

# Les Moniteurs

- [Définition](#)
  - [Un exemple simple de moniteur](#)
  - [Les instructions spéciales des moniteurs](#)
  - [Rendez-vous entre  \$N\$  processus](#)
  - [À l'intérieur des moniteurs](#)
  - [Problème des producteurs-consommateurs](#)
  - [Problème des lecteurs-rédacteurs](#)
- 

## Définition

Un moniteur est un outil **évolué** de synchronisation.

Introduits par Brinch & Hansen en 1972-73 et Hoare en 1974. Un moniteur =

- des variables d'état ;
- des procédures internes ;
- des procédures externes (points d'entrée) ;
- des conditions ;
- des primitives de synchronisation.

Les variables d'état sont manipulables par les procédures externes seulement (*encapsulation*).

## Un exemple simple de moniteur

```
Moniteur incr_decr  
incr_decr : moniteur ;  
  
var i : entier ;  
  
procédure incrémente ;  
début  
i := i + 1 ;  
fin ;  
  
procédure décrémente ;  
début  
i := i - 1 ;  
fin ;  
  
début  
i := 0 ;  
fin  
  
fin incr_decr.
```

Chaque procédure du moniteur est exécutée en **exclusion mutuelle**.

⇒ l'accès au moniteur s'effectue en exclusion mutuelle. Sauf si un processus exécute la primitive

attendre.

## Les instructions spéciales des moniteurs

Les variables de type **condition**.

Les primitives **vide**, **attendre** et **signaler**.

Soit  $c$  une condition.

- $c.attendre$  : bloque le processus et le place en attente de  $c$ .
- $c.vide$  : *vrai* si aucun processus n'est en attente de  $c$ , *faux* sinon.
- $c.signaler$  : **si non**  $c.vide$  **alors** réveiller un processus en attente de  $c$ .

## Rendez-vous entre $N$ processus

### Moniteur de Rendez-vous

```
rendez_vous : moniteur ;
```

```
. var n : entier ;  
. tous_là : condition ;
```

```
procédure arriver ;
```

```
  début
```

```
    n := n + 1 ;
```

```
    si n < N alors
```

```
      . tous_là.attendre ;
```

```
    tous_là.signaler ;
```

```
  fin ;
```

```
  début
```

```
    n := 0 ;
```

```
  fin
```

```
fin rendez_vous.
```

## À l'intérieur des moniteurs

Chaque **moniteur** possède une file d'attente **globale**.

Chaque variable **condition** référence une **file d'attente**.

- $c.attendre$  ⇨ placer le processus dans la file d'attente associée à  $c$ .
- $c.signaler$  ⇨ sortir le processus suivant de la file d'attente associée à  $c$ .
- $c.vide$  ⇨ teste si la file d'attente associée à  $c$  est vide.

Problème : un seul processus actif à la fois au sein d'un moniteur, donc comment implémenter la primitive **signaler** ?

*Note 1* : un moniteur est en général implémenté avec des ... sémaphores !

*Note 2* : c'est tout à fait logique, un système est bâti par niveaux d'abstraction successifs, un niveau  $i$  étant

implémenté par les primitives du niveau *i-1*.

## Problème des producteurs-consommateurs

processus producteur	processus consommateur
... produire(message_produit) ; tampon.déposer(message_produit) ; ...	... tampon.retirer(message_à_consommer) ; consommer(message_à_consommer) ; ...

À quoi ressemble le moniteur *tampon* ?

Moniteur <i>tampon</i>
tampon : <b>moniteur</b> ;  . <b>var</b> n : 0..N ; . non_plein, non_vide : condition ;  <b>procédure</b> déposer(m : message) ; <b>début</b> <b>si</b> n = N <b>alors</b> . non_plein.attendre ; n := n + 1 ; entrer(m) ; non_vide.signaler ; <b>fin</b> ;  <b>procédure</b> retirer( <b>var</b> m : message) ; <b>début</b> <b>si</b> n = 0 <b>alors</b> . non_vide.attendre ; sortir(m) ; n := n - 1 ; non_plein.signaler ; <b>fin</b> ;  . <b>type</b> message : ... ; . <b>var</b> file : tableau[0..N-1] de message ; . tête, queue : 0..N-1 ;  <b>procédure</b> entrer(m : message) ; <b>début</b> file[queue] := m ; queue := (queue + 1) <b>mod</b> N ; <b>fin</b> ;  <b>procédure</b> sortir( <b>var</b> m : message) ; <b>début</b> m := file[tête] ; tête := (tête + 1) <b>mod</b> N ; <b>fin</b> ;

```

début
n := 0 ;
tête := 0 ;
queue := 0 ;
fin

fin tampon.

```

## Problème des lecteurs-rédacteurs

Soit un fichier (SGBD, ...) manipulé par deux sortes de processus différents :

1. les **lecteurs** qui consultent le fichier sans en modifier le contenu ;
2. les **rédacteurs** qui peuvent en modifier le contenu.

Soient *nlect* et *nred* le nombre de lecteurs et rédacteurs accédant au fichier à un instant donné.

Il faut respecter les **contraintes d'intégrité** (maintien de la **cohérence**) :

- $nred = 0$  et  $nlect \geq 0$  ;
- $nred = 1$  et  $nlect = 0$  ;

### Moniteur *lecteur\_rédacteur*

```
lecteur_rédacteur : moniteur ;
```

```

. var ecr : booléen ;
. nl : entier ;
. c_ecr, c_lect : condition ;

```

```
procédure début_lire ;
```

```
début
```

```
nl := nl + 1;
```

```
si ecr alors
```

```

. c_lect.attendre ;

```

```

. c_lect.signaler ;

```

```
fin ;
```

```
procédure fin_lire ;
```

```
début
```

```
nl := nl - 1;
```

```
si nl = 0 alors
```

```

. c_ecr.signaler ;

```

```
fin ;
```

```
procédure début_écrire ;
```

```
début
```

```
si ecr ou nl > 0 alors
```

```

. c_ecr.attendre ;

```

```
ecr := vrai ;
```

```
fin ;
```

```
procédure fin_écrire ;
```

```
début
```

```
ecr := faux ;
```

```
si nl > 0 alors
```

```
. c_lect.signaler ;
```

```
sinon
```

```
. c_ecr.signaler ;
```

```
fin ;
```

```
début
```

```
ecr := faux ;
```

```
nl := 0 ;
```

```
fin
```

```
fin lecteur_rédacteur.
```

### Critiques :

La priorité est donnée aux **lecteurs**.

Questions :

1. comment donner la priorité aux **rédateurs** ?
2. comment **équitablement** gérer la priorité (mais est-ce souhaitable ?) ?



[Retour au sommaire.](#)