Communication Inter -Processus
Corrigés d'exercices

RAPPEL : Les semaphres et exp de deroulement :

```
Var vrrou : entier=0 ; //var système ;
Process P () ;
 Début
    …..
   …

     Demande d'accès en SC() ;
       SC ; //un ensemble d'action à exécuter en exclusion mutuelle
     Libérer l'accès à la SC() ;

   ….
Fin.

     Procédure Demande d'accès en SC() ;
      Debut
         Tq verrou != 0 faire fait;   //attente active
          Verrou=1 ;
      Fin ;

     Procedure Libérer l'accès à la SC() ;
        Debut
                verrou=0 ;
        Fin ;
```

ScenarioA
Process A execute Dem, il accède, verrou=1,
Si B demande, il trouve verrou = 1➔ attender

Scenario2 :
A commence à executer Dem,
?verrou !=0 (lire verrou, ensuite comparer avec 0 ?
On suppose que A a lu verrou, il trouve qu'il est à 0
Juste après FIN Q ;
B commence à exécuter Dem ;
?verrou !=0 (lire verrou, ensuite comparer avec 0 ?
On suppose que B a lu verrou, il trouve qu'il est à 0

Sémaphores : on suppose une ressourece partagée à 01 seul accès,
s= semaphore ;
Init (s, 1) ;

Process A()	Process B()	Process C()	➔temps
Debut	Debut	Debut	s=1,
…..	…..	…..	A: P(s)-> s=0; A execute ……SC……. V(s);s=-1+débloqueB
P(s) ;	P(s) ;	P(s) ;	B: P(s) ➔ s=-1; B en ATT en FA (bloqué)……., --SC --- V(s),s=0,réveillerC
SC() ;		SC() ;	C : P(s) ➔ s=-2 --- C en FA (en queue)…… ……----SC------V(s),s=1 ;

V(s) ; **..** **Fin.**	**SC() ;** **V(s) ;** **..** **Fin.**	**V(s) ;** **..** **Fin.**	

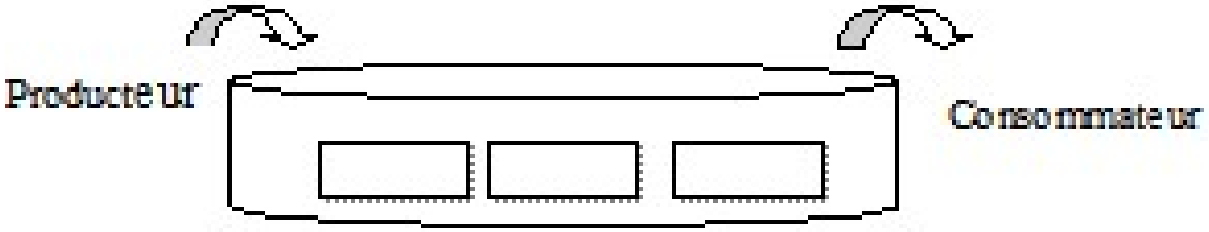
Producteur/Consommateur

Problème du producteur-consommateur

- **Principe**:

On considère 02 processus (**Producteur** et Consommateur qui partagent un tampon (taille N)

- Le producteur place des info. dans le tampon,
- Et, le consommateur, les récupère.



- **<u>Problème</u>** :

Le Producteur veut déposer un élément ➔ mais tampon <u>plein</u>

Solution :

- Producteur doit se bloquer s'il n'y a pas de case vide dans le tampon
- il est **débloque** si Consommateur vide une case du tampon.

On aura besoin d'une variable pour compter les cases pleines.

- Et si le consommateur veut consommer et qu' il ne trouve pas de case vide?
  - il doit se bloquer en attente d'une case vide qui sera nécessairement libérée par le consommateur
  - il sera débloqué par le consommateur lorsqu'il aura libérer une case sur N cases.

Donc : On prend 02 semaphores :

Full : qui compte cases occupées. Au debut, il y a 0 cases pleines, donc, il doit être initialisé à 0.

Empty : qui compte les cases vides. Au debut, il y a N cases vides, donc, il faut l'initialise à N.

Si la valeur de Empty = 0 alors → consommateur doit se bloquer

Si Full = N alors tampon vide → producteur doit se bloquer

Proposition à base de sémaphore.

variables :

Full : semaphore //nb d'emplacements occupés.

Empty : semaphore // nb d'emplacements vides.

Initialisation

init(full, 0); //semaphore bloquant ou privé

init(empty, N); //smaphore compteur

```
Void producer (void)
{ int item
        While (TRUE)
        { …
          P(empty) ;                  // demande tampon
            Insert-item(item) ;              // ecrire : nouvel élément ds tampon  - dépose un élément
          V(full) ;                  // incrémente pleins
}}
```

```
Void consumer (void)
{
Int item ;
While (TRUE)
{   ….
    P(full) ;            /*décrémente  nb  pleins */
      Item = remove-item() ;   //Lire
    V(empty);
    ….
}}
```

cette solution pose un problème ?

⇨ A un instant donné le tampon ne peut être accédé en lecture et écriture (Consommation/Production) en même temps. Donc, le dépôt et le retrait sont deux opérations qui doivent s'executer en EM stricte.
Le senario ci-dessous est non autorisé.

Prod()	Conso()
P(empty) //empty = 2	
ecrire	P(Full) //Full=3
	lire

Il faut donc, les proteger avec un semaphore binaire puisqu'elle doivent s'executer en EM stricte.

La Solution

variables :

Full : semaphore //nb d'emplacements occupés.
Empty : semaphore // nb d'emplacements vides.
Mutex : semaphore d'EM

Initialisation

init(mutex,1); // un seul processus entre en SC
(SEMAPHORES BINAIRES)
init(full, 0); //semaphore bloquant ou privé
init(empty, N); //smaphore compteure

```
Void producer (void)
{ int item
        While (TRUE)
        { …
          P(empty) ;                  // demande tampon
          P(mutex) ;                  //  entre en  section critique
            Insert-item(item) ;              // nouvel élément ds tampon
          V(mutex) ;                  //  quitte la section critique
          V(full) ;                  // incrémente pleins
}}
```

```
Void consumer (void)
{
Int item ;
While (TRUE)
 {      ....
       P(full) ;              /*décrémente  nb  pleins */
       P(mutex) ;
         Item = remove-item() ;
       V(mutex);
       V(empty);
       ....
}}
```

Lecteurs /Rédacteurs

Nous considérons une Base de données où plusieurs processus peuvent lire en même temps mais un seul peut écrire à la fois.

1. Discuter les différents cas possibles.
2. Proposer un code pour les processus lecteurs et le code d'un processus écrivain.
3. Discuter les propriétés que doit vérifier l'accès à la SC.

Corrigé :

1er Cas: Plusieurs Lecteurs et 0 Redacteur (HYPOTHESE)

Si on suppose qu'il n'y a pas de redacteur alors :
les lectures peuvent se derouler en même temps sur la BDD et puisqu'il n' y a pas de rédacteur

- ⇨ Aucune section critique
- ⇨ Donc : pas de SC

```
Process Lecteur()
{…
        <lire>
  …
}
```

2eme Cas: 00 Lecteurs et 01 seul Redacteur

1 seul redacteur, pas d'autre ecrituress ni lecture ➔ le redacteur est seul a ecrire➔ aucune contrainte
PAS de SC

```
Process Redacteur()
{…
        <Ecrire>
…
}
```

3eme Cas: 00 Lecteur et PLUSIEURS Redacteurs

Les ecritures ne peuvent se faire en même temps ➔ elles doivent s'executer en E. M. **STRICTE**
Donc : 'ecrire' est une SC ➔ nous allons proposer une solution a base de semaphore

Declartion : . …s : semphore;……
Initialisation : …Init (s, 1) ;……

```
Process Redacteur()
{…
  …P(s)……//Demande autorisation Écriture
        <Ecrire>
  …… //Libérer  Accès SC
  …
}
```

Rappel

Fct V(s) ;

{ s++ ; Si FA non vide alors

Debloquer Process en tet de FA ;

FSi ;

}

Deroulement :

Process Redacteur 1	**Process Redacteur 2**	**Process Redacteur 3**
	P(s), s est egale 1 donc s--, s=0	
	ECRIRE	
		P(s) s=-1
		Bloqué
P(s) s= -2		
Bloqué		
	V(s), s est = a -2 donc s++= -1;	
	Donc debloquer P3	
		Ecrire
		V(s), s=-1, s++=0
		Debloquer P1
ecrire		
V(s), s =0 ; s++ =1		

4eme Cas: 01 Lecteur et 01 Redacteur

Ne peuvent lire et écrire en même tamps ➔ les opérations Lecture et écriture doivent s'executer en E.M. STRICTE

⇨ Les operations 'lecture' et 'écriture' sont des SC

Declartion : s : semaphore **Initialisation** : ……Init (s,1)……	
Process Lecteur() {… …P(s)…… //Demande autorisation Lecture <lire> …V(s)…… //Libérer Accès SC … }	Process Redacteur() {… ……P(s)…… //Demande autorisation Écriture <Ecrire> ……V(s)…… //Libérer Accès SC … }

Faire le deroulement avec arrivée, en premier du recteur et ensuite du lecteur.

5eme Cas: 01 Lecteur et PUSIEURS Redacteurs

Toutes les ecritures doivent s'executer en EM entre elles et aussi avec l'operation de lecture.

➔ les opérations 'écriture' et l'operation 'Lecture' doivent s'executer en **E.M. STRICTE**

⇨ Les operations 'lecture' et 'écriture' sont des SC

Declartion : .....s : semaphore.....	
Initialisation : .....init (s, 1).....	

Process Lecteur()
{...
.....P(s)..... //Demande autorisation Lecture
<lire>
.....V(s)..... //Libérer Accès SC
...
}
	Process Redacteur()
	{...
	.....P(s)..... //Demande autorisation Écriture
	<Ecrire>
	.....V(s)..... //Libérer Accès SC
	...
	}

6eme Cas: Plusieurs Lecteurs et 01 Redacteur

Toutes les lectures peuvent s'executer en parallele **Mais**
elles doivent s'executer en EM avec l'operation d'écriture.

➔ toutes les opérations 'Lecture' paralleles
et l'operation 'écriture' doivent s'executer en E.M. STRICTE
Les operations 'lecture' et 'écriture' sont des SC

Pilosophes

1. Définir le problème des philosophes.
2. Proposer une ou des solutions à ce problème sans utilisation de sémaphores ? Critiquer ces solutions.
3. Proposer une solution avec utilisation de sémaphores ? Critiquer la solution.

Corrigé : Les philosophes

1ère Solution:

```
fonction Philosophe (int i)
Debut
  Tant que Vrai    faire
        Penser() ;
     PrendreForchette (i) ;
     PrendreForchette (i+1) ;
        Manger() ;
    PoserFourchette(i)
    PoserFourchette(i+1)
    Fait ;
  Fin
```

DONC

```
N : entier=5
m[5] : tableau de
       sémaphores
       binaires ;

pour i=0 à 4 faire
  init( m[i], 1) ;
fait ;
```

```
fonction Philosophe (int i)
Debut
  Tant que Vrai    faire
        Penser() ;
      PrendreForchette (i) ;
      PrendreForchette (i+1) ;
        Manger() ;
     PoserFourchette(i)
     PoserFourchette(i+1)
     Fait ;
  Fin
```

```
      // Nous pouvons définir les   fonctions
      // précédentes comme suit :
Fonction PrendreFourcette(k)
Debut
   P(m[k]) ;
Fin.     // retourne 1 si exécution avec succès

Fonction DeposerFourcette(k)
Debut
   V(m[k]) ;
Fin.    // retourne 1 si exécution avec succès
```

Est-ce que la solution respecte les conditions d'exclusion de concurrence en Exclusion Mutuelle?
Y a-t-il un problème d'interblocage? Peut on accepter la solution.

➔ Interblocage (chaque philosophe i a la fourchette i à un instant t, et il n'a pas la fourcette i+1

2ème Solution:

Principe de la solution :

- Un philosophe a trois états : "penser'', "faim", "manger".
- Un philosophe ne peut passer à l'état "manger" que si aucun de ses voisins n'est à l'état "manger".
- Nous avons les philosophes 0, 1, 2, 3 et 4, et Vd : voisin de droite, Vg : Voisin de gauche

```
N : entier=5
State[5] : tableau d'état  (penser, faim, manger)
S[5] : tableau de semaphores privés
mutex : sémaphores  binaires ;
    init(mutex, 1) ;
    pour i=0 à 4 faire
      init( S[i], 0) ;
    fait ;
     Define  Vg   (i+N-1)%N
     Define  Vd   (i+1)%N
    pour i=0 à 4 faire  init( State[i], penser") ;
    fait ;
```

```
AccèsSC(philosophe i)
Debut
   P(mutex)
     State[i]= "faim" ;

   Si state[Vg] != manger  et  state [Vd] != manger) Alors
                                          State[i]  = mange ;
                                          V(S[i])   // il se donne 1 droit d'accéder en SC r
   Fsi ;
   V(mutex)

   P(S[i]) ;        // il se bloque s'il n'a pas exécuté V(S[i]), jusqu'à ce que son voisin Vd ou Vg
                    //lui donne cette possibilité en exécutant pour lui V(S[i])
Fin.
```

```
SortieSC(philosophei)
Début
   P(mutex)
     State[i]= "penser" ;
                    // ?Vg veut et peut manger
     Si state[Vg] == faim et state [Vd] != manger)
       Alors  State[Vg] = mange ;
              V(S[Vg]) ; // il donne alors droit au Vg d'accéder enSC
     Fsi ;
                    // ?Vd veut et peut manger
     Si state[Vd] == faim et state [Vg] != manger)
     Alors        State[Vd] = mange ;
                  V(S[Vd]) ;              // il donne alors le droit au Vd d'accéder en SC mai
     Fsi ;
   V(mutex)
Fin.
```