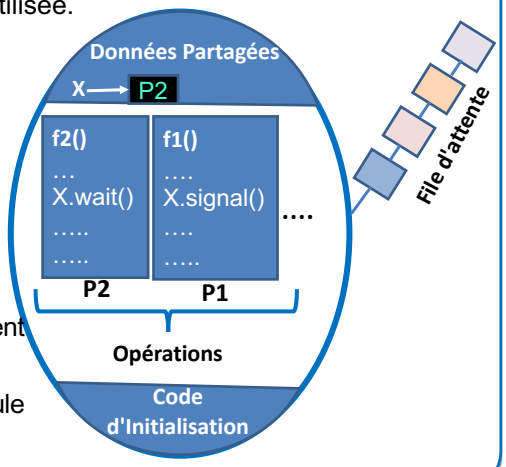
P1 pourra poursuivre son exécution de f1() quand P2 aura quitté le moniteur.



Les Moniteurs: Variable de type Condition

- **Signal & Exit (SX):** le signaleur quitte la méthode directement après le signal et le processus signalé peut démarrer directement. Cette philosophie n'est pas souvent utilisée.

P1 quitte immédiatement le moniteur après l'appel de Signal() et P2 poursuit son exécution de f2().



- Les politiques disponibles dépendent du langage de programmation.
- En **Java**, il n'y a qu'une seule politique disponible, celle de **SC**.

Les Moniteurs: Acquisition d'une ressource unique

Exemple 1: acquisition d'une ressource unique

```
Allocateur: Moniteur
   Var      Occupé: Booléen; // initialisée à Faux
            Libre: Condition;
   PROCEDURE Acquisition()
    Begin
      while (Occupé) Then  Libre.wait();
      Occupé := True;
    End;
  PROCEDURE Libération()
    Begin
      Occupé=Faux;
      Libre.signal();
    End;
```

Processus

```
Appel de Acquisition()
dans le moniteur
Allocateur
allocateur.Acquisition();
```

↓

S.C

↓

```
Appel de Liberation()
dans le moniteur
Allocateur
allocateur.Libération();
```

```
Begin // corps d'initialisation des variables
      Occupé :=Faux;
End;
```

Les Moniteurs: Acquisition d'une ressource unique

Utilisation du moniteur par un processus:

allocateur.Acquisition(); *// acquérir la ressource via le moniteur*

Utiliser_Ressource_Dans_SC(); *// peut faire partie du moniteur.*

allocateur.Libération(); *// Libérer la ressource*

Les Moniteurs: Producteur-Consommateur

Exemple 2: Producteur-Consommateur

```
Prod_Cons: Moniteur
Var     Buffer: Array [0..N-1] of message;
        read, write: Condition;
        compteur , indiceP, indiceC : Integer ;

PROCEDURE   Ecrire(A: message)
Begin
  if compteur = N  Then  write.wait;
  Buffer[indiceP] :=A;
  indiceP :=(indiceP+1) Mod N;
  compteur := compteur + 1;
  if compteur = 1  Then  read.signal;
End

PROCEDURE  Lire( X:message)
Begin
  if compteur = 0  Then  read.wait;
  X := Buffer[indiceC];
  indiceC := (indiceC  + 1)  Mod  N
  compteur := compteur -1;
  if compteur = N-1 Then  write.signal;
End

Begin
    compteur:=0;  indiceP := 0; indiceC := 0;
End // fin du moniteur
```

Les Moniteurs: Avantages

Avantages des moniteurs

• Les sections critiques sont transformées en fonctions dans un moniteur

➔ regrouper les sections critiques du programme

• La gestion des sections critiques n'est plus à la charge du programmeur. Elle est réalisée par l'implantation du moniteur (*supporté par le compilateur*).

• Le moniteur tout entier est implémenté comme une section critique dont l'accès est assuré par le compilateur.

Les Moniteurs: Exemple en Java

Exemple: Producteur-Consommateur en Java

```
public class ProdCons
{ private Object buffer[]; /* Mémoire partagée */
  private int N;           /* Capacité de la zone */
  private int count, in, out; /* nb d'éléments, indices */
  private Waitqueue vide, plein; /* files d'attente */
public ProdCons(int tailleZone)
{ N = tailleZone; /* création d'un tampon de taille tailleZone*/
  buffer = new Object[N];
  count = 0; in = 0; out = 0;
vide= new Waitqueue();
plein= new Waitqueue();
}
```

Ref: **http://www.montefiore.ulg.ac.be/~pw/cours/psfiles/struct-cours7.pdf**

Les Moniteurs: Exemple en Java

```
public synchronized  void  append(Object data)
{  if (count == N) plein.qWait();
buffer[in] = data;
in = (in + 1) % N; count++;
vide.qSignal();
}
public synchronized  Object  take()
{
Object  data;
    if (count == 0) vide.qWait();
data = buffer[out];
out = (out + 1) % N;
count--;
plein.qSignal();
return data;
}
}
```