

## I- LE PROBLEME DU PRODUCTEUR-CONSOMMATEUR :

### Hypothèse :

- **T** : **Tableau** de taille quelconque, soit  $N \geq 1$ . Le tableau T s'appelle souvent : **Tampon**, **Buffer** ou **Bac** désignant n'importe quelle zone mémoire alloué pour une structure de donnés.
- Un **objet** désigne un ensemble de données ou informations, etc. Il s'appelle aussi message.
- Il existe 02 types de processus. Le premier appelé **Producteur** et le deuxième appelé **Consommateur**.
- Le processus **Producteur** répète les deux instructions suivantes :
  - o **Obj = Produire\_un\_objet ();** // Créer un nouvel Objet soit "Obj";
  - o **Déposer (Obj, T);** // Déposer l'objet "Obj" dans une **case vide** de Tampon T
- Le processus **Consommateur** répète les deux instructions suivantes : .
  - o **Obj = Prélever\_un\_objet (T);** // retirer ou enlever un objet depuis une **case plein** de T
  - o **Consommer\_un\_objet (Obj);** // utiliser l'objet "Obj" ;
- La figure suivante résume le fonctionnement deux ces processus.

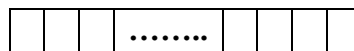
```

Producteur ()
Repete {

    Obj = Produire_un_objet ();
    Déposer(Obj, T);

} tant que vrai
    
```

**T** : tampon de Taille  $N \geq 1$



```

Consommateur ()
Repete {

    Obj = Prélever_un_objet (T);
    Consommer_un_objet (Obj);

} tant que vrai ;
    
```

### Problématique :

- Le problème du Producteur-Consommateur consiste à trouver comment synchroniser les deux processus de sorte que :
  - o Le Producteur ne dépose un objet que si le tampon il existe une **case vide** dans T.
  - o Le Consommateur ne prélève un objet que si il existe une **case plein** dans le tampon.
  - o Le Consommateur n'essaie pas de consommer un objet qui est en train d'être déposé par le Producteur (Pour éviter l'accès à la même case par le Producteur et le consommateur).
  - o Si le producteur est en attente parce que le tampon est plein, il doit être averti dès que cette condition cesse d'être vraie (**une case devient vide**).
  - o Si le consommateur est en attente parce que le tampon est vide, il doit être averti dès que cette condition cesse d'être vraie (**une case devient plein**).

### Solution :

- **Solution 01 :**

On introduit donc deux variables (de type sémaphore) caractérisant l'état du tampon :

- **NPLEIN** : nombre des cases pleins (contiens des objets) dans le tampon (début : 0) ;

- NVIDE : nombre des cases vides disponibles dans le tampon (N au début);  
Ainsi; la solution sera comme suit :

**Déclaration et initialisation**

```
Semaphore NPLEIN, NVIDE;
init(NPLEIN , 0); //// car il existe 0 cases plein au début.
init(NVIDE , N); // car il existe N cases vides au début.
Objet T [N] ; // tableau contient des objets;
```

Processus Producteur ( )	Processus Consommateur ( )
<pre>{   Objet obj ;   Repeter   {     <b>Obj = Produire_un_objet ();</b>     <b>P(NVIDE ) ;</b>     <b>Deposer(Obj, T);</b>     <b>V(NPLEIN ) ;</b>   }tant que vrai ; }</pre>	<pre>{   Objet obj ;   Repeter   {     <b>P(NPLEIN ) ;</b>     <b>Obj = Prélever _un_objet (T);</b>     <b>V(NVIDE ) ;</b>     Consommer_un_objet (Obj);   }tant que vrai ; }</pre>

**Solution 02 :**

La même solution précédent en détaillant les deux procédures :

- o Deposer\_un\_objet(obj, T) et
- o *Obj* = Prélever \_un\_objet (T);

Pour cela on est besoin de variables suivantes :

**Déclaration :**

```
int in;// indice de prochaine case vide de tableau à utiliser pour déposer un objet;
int out; // indice de prochaine case plein de tableau à utiliser pour prélever un objet;
```

**Initialisation :** In=out=0;

Deposer\_un\_objet (obj, T) ≡  $\left\{ \begin{array}{l} T[in] \leftarrow Obj ; \\ in \leftarrow (in+1) \text{ mode } n; \end{array} \right.$

*Obj* = Prélever \_un\_objet (T); ≡  $\left\{ \begin{array}{l} T[out] \leftarrow Obj ; \\ out \leftarrow (out+1) \text{ mode } n; \end{array} \right.$

- Pour éviter l'accès à la même case par le **Producteur** et le **Consommateur**, un sémaphore, appelé **MUTEX**, devra utiliser pour assurer qu'un seul processus accède à la fois au tableau T.

Semaphore Mutex;  
Init(Mutex,1);

Deposer\_un\_objet (obj, T)  $\equiv$   $\left\{ \begin{array}{l} P(mutex) \\ T[in] \leftarrow Obj; \\ V(Mutex); \\ In \leftarrow (in+1) \text{ mode } N; // \text{ écrit aussi } in \leftarrow in+1 \% N; \end{array} \right.$

Obj = Prélever\_un\_objet (T);  $\equiv$   $\left\{ \begin{array}{l} P(mutex) \\ Obj \leftarrow T[out]; \\ V(Mutex); \\ out \leftarrow (out+1) \text{ mode } N; \end{array} \right.$

La solution finale est le suivant :

**Semaphore** MUTEX, NPLEIN, NVIDE;  
**Objet** T [N] ;  
**Int** in,out;  
Init(MUTEX, 1); Init(NPLEIN , 0); Init(NVIDE , N);

Processus Producteur ( )	Processus Consommateur ( )
<pre> {   int in =0 ;   Objet obj ;   Repeter   {     Obj = Produire_un_objet ();     P(NVIDE ) ;     P(MUTEX) ;     Tampon[in] ← Obj;     V(MUTEX) ;     In ← (in+1) mode n ;     V(NPLEIN ) ;   } tant que vrai ; } </pre>	<pre> {   int out =0 ;   Objet obj ;   Repeter   {     P(NPLEIN ) ;     P(MUTEX ) ;     obj ← Tampon[out];     V(MUTEX) ;     Out ← (out+1) mode N ;     V(NVIDE ) ;     Consommer_un_objet (Obj);   } tant que vrai ; } </pre>