

# Sémaphores

```

class sémaphore;
  n: integer;
  en_attente: queue de processus;

  procedure P( );
  begin
    n := n - 1;
    if n < 0 then bloquer le processus en queue de en_attente;
    end if;
  end P;

  procedure V( );
  begin
    n := n + 1;
    if n ≤ 0 then débloquer prss en tête de en_attente;
    end if;
  end V;

end sémaphore;

```

## Exclusion mutuelle à l'aide de sémaphores

Le *sémaphore* est un verrou généralisé pour lequel le champ *état* est de type entier (proposé par Dijkstra, 1968).

C'est un outil plus puissant que le verrou;

- *verrouiller* est remplacé par une opération appelée *P* ;
- *déverrouiller* est remplacé par une opération appelée *V* .

- P et V sont des sections critiques (et sont implémentés dans le noyau);
- si  $n \leq 0$ , la valeur absolue  $|n|$  indique le nombre de processus dans la queue *en\_attente*.

---

```
mutex: sémaphore init 1;
```

```
process p1;                process p2;
begin                      begin
...                        ...
mutex.P;                  mutex.P;
console.écrire(           console.écrire(
  'erreur sur disque');    'imprimante pas prête');
mutex.V;                  mutex.V;
...                        ...
end p1;                   end p2;
```



```
module imprimante;
  procedure allouer(var no_impr: integer);
  procedure rendre (no_impr: integer);
end imprimante;
```

```
process p;
  var no: integer;
begin
  ...
  imprimante.allouer(no);
  écrire sur l'imprimante no;
  imprimante.rendre(no);
  ...
end p;
```



## Un problème d'allocation de ressources

---

- pool de  $m$  imprimantes à partager entre plusieurs processus;
- chaque processus:
  - peut demander une imprimante;
  - utilise l'imprimante;
  - restitue l'imprimante.
- $m=1$  est identique au problème d'exclusion mutuelle.



Deux problèmes:

- autoriser au plus  $m$  processus en "section critique";
- attribuer une imprimante aux processus en "section critique".

```
module imprimante;
  const m = ...; // nombre d'imprimantes
  var
    occupé: array 1..m of boolean; // initialement false;
    mutex: sémaphore init 1;
    nb_imprimantes: sémaphore init m;
  ...
```



```

procedure allouer (var no_impr: integer);
begin
  nb_imprimante.P;
  // au moins une imprimante est libre
  mutex.P ;
  no_impr := 1;
  while occupé[no_impr] do
    no_impr := no_impr + 1;
  end while;
  occupé[no_impr] := true;
  mutex.V;
end allouer;

```

---

Quelques questions:

- peut-on permuter **nb\_imprimantes.P** et **mutex.P** dans *allouer*?
- peut-on permuter **nb\_imprimantes.V** et **mutex.V** dans *rendre*?
- **mutex.P** et **mutex.V** sont-ils nécessaires dans *rendre*?

---

```

procedure rendre (no_impr: integer);
begin
  mutex.P ;
  occupé[no_impr] := false ;
  mutex.V ;
  nb_imprimantes.V ;
end rendre;

```

## Problème des lecteurs et rédacteurs

---

Généralisation du problème d'exclusion mutuelle:

- ressource critique: fichier, données;
- deux classes de processus: processus lecteurs et processus rédacteurs;
- pas d'écritures concurrentes;
- pas de lecture pendant une écriture;
- lectures concurrentes autorisées.

---

Exemple:

- ressource critique: **compteur** ;
- processus rédacteur #1: **compteur** := **compteur** + 1 ;
- processus rédacteur #2: **compteur** := **compteur** - 1 ;
- processus lecteur: **v** := **compteur** ;

cet exemple ne montre pas pourquoi on interdit les lectures pendant les écritures.

## Variantes du problème

---

- priorités aux lecteurs;
- priorités aux rédacteurs;
- priorités égales (accès selon ordre d'arrivée).

Pour illustrer, considérons à l'instant t:

- section critique occupée par *lect1* ;
- arrivée dans l'ordre: *lect2*, *réd3*, *lect4*, *réd5*, *lect6* ;

---

Autre exemple:

- ressources critiques: **v1** et **v2** ;
- processus rédacteur:
  - if **v1** > **x** then
  - v1** := **v1** - **x**;
  - v2** := **v2** + **x**;
  - end if;
- processus lecteur: **somme** := **v1** + **v2** .

Ordre d'accès à la section critique:

- priorité aux lecteurs:
  - (*lect1*, *lect2*, *lect4*, *lect6*); *réd3*; *réd5* ;
- priorité aux rédacteurs:
  - lect1*; *réd3*; *réd5*; (*lect2*, *lect4*, *lect6*) ;
- priorités égales:
  - (*lect1*, *lect2*); *réd3*; *lect4*; *réd5*; *lect6* ;

## Spécification formelle du problème

à l'aide des *compteurs de Robert*:

- $\#act(P)$  = nombre de processus exécutant la procédure P;
- $\#att(P)$  = nombre de processus en attente de pouvoir exécuter la procédure P;

Dans le problème des lecteurs-rédacteurs:

- P correspond soit à la procédure *écrire*, soit à la procédure *lire*.

## Priorités égales (avec sémaphores)

Tentative #1:

**lr: sémaphore init 1;**

process <b>lecteur</b> ;	process <b>rédacteur</b> ;
begin	begin
...	...
<b>lr.P</b> ;	<b>lr.P</b> ;
<b>lr.V</b> ;	<b>écrire</b> ;
<b>lire</b> ;	<b>lr.V</b> ;
...	...
end <b>lecteur</b> ;	end <b>rédacteur</b> ;

Conditions réalisées:

- lecture possible lorsque  $\#act(écrire) = 0$  ;
- écriture possible lorsque  $\#act(écrire) = 0$  .

Lecteurs-rédacteurs sans priorité:

- lecture possible lorsque  $\#act(écrire) = 0$  ;
- écriture possible lorsque  $\#act(écrire) = 0$  et  $\#act(lire) = 0$  .

Priorité aux lecteurs:

- lecture possible lorsque  $\#act(écrire) = 0$  ;
- écriture possible lorsque  
 $\#act(écrire) = 0$  et  $\#act(lire) = 0$  et  $\#att(lire) = 0$  .

Priorité aux rédacteurs:

- lecture possible lorsque  $\#act(écrire) = 0$  et  $\#att(écrire) = 0$  ;
- écriture possible lorsque  $\#act(écrire) = 0$  et  $\#act(lire) = 0$  .

Tentative #2:

**lr, r : sémaphore init 1;**

**nb\_lecteurs: integer (\* init 0 \*);**

process <b>lecteur</b> ;	process <b>rédacteur</b> ;
begin	begin
...	...
<b>lr.P</b> ;	
<b>nb_lecteurs := nb_lecteurs + 1;</b>	<b>lr.P</b> ;
if <b>nb_lecteurs = 1</b> then <b>r.P</b> end if;	<b>r.P</b> ;
<b>lr.V</b> ;	
<b>LIRE</b> ;	<b>ECRIRE</b> ;
<b>nb_lecteurs := nb_lecteurs - 1;</b>	<b>r.V</b> ;
if <b>nb_lecteurs = 0</b> then <b>r.V</b> end if;	<b>lr.V</b> ;
...	...
end <b>lecteur</b> ;	end <b>rédacteur</b> ;

Version finale:

- protéger la ressource critique *nb\_lecteurs* par une section critique;
- peut être fait en ajoutant un sémaphore **mutex**.



module **lecteurs\_rédacteurs**;

**lr, r, mutex : sémaphore** init 1 ;  
**nb\_lecteurs**: integer (\* init 0 \*);

```
procedure début_lecture;
begin
  lr.P ;
  mutex.P ;
  nb_lecteurs := nb_lecteurs + 1;
  if nb_lecteurs = 1 then r.P end if;
  mutex.V ;
  lr.V ;
end début_lecture;
```

```
procedure fin_lecture;
begin
  mutex.P ;
  nb_lecteurs := nb_lecteurs - 1;
  if nb_lecteurs = 0 then r.V end if;
  mutex.V ;
end fin_lecture;
```



module **lecteurs\_rédacteurs**;

**lr, r, mutex : sémaphore** init 1;  
**nb\_lecteurs**: integer (\* init 0 \*);

```
process lecteur;
begin
  ...
  lr.P ; mutex.P ;
  nb_lecteurs := nb_lecteurs + 1;
  if nb_lecteurs = 1 then r.P end if;
  mutex.V ; lr.V ;
  LIRE;
  mutex.P;
  nb_lecteurs := nb_lecteurs - 1;
  if nb_lecteurs = 0 then r.V end if;
  mutex.V;
  ...
end lecteur;
```

```
process rédacteur;
begin
  lr.P ;
  r.P ;
  ECRIRE;
  r.V ;
  lr.V ;
  ...
end rédacteur;
```



```
procedure début_écriture;
begin
  lr.P ;
  r.P ;
end début_écriture;
```

```
procedure fin_écriture;
begin
  r.V ;
  lr.V ;
end fin_écriture;
```

end **lecteurs\_rédacteurs**;

