

## Retour sur les moniteurs

### Lecteurs-rédacteurs (priorité égale)

Priorité égale:

- on utilise un signal pour gérer la queue des processus en attente;
- des lecteurs et des rédacteurs peuvent être bloqués devant ce signal;
- un processus cessera d'attendre dès que plus aucun processus n'écrit;
- le signal est nommé *aucun\_rédact* (il transporte la condition *aucun rédacteur n'est en train d'écrire*);

Comment autoriser plusieurs lectures concurrentes?

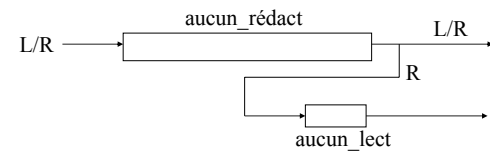
- cas 1: un rédacteur est en tête de la file *aucun\_rédact* ;
- cas 2: un lecteur  $l_1$  suivi d'un lecteur  $l_2$  sont en tête ;
- cas 3: un lecteur  $l_1$  suivi d'un rédacteur sont en tête ;
- etc.

## Objectifs

- (autres) exemples d'utilisation des moniteurs;
- sémantique complète des moniteurs;
- implémentation des moniteurs;

Solution:

- on introduit un 2<sup>ème</sup> signal;
- un processus rédacteur attend devant ce signal si une lecture est en cours;
- on appelle ce signal *aucun\_lect* (il transporte la condition *aucun lecteur n'est en train de lire*).



```

monitor lecteurs_rédacteurs;
defines début_lecture, fin_lecture, début_écriture, fin_écriture;

var nb_lecteurs: integer; (* nb de lecteurs en train de lire; init 0 *)
    nb_tot_rédacteurs: integer;
    (* nb de rédacteurs en attente ou en train d'écrire; init 0 *)

signal aucun_lect, (* aucun lecteur en train de lire *)
    aucun_rédact; (* aucun rédacteur en train d'écrire *)

```

```

procédure début_écriture;
begin
    nb_tot_rédact := nb_tot_rédact + 1;
    if nb_tot_rédact > 1 then aucun_rédact.wait end if;
    if nb_lect > 0 then aucun_lect.wait end if;
end début_écriture;

procédure fin_écriture;
begin
    nb_tot_rédact := nb_tot_rédact - 1;
    aucun_rédact.send;
end fin_écriture;

end lecteurs_rédacteurs;

```

```

procédure début_lecture;
begin
    if nb_tot_rédact > 0 then aucun_rédact.wait end if;
    nb_lect := nb_lect + 1;
    aucun_rédact.send;
end début_lecture;

procédure fin_lecture;
begin
    nb_lect := nb_lect - 1;
    if nb_lect = 0 then aucun_lect.send end if;
end fin_lecture;

```

## Sémantique des moniteurs classiques

Déjà mentionné:

- exclusion mutuelle;
- **wait** lève l'exclusion mutuelle;
- **send** réveille le processus éventuellement en attente du signal et, le cas échéant, bloque le processus qui envoie le signal;

monitor **m**; defines **p1**, **p2**;  
signal **s**;

```

procedure p1;
begin
...
s.wait;
...
s.send;
...
end p1;

```

```

procedure p2;
begin
...
s.send;
...
s.send;
...
end p2;

```

end **m**;

processus **X**: **m.p1**  
processus **Y**: **m.p1**  
processus **Z**: **m.p2**

Comme se passe l'exécution?

## Actions d'un processus

- *entrée*: action qui correspond à l'appel d'une procédure du moniteur;
- *wait*: action qui correspond à l'appel de la procédure wait;
- *send*: action qui correspond à l'appel de la procédure send;
- *sortie*: action qui correspond à la fin de l'exécution d'une procédure du moniteur.

## Etats d'un processus

- *activable*: état d'un processus occupant le moniteur. Un seul processus par moniteur peut être dans ce état;
- *stoppé*: état d'un processus susceptible de devenir activable à la seule condition qu'aucun processus ne soit activable dans le moniteur. Deux sous-états:
  - *stoppé-entrée*: état d'un processus désirant entrer dans le moniteur occupé;
  - *stoppé-send*: état d'un processus bloqué après avoir exécuté send;
- *attente*: état d'un processus ayant exécuté wait;
- *sorti*: état d'un processus se trouvant en dehors du moniteur (c-à-d ne se trouvant dans aucun des états ci-dessus).

## Changements d'état d'un processus

- *Transition simple*:  $(e) - a \rightarrow (e')$   
si un processus dans l'état  $e$  exécute l'action  $a$ , il passe dans l'état  $e'$ .
- *Transition couplée*:  $(e_1, e_2) - a \rightarrow (e'_1, e'_2)$   
si un processus  $p_1$  dans l'état  $e_1$  exécute l'action  $a$ , et qu'il existe un processus  $p_2$  dans l'état  $e_2$ , alors l'exécution de  $a$  par  $p_1$  fait passer  $p_1$  dans l'état  $e'_1$  et  $p_2$  dans l'état  $e'_2$ .

- si plusieurs règles s'appliquent lors d'une action, la première règle applicable dans la liste associée à l'action est choisie;
- les processus dans l'état *stoppé-send* sont gérés en *pile* ;
- les processus dans l'état *stoppé-entrée* sont gérés en *queue* ;
- les processus *en attente d'un signal* sont gérés en *queue* .

## Exemples

## 1. retour sur le transparent #9;

```

monitor m;  defines p1, p2;
signal s;
  procedure p1;
    begin
      ...
      s.wait;
      ...
      s.send;
      ...
    end p1;
  procedure p2;
    begin
      ...
      s.send;
      ...
    end p2;
end m;

processus X: m.p1
processus Y: m.p1
processus Z: m.p2

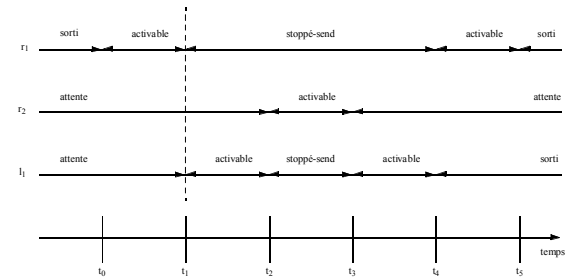
```

## Un exemple de sémantique des moniteurs (Portal)

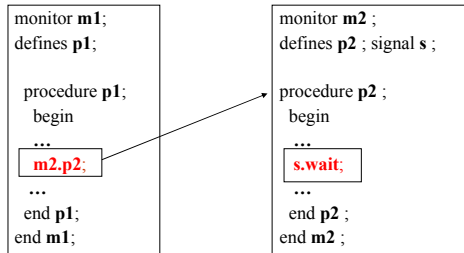
- |   |                                       |
|---|---------------------------------------|
| A <sub>1</sub> ) (sorti, activable)         | – entrée → (stoppé-entrée, activable) |
| A <sub>2</sub> ) (sorti)                    | – entrée → (activable)                |
| B <sub>1</sub> ) (activable, stoppé-send)   | – s.wait → (attente de s, activable)  |
| B <sub>2</sub> ) (activable, stoppé-entrée) | – s.wait → (attente de s, activable)  |
| B <sub>3</sub> ) (activable)                | – s.wait → (attente de s)             |
| C <sub>1</sub> ) (activable, attente de s)  | – s.send → (stoppé-send, activable)   |
| C <sub>2</sub> ) (activable)                | – s.send → (activable)                |
| D <sub>1</sub> ) (activable, stoppé-send)   | – sortie → (sorti, activable)         |
| D <sub>2</sub> ) (activable, stoppé-entrée) | – sortie → (sorti, activable)         |
| D <sub>3</sub> ) (activable)                | – sortie → (sorti)                    |

## 2. lecteurs-rédacteurs

(initialement: r1 en train d'écrire; l1, r2 attendent *aucun\_rédact*; r1 termine son écriture).



## Appels imbriqués de moniteurs



Le processus p appelle m1.p1, et se bloque dans le moniteur m2.

L'exclusion mutuelle est levée sur m2.

Est-elle levée sur m1?

Implémentation des moniteurs:  
exemple du noyau Portal

Contexte: priorité des processus.

- 0 (min) à 7 (max);
- priorité également associée à un moniteur;
- un processus de priorité **p** NE PEUT PAS appeler un moniteur de priorité  $m < p$  (vérifié à la compilation);
- lorsqu'un processus de priorité **p** s'exécute dans un moniteur de priorité **m**, sa priorité passe à **m** (évite le problème d'inversion de priorité rencontré dans le contexte des systèmes d'exploitation).  
A la sortie du moniteur la priorité du processus retombe à **p**.

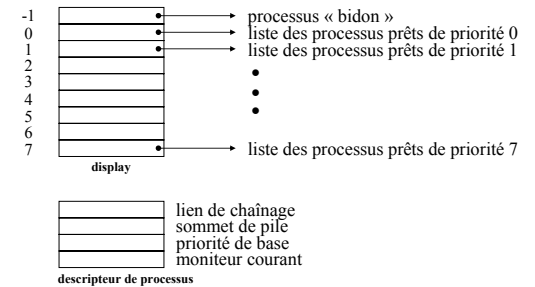
NON !!!

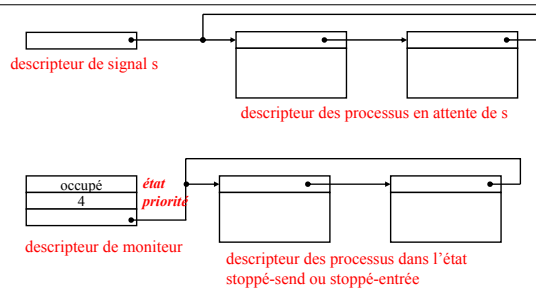
Idem en Java !!!

Remarques:

- dans un langage à objets l'appel imbriqué de moniteurs peut survenir naturellement, si l'on n'y prend garde;
- cela peut conduire à des interblocages;
- il faut être extrêmement attentif à ce problème.

## Structures de données du noyau





## Noyau vs. compilateur

- noyau « moniteurs » : implémente les quatre procédures d'entrée, de sortie, d'attente (wait) et d'envoi de signal (send);
- soit p une procédure d'un moniteur m:
 

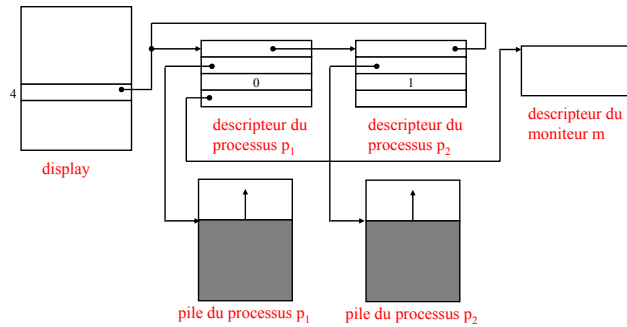
```

      procedure p;
      begin
      ...
      end p;
      
```

- le compilateur insère les appels aux procédures entrée et sortie:

```

procedure p;
code
  entrée (adresse du descripteur du moniteur);
  ...
  sortie;
end p;
  
```



## Procédure de base

procedure **processusSuivant**;

begin

**parcourir le display par ordre de priorité décroissante jusqu'à trouver une liste non vide;**

**sélectionner pour exécution le processus en tête de cette liste;**

end **processusSuivant**;



## Entrée dans un moniteur

```

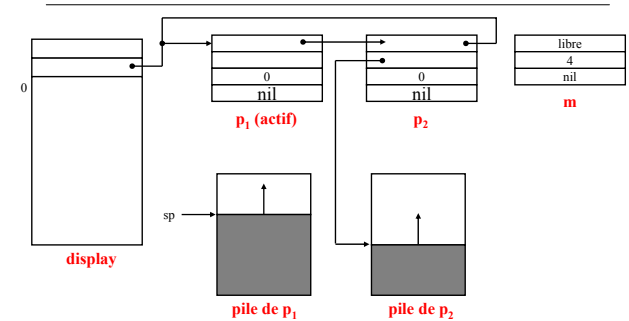
procédure entrée (nouveau moniteur: adresse de descripteur de moniteur);
begin
  empiler sur la pile du processus l'adresse du descripteur du moniteur courant
  (* contenu dans le descripteur du processus; cette information dans la pile sera
  utilisée lors de la sortie du moniteur *);
  if priorité nouveau moniteur < priorité processus then erreur end if;
  stocker dans le descripteur du processus l'adresse du descripteur du nouveau
  moniteur;
  if nouveau moniteur est libre then
    le nouveau moniteur devient occupé;
    if priorité courante du processus < priorité du moniteur then
      sortir le processus de la liste du display dans laquelle il se trouve;
      insérer le processus en tête de la liste du display correspondant à la priorité du
      moniteur
    end if
  else
    sortir le processus de la liste du display dans laquelle il se trouve;
    insérer le processus en queue de la liste "stoppé-send /stoppé-entrée" du nouveau
    moniteur;
    processusSuivant
  end if
end entrée;

```

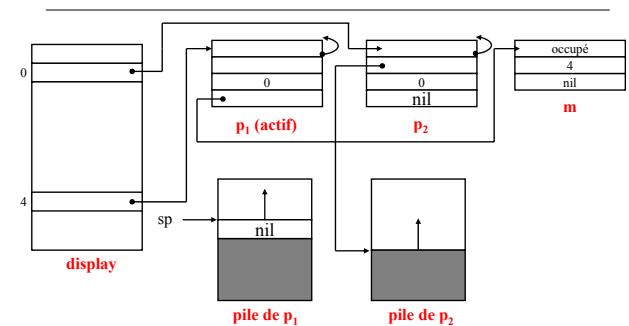
## Scénario illustratif

- processus  $p_1$ ,  $p_2$  (priorité de base 0)
- moniteur  $m$
- $p_1$  entre dans  $m$
- $p_1$  exécute `s.wait`
- $p_2$  entre dans  $m$
- $p_2$  exécute `s.send`
- $p_1$  sort de  $m$
- $p_2$  sort de  $m$

## 1) Situation avant l'entrée de $p_1$ dans le moniteur $m$



## 2) Situation après l'entrée de $p_1$ dans le moniteur $m$



## Wait

```

procédure wait (s: adresse de descripteur de signal);
begin
  (* le descripteur du moniteur courant est accessible via le descripteur du processus *)
  sortir le processeur de la liste du display dans laquelle il se trouve;
  insérer le processus en queue de la liste d'attente du signal s;
  if liste "stoppé-send / stoppé-entrée" du moniteur courant est vide
  then
    le moniteur courant devient libre
  else
    sortir le processus en tête de la liste "stoppé-send / stoppé-entrée" du moniteur
    courant;
    insérer ce processus en tête de la liste du display correspondant à la priorité du
    moniteur (* ce processus sera le prochain processus actif *)
  end if;
  processusSuivant;
end wait;

```

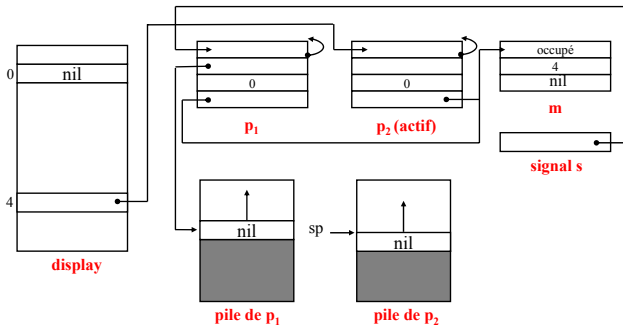
## Send

```

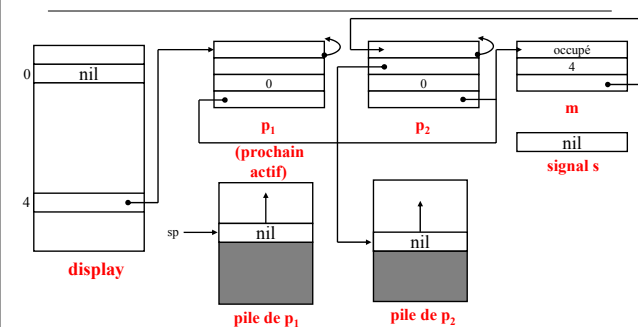
procédure send (s: adresse de descripteur de signal);
begin
  (* le descripteur du moniteur courant est accessible via le descripteur du processus *)
  if liste d'attente du signal s n'est pas vide
  then
    sortir le processus en tête de la liste d'attente du signal s
    (* appelons le le "processus débloqué" *);
    sortir le processus courant, c'est-à-dire celui qui exécute send, de la liste du
    display dans laquelle il se trouve;
    insérer le processus courant en tête de la liste "stoppé-send / stoppé-entrée" du
    moniteur courant;
    insérer le processus débloqué en tête de la liste du display correspondant à la
    priorité du moniteur (* ce processus sera le prochain processus actif *);
    processusSuivant
  end if;
end send;

```

### 3) Situation après exécution de s.wait par $p_1$ et entrée de $p_2$ dans le moniteur m



### 4) Situation après exécution de s.send par $p_2$





## Sortie de moniteur

```

procédure sortie;
begin (* le descripteur du moniteur courant est accessible via le descripteur du processus *)
  sortir le processus de la liste du display dans laquelle il se trouve;
  if liste « stoppé-send / stoppé-entrée » du moniteur courant est vide
  then le moniteur courant devient libre
  else
    sortir le processus en tête de la liste « stoppé-send/stoppé-entrée » du moniteur courant;
    insérer ce processus en tête de la liste du display correspondant à la
    priorité du moniteur courant
    (* ce processus sera le prochain processus actif *);
  end if;
  désempiler de la pile du processus (qui sort du moniteur) l'adresse d'un descripteur de
  moniteur (* empilé lors de l'entrée dans le moniteur *)
  if adresse de ce descripteur de moniteur = nil
  then (* le processus se trouve en dehors de tout moniteur *)
    insérer le processus en queue de la liste du display correspondant à la priorité de base du
    processus
  else
    insérer le processus en queue de la liste du display correspondant à
    la priorité du moniteur dans lequel se retrouve le processus
  end if;
  processusSuivant;
end sortie;

```

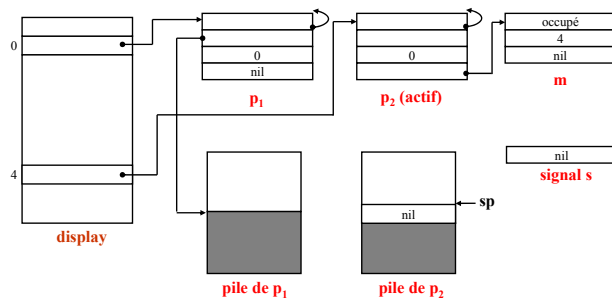
## Préemption sur interruption de l'horloge

Procédure appelée à chaque interruption de l'horloge:

```

procédure commutationHorloge;
begin
  sortir le processus actif de la tête de la liste du display dans laquelle il
  se trouve;
  insérer le processus en queue de la même liste du display;
  processusSuivant;
end commutationHorloge;

```

5) Situation après la sortie de  $p_1$  du moniteur  $m$ 

## Remarques sur les e/s en mode interruption

- pour un processus, l'attente d'une interruption est un problème de synchronisation;
- et donc, l'attente d'une interruption s'exprime naturellement à l'aide d'un mécanisme de synchronisation.

Exemple: mapping des interruptions sur « moniteurs + signaux »

1. lier un vecteur d'interruption (p.ex. 64) à un signal:  
signal interr\_écran [64]
2. dès lors, une interruption qui utilise le vecteur d'interruption 64 a le même effet qu'un processus exécutant `interr_écran.send` ;
3. attente d'une interruption: `interr_écran.wait` ;

## Exemple: écriture d'un caractère à l'écran

```

1  monitor [4] écran;
2  defines écrire;
3  type
4    bits = (b0, b1, b2, b3, b4, b5, interr, prêt, b8,
5            b9, b10, b11, b12, b13, b14, b15);
6    état = set of bits;
7  var
8    rē_écran at 177564B: état; (*registre d'état*)
9    rd_écran at 177566B: char; (*registre de donnée*)
10 signal
11   interr_écran [64B]; (*signal d'interruption*)
12 procedure écrire(c:char);
13   (*écrit le caractère c sur l'écran*)
14   code
15     if prêt not in rē_écran then
16       (*l'interface n'est pas prête*)
17       rē_écran := rē_écran + état(interr);
18       (*met le bit d'interruption à 1*)
19       interr_écran.wait; (*attend l'interruption*)
20       rē_écran := rē_écran - état(interr);
21       (*met le bit d'interruption à 0*)
22     end if;
23     rd_écran := c;
24   end écrire;
25 end écran;
```