Notes sur la gestion des tubes unix

Licence 3^{ieme} année, Bruno Jacob

Octobre 2004

1 Introduction

Il s'agit du mécanisme de communication caractéristique du système Unix entre des processus. Les tubes de communication ou *pipes* peuvent être vus comme des files (*first in, fisrt out*). Ils ne permettent pas la rémanence de l'information.
Les tubes standards sont seulement possibles entre des processus ayant un ancêtre commun.
Les tubes nommés peuvent faire communiquer tous les processus locaux à une machine.

2 Les tubes

2.1 Introduction

Un tube est implémenté comme un fichier mise à part qu'il ne possède pas de nom. Les communications qui transitent par lui sont unidirectionnelles. Un tube à une taille limitée : si un processus tente décrire dans un tube plein, alors il est suspendu.

2.2 Création

La création d'un tube est réalisée par la primitive `pipe`.

```
#include <unistd.h>
int pipe(int p[2]);
```

Elle correspond à la création de 2 descripteurs de fichiers, l'un permettant d'écrire dans le tube et l'autre d'y lire, par les opérations de lecture/écriture classiques (`read`/`write` par exemple).

À l'appel de `pipe`
- `p[0]` = descripteur par lequel on peut lire
- `p[1]` = descripteur par lequel on peut écrire
- si OK alors la valeur de retour = 0, sinon -1

Donc, si un processus père crée un tube et qu'il fait ensuite un `fork` pour créer un processus fils, comme le fils est une copie conforme du père, il héritera lui aussi du tube. Il aura donc lui aussi les mêmes descripteurs en lecture/écriture aux extrémités de ce tube. Le père et le fils pourront donc s'échanger des informations via ce tube.

Remarques :
- la seule façon de placer une fin de fichier dans un tube est de fermer celui ci en écriture dans tous les processus qui l'utilise (`close(p[1])`)
- Si un tube est fermé en lecture par tous les processus qui l'utilise et qu'un processus tente d'écrire dedans alors ce dernier recevra le signal `SIGPIPE` qui, s'il n'est pas capturé, interrompt le processus
- la primitive `read` renvoie `0` lorsque la fin de fichier est atteinte, c'est à dire
  - si le tube est vide **et**
  - si tous les processus qui utilisent ce tube l'ont fermé en écriture

2.3 Exemples

Communication entre un processus père et un processus fils.

```
#include <stdio.h>
#include <unistd.h>      /* pipe, fork, close, read, write */
#include <stdlib.h>      /* exit */
#include <sys/types.h>   /* fork, wait  */
#include <sys/wait.h>    /* wait */

int
main()
{
      int tube[2] ; /* tube de communication */
      char c ;

      /*-----*/

      if( pipe(tube) )
      {
            perror("pb sur creation du tube");
            exit(-1);
      }
```

```
    switch( fork())
    {
       case -1: /* erreur */
            {
                 perror("pb fork");
                 exit(-2);
            }
       case 0 : /* fils = recoit les lettres minuscules */
            {

                 /* le fils n'ecrit pas dans le tube */
                 close(tube[1]) ;
                 /* de plus cela permet d'atteindre la fin du tube */

                 /* lecture dans le tube */
                 while( read( tube[0] , &c , 1 ))
                      write( 1 , &c ,1 );

                 printf("\n") ;
                 exit(0);
            }
       default : /* pere = filtre les minuscules pour le fils */
            {
                 int cr ;

                 /* le pere ne lit pas dans le tube */
                 close(tube[0]);

                 while( (c=getchar()) != EOF )
                 {
                      if( (c<='z') && (c>='a') )
                      {
                           /* Ecriture dans le tube */
                           write(tube[1] , &c , 1 ) ;
                      }
                 }
                 /* Fermeture du tube en ecriture */
                 /* (pour que le fils atteigne la fin du tube) */
                 close( tube[1] );

                 /* Attente fin du fils */
                 wait(&cr);
            }
    }
    exit( 0 );
}
```

3 Les tubes nommés

3.1 Introduction

Les tubes nommés allient les propriétés des tubes standards et celles des fichiers. Comme les tubes ordinaires, les tubes nommés sont de taille limitée, les lectures/écritures sont réalisées suivant un mode *fifo*, les informations lues sont obligatoirement extraitent et les opérations de type `seek` sont interdites.

3.2 Création des tubes nommés

Un tube nommé peut être créé par n'importe quel processus par `mknod`, dont la fonction générale est de créer des catalogues et des fichiers spéciaux ou ordinaires.

```
#include <sys/stat.h>
int mknod(const char * ref, mode_t mode, dev_t droits);
```

`mknod` créer un i-noeud pour le fichier de nom `reference`.
Remarques

- `mode` spécifie le type et les droits d'accès.
  - type = `S_IFIFO` pour créer un tube nommé
  - droits = droits d'accès aux 3 classes d'usagers (`user, group, others`)
  Par exemple `S_IFIFO | 0666` crée un tube nommé accessible en lecture/écriture par tous.
- `mknod` n'est utilisable par un programmeur standard (différent du superutlisateur) **que** pour créer des tubes nommés.

3.3 Généralités

Tous processus qui veut communiquer par un tube nommé doit connaitre son nom. Il doit ouvrir par `open` un de ses descripteurs selon le type d'opération qu'il doit faire (lecture ou écriture).

- si un processus essaie d'ouvrir un tube nommé en lecture il sera suspendu jusqu'à ce qu'il existe un processus qui ouvre ce tube en écriture
- si un processus essaie d'ouvrir un tube nommé en écriture il sera suspendu jusqu'à ce qu'il existe un processus qui ouvre ce tube en lecture

3.4 Exemple

Communication entre 2 processus indépendants

3.4.1 Création du tube nommé

```
#include <stdio.h>
#include <stdlib.h>     /* exit */
#include <sys/stat.h>   /* mknod */

int
main( int nb_arg , char * tab_arg[])
{


     if( nb_arg != 2 )
     {
          fprintf( stderr , "%s - Creation d'un tube énomm\n\n" , tab_arg[0] );
          fprintf( stderr , "usage : %s <nom du tube nomme>\n" , tab_arg[0] );
          exit(-1);
     }

     if( mknod( tab_arg[1] , S_IFIFO | 0666 , 0 ) )
     {
          perror("Pb sur creation du tube nomme");
          exit(-2);
     }

     printf("Creation du tube %s reussie\n" , tab_arg[1] );
     exit(0);
}
```

3.4.2 Ecriture dans le tube nommé

```
#include <stdio.h>
#include <unistd.h>     /* close, read, write */
#include <stdlib.h>     /* exit */
#include <sys/types.h>  /* open */
#include <sys/stat.h>
#include <fcntl.h>

int
main( int nb_arg , char * tab_arg[])
{
     char c ;
     int fd_tube ;

     /*-----*/

     if( nb_arg != 2 )
     {
          fprintf( stderr , "%s - Ecriture d'un tube énomm\n\n" , tab_arg[0] );
          fprintf( stderr , "usage : %s <nom du tube nomme>\n" , tab_arg[0] );
          exit(-1);
     }
```

```
    /* Ouverture du tube en ecriture */
    if( (fd_tube=open( tab_arg[1] , O_WRONLY , 0 )) == -1 )
    {
        perror( "Pb sur ouverture du tube nomme en ecriture");
        exit(-2) ;
    }

    /* Filtrage des minuscules */
    while( (c=getchar()) != EOF )
    {
        if( (c<='z') && (c>='a') )
        {
            /* Ecriture dans le tube */
            write( fd_tube , &c , 1 ) ;
        }
    }

    /* Fermeture du tube */
    close( fd_tube );

    exit( 0 );
}
```

3.4.3 Lecture dans le tube nommé

```
#include <stdio.h>
#include <unistd.h>     /* close, read, write */
#include <stdlib.h>     /* exit */
#include <sys/types.h>  /*open */
#include <sys/stat.h>
#include <fcntl.h>

int
main( int nb_arg , char * tab_arg[])
{
    char c ;
    int fd_tube ;

    /*-----*/

    if( nb_arg != 2 )
    {
        fprintf( stderr , "%s - Ecriture d'un tube énomm\n\n" , tab_arg[0] );
        fprintf( stderr , "usage : %s <nom du tube nomme>\n" , tab_arg[0] );
        exit(-1);
    }

    /* Ouverture du tube en lecture */
    if( (fd_tube=open( tab_arg[1] , O_RDONLY , 0 )) == -1 )
```

```
    {
        perror( "Pb sur ouverture du tube nomme en lecture");
        exit(-2) ;
    }

    /* Lecture dans le tube */
    while( read( fd_tube , &c , 1 ))
        write( 1 , &c ,1 );
    printf("\n");

    /* Fermeture du tube */
    close( fd_tube );

    exit( 0 );
}
```