Chapitre 2 Communication

4. COMMUNICATION INTERPROCESSUS:

La communication est un outil important qui permet de faire évoluer l'ensemble des processus dans un système. Il existe deux stratégies pour réaliser la communication:

a-mémoire partagée
b-messages systèmes

Dans la cas (a), la communication entre processus est réalisée à travers des variables partagées. Dans ce qui suit, on s'intéressera au système de communication par messages.

La fonction du sous-système de messages est de permettre aux processus de communiquer sans avoir besoin de variables partagées. Il doit offrir deux opérations de base:

-SEND(message)
-RECEIVE(message)

Ainsi, si deux opérations P et Q désirent communiquer, elles doivent s'échanger des messages. Pour cela, on aura besoin de liens de communication. Cette liaison peut être directe, indirecte, unidirectionnelle ou bidirectionnelle.

4.1 Communication directe:

Tout processus qui désire communiquer avec ce mode, doit nommer explicitement le receveur ou l'expéditeur. Les primitives SEND et RECEIVE sont définies comme suit:

SEND(P,message): envoyer le "message" au processus P.
RECEIVE(Q,message): recevoir le "message" du processus Q.

La liaison associée à ce mode possède les propriétés suivantes:
-Un lien est établi automatiquement entre deux processus qui désirent communiquer. Les processus ont besoin de connaître leur identité respective.
-Le lien est établi exactement entre deux processus.

Exemple:
Soit le problème du producteur/consommateur qui peut être décrit par le code suivant:

```
PARBEGIN
    Producteur: Repeat
                    produit un élément
                     .
                     .
                    SEND(Consommateur,un élément)
                Until false
```

```
        Consommateur: Repeat
                RECEIVE(Producteur,un élément)
                .
                .
                consomme élément reçu
              Until false
    PAREND
```

4.2 Communication indirecte:
Les messages sont envoyés et reçus à travers des boites aux lettres. Ces dernières peuvent être vues comme des objets où l'on peut déposer et retirer des messages. Chaque boite aux lettres dispose d'un identificateur unique. Les processus peuvent alors communiquer à travers plusieurs boites aux lettres.

Les primitives SEND et RECEIVE sont définies comme suit:
SEND(A,message): envoyer le "message" à la boite aux lettres A.
RECEIVE(A,message): recevoir le "message" de la boite aux lettres A.

Dans ce cas, un lien peut être établi entre deux processus s'ils possèdent une boite aux lettres en commun. Un lien peut être établi entre plusieurs processus.

4.3 Capacité des liaisons:
La liaison dispose d'une capacité qui détermine le nombre de messages qui y résident temporairement. Elle peut être vue comme une file associée à la liaison. Il y a trois façons d'implanter cette file.

4.3.1 Capacité nulle:
La liaison ne peut pas garder de message en attente, la file doit toujours être de longueur 0. Par conséquent, la communication doit s'établir par RENDEZ-VOUS. Dans ce cas, l'expéditeur doit attendre jusqu'à ce que le destinataire puisse recevoir le message.

4.3.2 Capacité limitée:
La file dispose d'une longueur finie, au plus n messages peuvent être en attente dans cette file. Si elle n'est pas pleine et qu'un message est envoyé, il est alors placé en queue de la file et l'expéditeur peut continuer son exécution. Par contre, si la file est pleine, l'expéditeur est mis en attente jusqu'à la libération de place dans cette file.

4.3.3 Capacité illimitée:
La file possède une longueur potentiellement infinie, le nombre de messages l'est aussi. De plus, l'expéditeur n'est jamais mis en attente.

4.4 Les messages:
Les messages échangés entre processus peuvent être répartis en trois classes:
a-taille fixe
b-taille variable
c-message typé

a) L'implantation de ce type de message est simple, l'inconvénient est que les programmes qui les utilisent deviennent difficiles.

b) L'implantation devient difficile mais la programmation est facilitée.

c) Dans ce cas, on associe à chaque boite aux lettres un type, les messages sont envoyés et reçus à travers les boites qui possèdent leurs types.

Remarque: Processus terminé et message
-Si un processus receveur P attend un message d'un processus Q qui a terminé son exécution (cas d'erreur par Exemple). Alors, il restera bloqué indéfiniment.
-Si un processus expéditeur P envoie un message à un processus Q qui a terminé son exécution. Dans un système à buffers, P continue son exécution. si P désire connaître que son message est bien arrivé à Q; il doit programmer explicitement un accusé de réception: le processus P sera alors bloqué.

Exemple: communication interprocessus sous UNIX
Les moyens de communication interprocessus UNIX sont: pipes (tubes), signaux, sockets, messages et mémoire partagée. On donnera un exemple utilisant le pipe .
Le pipe est un moyen de communication unidirectionnel entre deux processus. Les deux noms locaux d'un pipe sont fd[0] et fd[1]. La primitives de création d'un pipe est:

```
pipe(fd);
int fd[2];
```

Elle permet de créer un pipe entre deux processus, fd[1] est réservé à l'écriture alors que fd[0] est réservé à la lecture. La primitive pipe retourne -1 en cas d'erreur.

Communication père-fils: Un fils hérite des tubes crées par le père et de leurs descripteurs.

```
#include <stdio.h>
#include <errno.h>
#define TAILLE 30 /* par exemple */
main ()
{
char envoi [TAILLE], reception [TAILLE];`int p [2], i, pid;
strcpy (envoi, "texte transmis au tube");
if (pipe (p) < 0)
{
perror ("erreur de creation du tube ");
exit (1);
}
if ((pid = fork()) == -1)
{
perror ("erreur au fork ");
exit (2);
}
if (pid > 0) /* le pere ecrit dans le tube */
{
```

```
for (i=0 ; i<5; i++) write (p[1], envoi, TAILLE);
wait (0);
}
if (pid == 0) /* le fils lit dans le tube */
{
for (i=0 ; i<5 ; i++) read (p[0], reception, TAILLE);
printf (" --> %s\n", reception);
exit (0);
}
}
```