Chapitre 3: Communication entre processus

1- Introduction

2- Communication par partage de variables

2.1- Modèle du producteur/ consommateur
- 2.1.1- un producteur et un consommateur
  - Solution1
  - Solution 2
  - Solution 3
  - Propriétés
  - Gestion du tampon
- 2.1.2- Plusieurs producteurs et plusieurs consommateurs.

2.2- Modèle des lecteurs/ rédacteurs.
- a- Pas de priorité explicite
- b- Priorité aux lecteurs

3- Communication par échanges de messages

3.1- Communication par désignation
- 3.1.1. Désignation directe : communication dans CSP
- 3.1.2. Désignation indirecte : les boites aux lettres

3.2- Capacité des liens de communication
- Capacité nulle
- Capacité limité
- Capacité Illimitée

3.3- Remarques

1- Introduction

Dans un système d'exploitation, en plus de leur compétition pour l'acquisition de ressources, les processus peuvent coopérer pour réaliser des tâches communes. Cette coopération nécessite leur communication. Pour réaliser la communication, il est nécessaire d'utiliser des outils de synchronisation permettant de coordonner les processus dans leurs communications.

La communication peut se faire soit par partage de variables soit par échange de messages. Dans le premier cas, la synchronisation doit être prise en charge par les processus eux même, en utilisant les outils classiques tels que les sémaphores, selon leur logique de communication. Par contre dans le deuxième cas, la synchronisation est gérée par le système d'exploitation lui même. Pour cela, le système offre des primitives telle que *Send (message)* qui permet d'envoyer un message et *Receive (message)* pour recevoir un message.

2- Communication par partage de variables

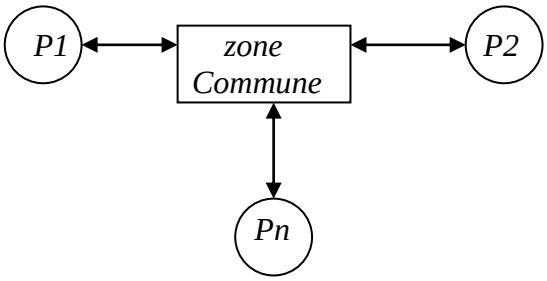

Figure 3.1 : Modèle de partage de mémoire.

Ex : Soit *Vp* la variable partagée par *P1* et *P2*.

P1	*P2*
-	-
X← *Vp ;*	*Vp*← *expr ;*
-	-

Chaque processus peut lire ou écrire dans cette zone et plusieurs peuvent réaliser cette opération en concurrence (voir Figure 3.1). Pour garantir la cohérence de ces variables, il faut gérer l'accès des processus à cette zone. D'ici surgit la nécessité d'utilisation des outils classiques de synchronisation. La gestion de la communication est laissée à la charge de l'utilisateur selon son application.

2.1- Modèle du Producteur / Consommateur

On distingue deux types de processus, les processus dont le rôle est de produire des informations et des processus dont le rôle est de consommer ces informations.
Pour réaliser cette communication, on utilise un tampon (Figure 3.2) qui est une structure de données comprenant n cases. Chaque case peut contenir une unité d'information (ou article) que peut produire un producteur (et que peut finalement consommer un consommateur)

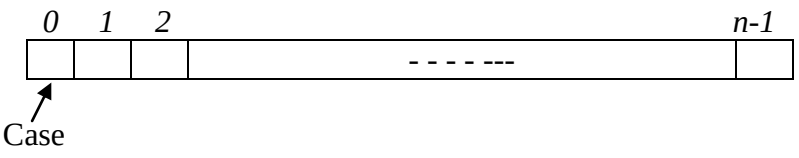

Figure 3.2 : Structure de tampon.

Les processus utilisent deux procédures pour accéder au tampon :

- Procédure *Déposer (article)*
- Procédure *Prélever (article)*

Le producteur/consommateur dépose/prélève une unité d'information à la fois. Soit *nb* le nombre d'articles contenus dans le tampon à un moment donné.

- Le producteur ne peut déposer que s'il y a de la place dans le tampon (*nb<n*).
- Le consommateur ne peut prélever que s'il y a au moins une case pleine (*nb<>0*),
- Un même article ne peut être prélevé qu'une seule fois.
- L'exclusion mutuelle doit être assurée au niveau d'une même case.
- Les deux processus doivent pouvoir accéder simultanément au tampon.

2.1.1- Un producteur et un Consommateur :

Nous allons présenter la solution de manière progressive de construction.

Solution 1

var nb : entier :=0;

```
Processus Producteur () ;
        Var Art : Tart ;
   Debut
        Repeter
          Produire (Art) ;
          Tq (nb=n) Faire rien Fait;
          Déposer (Art) ;
           nb :=nb+1 ;
          < Autres instructions>
        Jusqu'à Faux ;
   Fin.
```

```
Processus Consommateur ();
        Var Art : Tart ;
    Debut
        Repeter
          Tq (nb=0) Faire rien Fait ;
          Prélever( Art) ;
           nb :=nb-1 ;
           Consommer (art) ;
          < Autres instructions>
        Jusqu'à Faux
    Fin.
```

- Si la condition de dépôt ou de prélèvement n'est pas satisfaite, le processus correspondant attend de manière active ⇒ Consommation inutile du temps CPU.
- Conflit d'accès au niveau de *nb*.

Solution 2

Dans ce cas, si la condition n'est pas satisfaite, le processus doit se bloquer ⇒ utilisation des primitives suivantes : attendre, réveiller, tester l'état d'attente.

```
Processus Producteur ();
        var Art : Tart ;
   Debut
     Repeter
      Produire (Art) ;
      Si (nb=n) Alors Attendre Fsi;
      Déposer (Art) ;
      nb :=nb+1 ;
     Si attente(cons) Alors Réveiller(cons) Fsi ;
    < Autres instructions>
   Jusqu'à Faux ;
  Fin.
```

```
Processus Consommateur () ;
        var Art : Tart ;
Debut
  Repeter
     Si (nb=0) Alors Attendre Fsi;
     Prélever(art) ;
      nb :=nb-1 ;
     Si attente(prod) Alors Réveiller(prod) Fsi ;
      Consommer (art) ;
       < Autres instructions>
     Jusqu'à Faux
    Fin.
```

Cette solution, comme la première, présente le problème d'exclusion mutuelle au niveau de la variable partagée *nb*.

Solution 3: Utilisation des Sémaphores.

var *np, nv : entier ; np := 0 ; nv :=n ;*

```
Processus Producteur ;
var Art : Tart ;
  Debut
      Repeter
        Produire (Art) ;
   (1)⌈nv :=nv-1 ;
      ⌊ Si (nv=-1) Alors Attendre Fsi;
        Déposer (Art) ;
   (2)⌈ np :=np+1 ;
      ⌊ Si (np=0) Alors
        Réveiller (cons) Fsi ;
        < Autres instructions>
      Jusqu'à Faux ;
 Fin.
```

```
Processus Consommateur ;
var Art : Tart ;
  Debut
      Repeter

   (3)⌈ np := np-1 ;
      ⌊ Si (np=-1) Alors Attendre Fsi;
        Prélever ( Art) ;
   (4) ⌈ nv :=nv+1
      ⌊ Si (nv=0)Alors
        Réveiller (prod) Fsi ;
        Consommer (art) ;
        < Autres instructions>
      Jusqu'à Faux
   Fin.
```

Les séquences *(1), (2), (3) et (4)* correspondent respectivement aux primitives *P(nv), V(np), P(np)* et *V(nv),* ce qui nous conduit à la solution finale suivante.

Var *nv, np : Sémaphores ; nv :=n ; np :=0 ;*

```
Processus Producteur () ;
      var Art : Tart ;
  Debut
      Repeter
        Produire (Art) ;
        P(nv) ;
        Déposer (Art) ;
        V(np) ;
        < Autres instructions>
      Jusqu'à Faux ;
  Fin.
```

```
Processus Consommateur () ;
      var Art : Tart ;
  Debut
      Repeter
        P(np) ;
        Prélever( Art) ;
        V(nv) ;
        Consommer (art) ;
        < Autres instructions>
      Jusqu'à Faux
  Fin.
```

Propriétés

1- Si la consommation suit l'ordre de production, le producteur et le consommateur n'opèrent jamais sur la même case.

2- Le producteur et le consommateur ne se bloquent jamais en même temps.

Gestion du tampon

La gestion la plus classique du tampon est la gestion circulaire (voir Figure 3.3).

Figure 3.3 : Tampon circulaire.

On utilise deux pointeurs de cases :

- *t* : pointe la première case pleine
- *q* : pointe la première case vide

Les deux primitives s'implémentent comme suit :

```
Procedure Déposer (Art) ;
        Debut
          T[q] : = Art ;
           q    : = (q+1) mod n ;
        Fin ;
```

```
Procedure Prélever (Art) ;
        Debut
         Art : = T[t] ;
         t    : = (t+1) mod n ;
        Fin ;
```

n étant la taille du tampon.

2.1.2- Plusieurs producteurs et plusieurs consommateurs

Dans ce modèle, il est nécessaire de rendre l'accès au tampon en exclusion mutuelle au sein de chaque famille. La solution devient :

```
nv, np : Sémaphore ; nv :=n ; np :=0 ;
mutexp, mutexc : sémaphore ; mutexp :=1 ; mutexc :=1 ;
```

```
Processus Producteur () ;
      var Art : Tart ;
   Debut
      Repeter
        Produire (Art) ;
        P(nv) ;
        P(mutexp) ;
         Déposer (Art) ;
        V(mutexp) ;
          V(np) ;
        < Autres instructions>
      Jusqu'à Faux ;
   Fin.
```

```
Processus Consommateur ();
      Var Art : Tart ;
   Debut
      Repeter
        P(np) ;
        P(mutexc) ;
        Prélever( Art) ;
         V(mutexc) ;
        V(nv) ;
         Consommer (art) ;
          < Autres instructions>
      Jusqu'à Faux
   Fin.
```

- Remarquons que cette solution ramène le modèle de plusieurs producteurs et plusieurs consommateurs au modèle d'un seul producteur et un seul consommateur. La seule différence est que plusieurs producteurs (resp. consommateurs) peuvent franchir au même temps la primitive *P(nv) (resp. P(np))*, mais l'accès au tampon est en séquentiel. De ce fait, les deux propriétés énoncées précédemment sont aussi vérifiées dans ce modèle. Etant donné que les producteurs/ consommateurs peuvent produire/ consommer simultanément des articles, un gain en parallélisme est alors constaté comme montré dans la Figure 3.4.

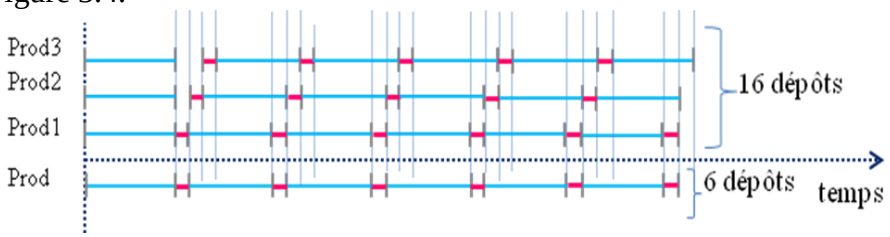

Figure 3.4 : Apport du modèle de plusieurs producteurs plusieurs consommateurs.

- D'autres modèles intermédiaires existent, par exemple :
  - Un producteur et plusieurs consommateurs ou
  - Plusieurs producteurs et un seul consommateur.
- En apparence, ce modèle n'apporte pas d'amélioration en termes de parallélisme. En réalité, le parallélisme existe toujours quand il s'agit de produire des articles car le dépôt ne consomme pas de temps.

Un sémaphore d'exclusion mutuelle est nécessaire au sein de chaque famille pour ces deux solutions.

2.2- Modèle des lecteurs/ rédacteurs.

Il s'agit de gérer l'accès concurrent à une ressource commune (un fichier par exemple) à plusieurs processus. Deux types d'opérations peuvent avoir lieu sur ce fichier :

- Consultation
- Modification.

Afin de garantir la cohérence des informations dans le fichier, les conditions suivantes doivent être assurées :

- Plusieurs lectures peuvent se faire en même temps,
- Pas d'écriture s'il y a au moins une lecture en cours.
- Pas de lecture ni écriture s'il y a une écriture en cours.

En superposition à ces conditions de cohérence, d'autres conditions liées à des priorités d'accès entre les deux familles de processus peuvent être ajoutées. Les priorités classiques sont : Pas de priorité explicite, priorité aux lecteurs, même priorité, priorité aux rédacteurs.

Nous allons examiner les deux premiers cas :

a- Pas de priorité explicite :

Le fichier est considéré comme une section critique pour un rédacteur. En effet, si un rédacteur est en cours aucun lecteur ni rédacteur ne peut y accéder. D'où l'utilisation du sémaphore *W*. Plusieurs lectures peuvent avoir lieu en même temps, d'où l'utilisation du compteur *nl*. Seul le premier lecteur (représentant de la famille) qui arrive teste l'occupation de la section critique par un rédacteur éventuel. Les autres accèdent directement si le premier réussit l'accès sinon se bloquent au niveau de *mutex* dont l'autre rôle est de protéger la variable de synchronisation *nl.*

```
Semaphore mutex:=1, W:=1;
Entier nl:=0;

Processus lecteur ();
Debut
        P(mutex);
        nl:=nl+1;
        Si nl=1 Alors P(W) Fsi;
        V(mutex);
        <Lecture>
        p(mutex);
        nl:=nl-1;
        Si nl=0 Alors V(W) Fsi;
        v(mutex);
Fin.
```

```
Processus redacteur ();
Debut
        P(W);
        <Ecriture>
        V(W);
Fin.
```

b- Priorité aux lecteurs

Sémaphore *mutex1:=1, mutex2:=1, W:=1;*
Entier nl:=0;

Processus *Lecteur;*
Debut
P(mutex1);
nl:=nl+1;
Si *nl=1* ***Alors*** *P(W)* ***Fsi****;*
V(mutex1);
<Lecture>
p(mutex1);
nl:=nl-1;
Si *nl=0* ***Alors*** *V(W)* ***Fsi;***
v(mutex1);
Fin.

Processus *rédacteur;*
Debut
P(mutex2);
P(W);
<Ecriture>
V(W);
V(mutex2);
Fin*.*

mutex2 empêche les rédacteurs d'attendre au niveau de *W* afin de donner la priorité aux lecteurs dont le premier, qui doit attendre (si une écriture est en cours), attend au niveau de *P(W)*, ''proche'' de la section critique.

3- Communication par échanges de messages

L'échange de messages permet aux processus de communiquer sans faire recours ni au partage ni à la gestion explicite d'une zone commune de mémoire. Le système de messages produit des mécanismes rendant possibles la communication entre processus et la synchronisation de leurs actions. Ce système dispose d'au moins des primitives :

- envoyer (message) et,
- recevoir (message).

Pour communiquer entre eux, un lien de communication doit être établi entre deux processus au préalable. Ce lien peut être implicite ou explicite, unidirectionnel ou bidirectionnel. Il permet la communication entre deux processus ou plusieurs. D'autres paramètres existent : la taille des messages (fixe ou variable) et la capacité du lien (limitée ou illimitée).
La primitive recevoir est par défaut bloquante dans le cas où le message n'est pas reçu. Cependant, dans certains systèmes, il est possible de configurer le non blocage et dans ce cas un statut d'exécution est retourné.

3.1- Communication par désignation

Pour communiquer entre eux, les processus doivent avoir un moyen pour se référencer mutuellement. Celui-ci peut être direct ou indirect.

3.1.1- Désignation directe

Dans ce cas, le correspondant est désigné directement par son nom. Les primitives sont comme suit :

envoyer (P, m) : envoyer le message *m* au processus *P*.
recevoir (Q, m) : recevoir le message *m* du processus *Q*.

Exemple : Producteur/ Consommateur.

Processus *prod () ;*	***Processus*** *cons ();*
var *m :------*	***var*** *m :------*
-	-
Repeter *------*	***Repeter*** *-----*
Produire_message (m) ;	*Recevoir (prod, m) ;*
Envoyer (cons, m) ;	*Consommer_message (m) ;*
Jusqu'à *Faux ;*	***Jusqu'à*** *Faux ;*
-------	------

Le consommateur est bloqué par défaut sur recevoir, si le message n'est pas arrivé. Dans ce cas, le lien est établi automatiquement entre le producteur et le consommateur et il est bipoint. Pour une paire de processus, il existe un seul lien.
Ce mode de communication est dit *symétrique :* C'est à dire, les deux processus se désignent *directement*. Une autre variante est l'adressage asymétrique. Seul l'émetteur désigne le récepteur. Les primitives deviennent :

envoyer (P, m) : envoyer le message *m* au processus *P*.
r*ecevoir (id, m)* : recevoir le message *m* de n'importe quel processus.

A la réception d'un message, id est affecté de l'identité du processus émetteur. L'inconvénient majeur de ces deux modes de communication est la modularité limitée.

Communication dans le langage CSP [Hoare 75]
Le langage CSP est caractérisé par :

- Un programme CSP est constitué d'un ensemble fixe de processus séquentiels non déterministes.
- La communication se fait par désignation directe et par rendez-vous. Les messages sont typés.

Considérons deux processus qui s'échangent des messages :
La forme *Q !m* veut dire envoyer m au processus *Q*.
La forme *P ?n* veut dire recevoir n du processus *P*.

La communication entre *P* et *Q* se produit quand les deux processus exécutent respectivement les deux primitives. L'effet est équivalent à $n \leftarrow m$; *m* doit être du même type que *n*, sinon blocage indéfini.
Si un processus envoie vers ou reçoit d'un processus terminé, celui-ci aura une terminaison anormale.

- Les structures de contrôle sont basées sur des commandes gardées de [Dijkstra 75]. Une commande a la structure :

  <*garde*> $\rightarrow$ <*liste de commandes*>

La garde est une liste de déclarations, d'expressions booléennes et de commandes d'entrées. Elle est évaluée comme *vrai* (ou *valide*), *faux*, ou *neutre*. Si au moins une condition booléenne est *vrai*, la partie des expressions booléennes est *vrai.*

- La partie commande d'entrée prend:
  - *vrai* : si l'émetteur est prêt à communiquer.
  - *neutre* : s'il n'est pas prêt.
  - *faux* : si l'émetteur est mort.
- Si la garde contient les deux parties:
  - Si l'expression booléenne est à faux; la garde est à faux,
  - Si l'expression booléenne est à *vrai*, la garde aura la valeur de la commande d'entrée.

La liste de commandes est exécutée si la garde est à *vrai* ou si elle est à *neutre* mais après disponibilité de l'émetteur pour la communication. L'exécution commence après la réception du message.

Exemple : *cpt>n ; P ?m → x :=x+1 ;*

- Une commande alternative est une combinaison de commandes gardées dont une seule au maximum est exécutée. Elle est choisie de manière aléatoire parmi celles évaluées à *vrai ou* sinon parmi celles à *neutre* mais après qu'une commande d'entrée devienne *vrai*. Si toutes les gardes sont à *faux*, l'exécution se termine en erreur.

  [G1 → Cmde1 □ G2 → Cmde2 □ ....□ Gn → Cmden]

- La boucle est spécifiée comme suit :

  **[G1 → Cmde1 □ G2 → Cmde2 □ ....□ Gn → Cmden]*

Cette commande répétitive exécute la commande alternative tant que possible. Elle se termine lorsqu' aucune garde n'est *valide*.

Exemple : Producteur / consommateur.

La solution est constituée de trois processus, le producteur, le consommateur et le communicateur qui est l'intermédiaire entre les deux processus. Celui-ci dispose du tampon et est à l'écoute des deux autres processus.

```
Com :  Tampon [0..99] art ;
       En, So : integer ;
       En :=0 ; So :=0 ;
       * [ En<So+100 ; Prod ? Tampon [ En Mod 100])  —> En :=En+1 ;
                        □
        So<En ; Cons ?encore())  → Cons ! Tampon [So mod 100] ;  So :=So+1 ;
       ]
```

Le producteur *Prod* exécute

```
Art :---- ;
Com ! art ;
```

Le consommateur *Cons* exécute

```
Art : .... ;
Com !encore() ;
Com ?art ;
```

3.1.1. Désignation indirecte : les boites aux lettres

Dans ce modèle, les messages sont envoyés (ou reçus) dans une boite aux lettres (BAL). Celle-ci peut être vue comme étant un emplacement où sont déposés (ou prélevés) les messages échangés. Chaque boite aux lettres est identifiée de manière unique. Deux processus ne peuvent communiquer que s'ils disposent d'une BAL commune. Ils peuvent communiquer par plus d'une BAL et plusieurs processus peuvent utiliser une même BAL pour communiquer.

Les primitives deviennent :

- *envoyer (B, message)* : envoyer *mess* vers la BAL *B.*
- *recevoir (B, message)* : recevoir *mess* de la BAL *B.*

- Une BAL peut être privé à un processus, dans ce cas il est le seul à l'utiliser en réception. La BAL disparaît à sa fin. Elle peut être aussi partagée avec d'autres processus.

Dans ce cas, si le message est destiné à un processus particulier, l'identité de celui-ci ou un signe le désignant doit accompagner le message. Elle disparaît si tous les processus terminent. Le partage peut s'effectuer implicitement, par héritage. Dès la création d'un processus, ce dernier partage avec son père ses BALs.

Un problème se pose dans le cas d'attente de plusieurs processus d'un message sur une même BAL. Deux solutions existent :
- Exclusion mutuelle sur l'exécution de recevoir ou
- Choix aléatoire d'un processus pour la réception du message.

- Une BAL peut être propriété du système d'exploitation, des primitives de création et de destruction sont alors nécessaires. Le premier processus crée la BAL, les autres récupèrent son identité (avec la même primitive d'appel) et par suite peuvent l'utiliser. Puisque cette BAL n'est propriété d'aucun processus, quand tous les processus qui lui font référence se terminent, celle-ci restera au niveau du système. Deux solutions sont envisageables, soit qu'un processus la détruit explicitement, soit que le système la détruit dès la fin du dernier processus qui en fait référence, soit par l'utilisation de la technique de ramasse muette pour récupérer son espace.

3.2- Capacité d'un lien de communication

C'est le nombre de messages que peut contenir temporairement un lien. Un lien peut être vu comme une file de messages attachés. Il existe Trois moyens d'implémentation de cette file selon sa capacité.

- Capacité zéro : La queue a une longueur maximale égale à zéro. Ainsi, le lien ne peut pas avoir de messages en attente. Donc, les deux processus doivent être synchronisés pour réaliser cette communication. Ce type de communication est appelé rendez-vous.
- Capacité limitée : Dans ce cas, la longueur du lien est n. Au maximum n messages peuvent être conservés par ce lien. Si le lien est rempli, l'émetteur doit attendre jusqu'à ce qu'un message soit extrait.
- Capacité illimitée : Tout nombre de messages peut être émis dans cette file. L'émission n'est pas bloquante (communication asynchrone).

3.3- Remarques

Dans les deux modes de communication (désignation directe ou indirect), un processus ne peut pas savoir si un message est arrivé à sa destination après l'envoi. Si l'arrivée est impérative, l'émetteur doit communiquer explicitement avec le récepteur selon un protocole qui permet d'assurer la réception du message.

Exemple :

***Processus** P*	***Processus** Q*
-	-
-	-
Envoyer (Q, message) ;	*Recevoir (P, message) ;*
Recevoir (Q, message) ;	*Envoyer (P, ''Ack'') ;*
-	-

Certains systèmes de messages utilisent une primitive qui envoie un message et qui permet de faire attendre un processus jusqu'à l'arrivée d'un accusé de réception.
Le système de messages est convenable dans un système distribué. Dans ce cas, les erreurs sont plus probables que dans un environnement centralisé. En centralisé, les messages sont implémentés par la mémoire partagée. Par contre, dans les systèmes distribués ils sont transférés par le lien de communication. Les problèmes liés sont: la terminaison du processus, la perte de messages, altération, le déséquencement, etc.