THEME :

COMMUNICATION INTER PROCESSUS PAR LES TUBES NON NOMMES

Sommaire

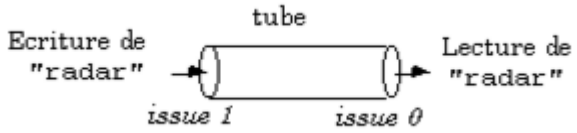
1. GENERALITES

1.1. Qu'est ce qu'un tube

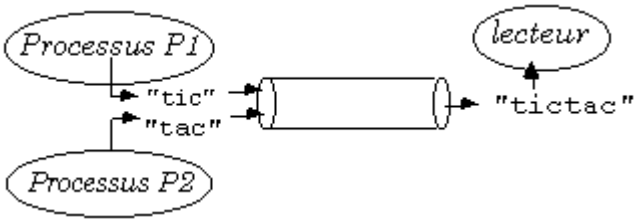
Un tube est un moyen de transmission de données d'un processus à un autre. Comme il est implanté par fichier, il sera désigné par des descripteurs et manipulés par des primitives read() et write() . Mais avec des particularités suivantes :

- La communication est unidirectionnelle. On écrit à un bout et un lit à l'autre (d'où le nom de tube). Cela entraine qu'il faut au moins deux descripteurs pour manipuler un tube.
- La communication est faite en mode FIFO (first in first out). Premier écrit, premier lu.
- Ce qui est lu quitte définitivement le tube et ne peut être relu. De même , ce qui est écrit est définitivement écrit.
- La transmission est faite en mode flots continu d'octets. L'envoi consécutif de deux séquences « abcd » et « efg » est semblable à « abcdefg » et peut être lu en totalité ou en morceaux comme « ab », « cde » et « efg » par exemple.
- Pour fonctionner un tube doit avoir au moins un lecteur et un écrivain. Il peut en avoir plusieurs.
- Un tube a une capacité finie et il y'a une synchronisation type producteur/consommateur entre lecteurs et écrivains : un lecteur peut avoir parfois attendre qu'il y'ait de la place dans le tube avant de pouvoir y écrire.

1.1.1. Illustration 1 : fonctionnement d'un tube en mode FIFO un lecteur et un écrivain



1.1.2. Illustration 2 : un lecteur et deux écrivains



1.2. Les tubes non nommés ou sans nom

Les tubes sans nom sont des liaisons unidirectionnelles de communication. La taille maximale d'un tube sans nom varie d'une version à une autre d'UNIX, mais elle est approximativement égale à 4KO. Les tubes sans nom peuvent être crées par l'appel système pipe() ou par le shell.

1.2.1. Les tubes non nommés du shell

Les tubes non nommés du shell sont créés par l'opérateur binaire « | » qui dirige la sortie standard d'un processus . les tubes de communication de shell sont supportés par toutes les versions d'UNIX.
Exemple. La commande shell suivante crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe.
Elle détermine le nombre d'utilisateurs connectés au système en appelant « who », puis en comptant les lignes avec *wc* .

Leibnitz> who | wc –l

Le premier processus réalise la commande « who ». le second processus exécute la commande *wc* –l. les résultats récupérés sur la sortie standard du premier processus sont dirigés vers l'entrée standard du deuxième processus via le tube de communication qui les relie. Le processus utilisant la commande « who » ajoute dans le tube une ligne d'information par l'utilisateur du système. Le processus réalise la commande *wc –l* récupère les lignes d'information pour en calculer le nombre total. Le résultat est affiché à l'écran. Les deux processus s'exécutent en parallèle, les sorties du premier sont stockées sur le tube de communication. Lorsque le tube devient plein, le premier processus est suspendu jusqu'à ce qu'il y'ait de l'espace nécessaire pour stocker une ligne d'information. De façon similaire, lorsque le tube devient vide le second processus est suspendu jusqu'à ce qu'il y'ait au moins une ligne d'information dans le tube.

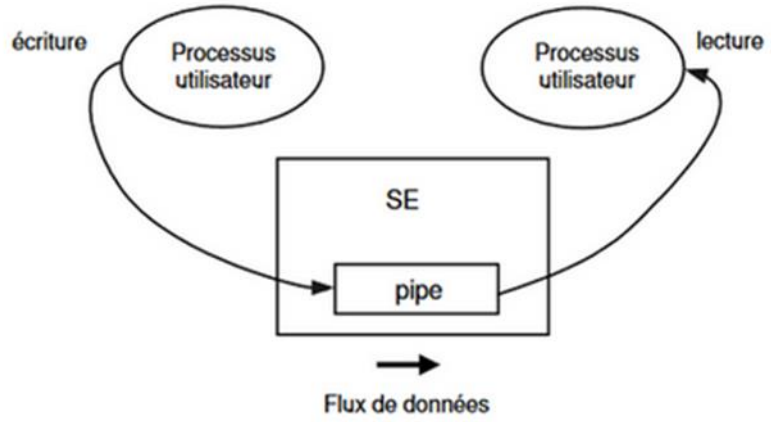
1.2.2. Les tubes non nommés de type pipe

Un tube de communication sans nom est créé par l'appel système pipe, auquel on passe un tableau de deux entiers :

Int pipe (int descripteurs [2]) ;

Les tubes sans nom sont, en général utilisés pour la communication entre processus père et ses processus fils avec un d'entre eux qui écrit sur le tube appelé processus écrivain et l'autre qui lit à partir du tube appelé processus lecteur.

- Le processus père crée un tube de communication sans nom en utilisant l'appel système pipe().
- Le processus père crée un ou plusieurs fils en utilisant l'appel système fork().
- Le processus écrivain ferme l'accès en lecture du tube.
- De même le processus lecteur ferme l'accès en écrivain du tube.
- Les processus communiquent en utilisant les appels système write() et read()
- Chaque processus ferme son accès au tube lorsqu'il veut mettre fin à la communication via le tube.



1.3. Problématique des tubes sans nom

Ce type de communication pose le problème de sections critiques : le moment ou les processus accèdent aux données partagées. En effet, si deux processus accèdent en même temps à une donnée commune il peut se produire différents résultats :

- un ou plusieurs processus concernés plantent

- un ou plusieurs processus concernés est interrompu : il (s) doit (ou doivent) attendre que la donnée commune soit libérée.

2. LES PRIMITIVES

Les tubes sans nom sont liaison unidirectionnelle de communication donc la taille maximale peut varie d'une version a une autre d'UNIX. Elle est approximativement égale à 4ko. Le fonctionnement de la communication sans nom repose sur trois primitives principales :

1- La primitive de création d'un tube : pipe()
2- La primitive de Lecture dans un Tube : read()
3- La primitive d'écriture dans un tube : write()

2.1. Primitive de création d'un tube : pipe()

Les tubes sont créés par la primitive pipe() qui est un appel système auquel on passe un tableau de 2 entiers. Les deux entiers sont appelés des descripteurs et mis en paramètre dans les deux positions du tableau. Ils peuvent être considérer comme une valeur entière que le système d'exploitation utilise pour accéder à un fichier.

- Un descripteur pour accès en lecture P[0]
- Un descripteur pour accès en écriture P[1]

Au retour de notre appel système, en cas d'échec le pipe nous renverra -1 sinon en cas de succès on aura 0 et tube aura été créé. Il est noté que seul note que seul le processus créateur du tube et ses descendants peuvent accéder au tube.

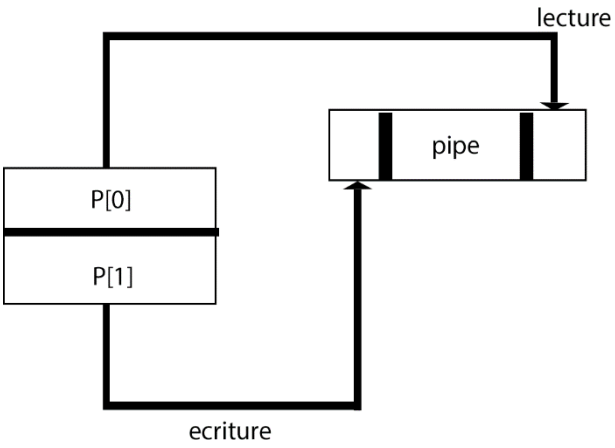
Exemple : Déclaration pour créer un tube :

```
#include <unistd.h>

int P[2] ;

Pipe(p) ;
```

Illustration : Pipe Lecture et Ecriture



2.2. La primitive de Lecture dans un Tube : Read()

La primitive Read() permet de lire dans un fichier mais cette opération doit se faire de façon atomique. On utilise le descripteur p[0]. Les précautions à prendre voudrait qu'un processus ferme systématique les descripteurs donc il n'a pas besoin : on lit dans le tube et on ferme le descripteur d'écrire p[1]. Cela permet d'éviter des erreurs aboutissant à des situations d'inter blocages (Deadlock).

Exemple Bout de programme « Code Complet juste en bas »

```
 char contenu [100] ;

int p[2] ;

…..

close(p[1]) ;

read(p[0], contenu,20) ;
```

Commentaire : On demande ici la lecture de 20 caractères dans le tube p. les caractères lus sont rendus dans la zone de contenu.

2.3. La primitive d'écriture dans le Tube :Write()

L'écriture dans le tube se fait en utilisant la primitive write() et le descripteur p[1] cette fois.

Exemple : Bout de programme « Code complet en dessous »

```
char contenu[100] ;
int p[2] ;
….
close(p[0]) ;
contenu = '' communication unidirectionnelle '' ;
write(p[1] , contenu,20) ;
```

Commentaire : On fait une demande d'écriture de 20 caractères dans le tube de descripteur p[1.] la séquence à écrire est prise dans la zone de contenu. write renvoie le nombre d'octet écrit. Ici aussi on ferme le descripteur de lecture en close(p[0]) ;

Code complet : Communication unidirectionnelle des processus fils et père.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(){
        int p[2];
        char contenu[100];
```

```
        char *texte = "communication unidirectionnelle";


        pipe(p) ;           // appel système pour créer un tube sans nom
        if(fork()==0){
                close(p[0]);     // fermeture du descripteur de lecture
                write(p[1] , texte, 20);
                close(p[1]) ;          // fermeture du descripteur d'écriture


        }else{
                close(p[1]);
                read(p[0] , contenu , 20);
                close(p[0]);
        }
        return 0;
}
```

3. Application de la communication interprocessus par les tubes non nommés dans le cas du problème producteur consommateur

3.1. Principe

On dispose d'un tampon qui est ici un tableau de N cases (N=10) , on considère que le producteur a produit un objet lorsqu'il a écrit le caractère '1' dans l'une des cases du tableau qui contenait le caractère '0', et que le consommateur a consommé lorsque celui-ci a écrit le caractère '0' dans l'une des cases qui contenait le caractère '1' ; d'autres part on dispose de deux tubes non nommés de communication où dans un tube le producteur notifie qu'il a produit un objet le consommateur lit la notification du producteur et lui répond en notifiant dans le deuxième tube qu'il a consommé un objet.

3.2. Code source

Ici le processus fils est consommateur et le processus père le producteur

```
 #include <stdio.h>
#include <unistd.h>
#include <string.h>
```

```
#define R 0
#define w 1
#define N 10
 int main(){
 int  fd1[2], fd2[2], nboctet;   //le producteur écrit dans fd1 et le consommateur dans fd2
 int  nbobjet=0,  i, j,  p=0;
char    message[100], tampon[N];
char* mp="le producteur a produit un objet";         //message du producteur
char* mc="le consommateur a consommé un objet";      //message du consommateur
 for(i=0;i<N;i++)
 {
          tampon[i]='0';         // Au départ le tampon est vide
  }
 pipe(fd1);         //création du pipe producteur
  pipe(fd2);      //création du pipe consommateur

  if(fork()==0)
  {
              // processus consommateur
       while(1)
      {
            close(fd2[R]) ;       //fermeture du pipe du consommateur en lecture
           if(nbobjet==0)
           {
                  //si le consommateur trouve que le nombre d'objets dans le tampon est nul
                  sleep(2);         //alors il dort
             }
             for(i=0;((i<N)&&(p!=1));i++)
            {
                 // tant que le consommateur n'a pas encore consommé la variable p !=1
                if(tampon[i]=='1')
```

```
            {
               //s'il ya au moins 1 objet dans le tampon alors le consommateur consomme
               tampon[i]='0';// alors le consommateur consomme un objet dans le tampon
                p=1;
                nbobjet--;     // le consommateur décrémente le nombre d'objet
              }
            }
            p=0;
            write(fd2[w],mc,strlen(mc)+1);   /*le consommateur écrit dans son pipe où le
            producteur à accès en lecture*/
            nboctet=read(fd1[R],message,100);  //le consommateur lit le message écrit
            dans le pipe du producteur
              close(fd2[w]);
              printf("le consommateur a lu  %d octets dans le fichier du producteur et le
            message lu est:%s puis le consommateur A son tour A consommer
            ",nboctet,message);   /* affichage du nombre d'octet et du message lu dans le
            pipe producteur*/
              printf("l'etat du tampon est: ");

           for(i=0;i<N;i++)
            {
                  printf("%c",tampon[i]);            // affichage de l'état du tampon
              }
          }
      }
     else
    {
            //processus producteur
            while(1)
            {
                  close(fd1[R]);
                  if(nbobjet==N)
                  {
```

```
                    //si le nombre d'objet du tampon==N(tampon plein)
                    sleep(2);       //alors le producteur entre en sommeil
                }
                for(i=0;  ((i<N)&&(p!=1))  ;i++)
                {
                        // s'il ya au moins une place vide dans le tampon alors le
                        producteur produit

                        if(tampon[i]=='0')
                        {
                            tampon[i]='1';    // alors le producteur produit un objet
                            dans le tampon
                            p=1;
                            nbobjet++;        // le nombre d'objet est incrémenté
                        }
                    }
                    p=0;
                    write(fd1[w],mp,strlen(mp)+1);  /*le producteur écrit dans son pipe
                    où le consommateur à accès en lecture*/
                    nboctet=read(fd2[R],message,100); //le producteur lit le message
                    dans le pipe du consommateur
                    close(fd1[w]);
                    printf("le producteur a lu  %d octets dans le fichier du
                    consommateur et le message lu est :%s puis le producteur A son tour A
                    produit  ",nboctet,message);
                    printf("l'etat du tampon est: ");
                for(i=0;i<N;i++)
                {
                    printf("%c",tampon[i]);          // affichage de l'état du tampon
                }
            }
        }
    return 0;}
```