IUP MIAGE
Système d'exploitation

Communications par tubes

1 L'appel système `pipe()`

L'appel système

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

crée une structure de communication appelée tube (*pipe* en anglais).

Il retourne deux descripteurs de fichiers, `fd[0]` et `fd[1]`.

`fd[0]` est ouvert en lecture sur le tube et `fd[1]` est ouvert en écriture sur le tube.

Les valeurs écrites sur `fd[1]` par la primitive `write()` pourront être lues sur `fd[0]` avec la primitive `read()`.

Une opération de lecture (`read()`) sur un pipe vide

– est bloquante si un descripteur est encore ouvert sur le tube en écriture ;
– retourne 0 si tous les descripteurs en écriture sur le tube ont été fermés.

`pipe()` retourne normalement 0 ; -1 en cas d'erreur.

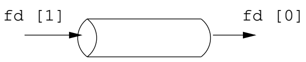

FIG. 1 – Un tube est associé à deux descripteurs de fichiers

Exercice 1
Le programme de la figure 2 écrit et lit dans un tube.

Question 1.1 Donnez le résultat de l'exécution de ce programme.

Question 1.2 Que se passerait-il si le compte de données écrites dans le tube était de 1024 au lieu de 6 (`STRLEN`) (mais que le compte des données lues reste 6) ?

Question 1.3 Que se passerait-il si l'ordre des appels systèmes `read()` et `write()` était inversé ?

2 Communication inter-processus par tube

Deux processus peuvent communiquer via un tube.

Le schéma de fonctionnement de base est le suivant ; on suppose que le père écrit dans le tube alors que le fils lit dans le tube (figure 3) :

1. Le processus crée le tube ;
2. Le processus fait un appel à `fork()` pour créer un fils.
3. Les deux processus père et fils possèdent alors chacun un descripteur en lecture et en écriture sur le tube.
4. Le processus père ferme son descripteur en lecture. Le processus fils ferme son descripteur en écriture sur le tube.
5. Le processus père peut écrire sur le tube ; les valeurs écrites pourront être lues par le fils.

Id: tubes.tex,v 1.13 2001/11/22 17:32:04 marquet Exp

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

#define STRING "hello"
#define STRLEN strlen (STRING)

int
main (void)
{
  char buf [1024] ;
  int fds [2] ;

  strcpy (buf, STRING) ;

  pipe (fds) ;

  for (;;) {
    fprintf (stderr, "*") ;
    write (fds[1], buf, STRLEN) ;
    read (fds[0], buf, STRLEN) ;
  }
}
```

FIG. 2 – Un processus écrit et lit dans un tube

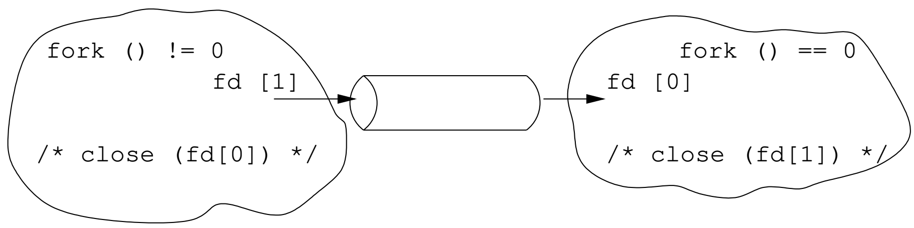

FIG. 3 – Partage d'un tube entre un processus père et un processus fils

```
% cat essai.c
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>

int
main (void)
{
  pid_t retour ;
  int tube[2], lettre = 0, chiffre = 0 ;
  char k ;
  if (pipe (tube) == -1) { perror ("creation pipe impossible\n") ; exit (1) ; }
  switch (retour = fork ()) {
    case -1 : perror ("Creation impossible") ; exit(1) ;
    case 0  :
      printf ("processus fils\n") ;
      /* le tube est ici ferme en ecriture: le dernier descripteur ouvert   */
      /* en ecriture sur le tube sera dans le processus pere. Quand celui   */
      /* ci fermera ce descripteur, le read effectué par le fils renverra 0 */
      close (tube[1]) ;
      while (read (tube[0], &k, 1) >0)
        if (isdigit (k)) chiffre++ ;
        else lettre++ ;
      printf ("%d chiffres recus\n", chiffre) ;
      printf ("%d lettres recues\n", lettre) ;
      exit (0) ;
    default :
      printf ("pere: a cree processus %d\n", retour) ;
      close (tube[0]) ;
      while (read (0, &k, 1) >0)
        if (isalnum(k)) write (tube[1], &k, 1) ;
      /* le tube est ici ferme en ecriture : un read sur le */
      /* tube vide retournera 0 dans le processus fils      */
      close (tube[1]) ;
      wait (0) ;
      printf ("pere: a recu terminaison fils\n") ;
  }
}
% ./essai
pere: a cree processus 14643
processus fils
qw;12;r5^D
3 chiffres recus
3 lettres recues
pere: a recu terminaison fils
```

FIG. 4 – Communication père/fils via un tube

Pour établir une communication bidirectionnelle, on utilise deux tubes.

Le programme `essai` de la figure 4 lit des caractères sur l'entrée standard et envoie les lettres et chiffres à son processus fils. Le processus fils compte les lettres et les chiffres et affiche les résultats à la fin. Le processus père attend la terminaison du fils pour s'arrêter.

Exercice 2
Donnez le code d'un programme dont le fonctionnement est le suivant : un processus père crée deux fils ; ces deux fils communiquent par un tube. Le premier fils envoie une suite de caractères au second fils. À la terminaison de la transmission, les deux fils terminent. Le père attend ses fils pour terminer lui-même.

Exercice 3
Soit le programme suivant : un processus père crée 3 fils et dialogue (père vers fils) avec chacun d'eux via un tube. Le père envoie des entiers que les fils lisent et affichent. Le programme de la figure 5 propose une solution initiale pour ce problème.

Expliquez pourquoi ce programme n'est pas correct.

Exercice 4
Étudiez la terminaison de la version de ce programme donnée à la figure 6.

Exercice 5
Un processus père lit des caractères sur l'entrée standard et filtre pour l'un des ses fils les lettres et pour l'autre les chiffres. Le premier fils calcule le nombre d'occurrences de chacune des lettres dans un tableau pendant que le second calcule la somme des chiffres. Lorsque le père atteint la fin de fichier sur l'entrée standard, il en avise ses fils qui envoient leurs résultats au père qui se charge de les afficher sur la sortie standard.

3 Copie d'un descripteur de fichier

Les primitives

```
#include <unistd.h>

int dup(int oldfd) ;
int dup2(int oldfd, int newfd) ;
```

dupliquent un descripteur de fichier.

`dup (oldfd)` retourne un nouveau descripteur ; ce descripteur est le plus petit qui n'est pas utilisé par le processus. Le nouveau descripteur référence le même fichier que `oldfd`; utilise le même pointeur de position et a le même mode d'accès (lecture, écriture...). `dup()` retourne -1 en cas d'erreur.

`dup2(oldfd, newfd)` duplique le descripteur `odlfd`. Le nouveau descripteur retourné est égal à `newfd`. Si `newfd` est utilisé, il est d'abord libéré comme avec `close()`. Le nouveau descripteur référence le même fichier que `oldfd`, utilise le même pointeur de position et a le même mode d'accès (lecture, écriture ...). `dup2()` retourne la valeur de `newfd`, -1 en cas d'erreur.

L'utilisation typique de ces primitives est de rediriger les entrées/sorties d'un processus. Par exemple, le programme suivant redirige l'entrée standard du processus depuis le fichier dont le nom est donné avant d'exécuter la commande `wc`.

```
% cat redirect.c
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
```

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

#define NBFILS  3
#define MAXVAL  9

static void
creerFils (int descr)
{
  int val ;
  int pid ;

  /* mon numero de processus */
  pid = getpid () ;

  /* Lecture de valeur depuis le descripteur et affichage */
  while (read (descr, &val, sizeof (int)) > 0)
    printf ("Valeur %d par le processus %d\n", val, pid) ;

  /* Fermeture du descripteur */
  close (descr) ;
} /* creerfils () */

void
main ()
{
  int i, fils ;
  /* les tubes pour chacun des fils */
  int tubes [NBFILS][2] ;

  /* Creation des tubes */
  for (fils=0 ; fils < NBFILS; fils++)
    if (pipe (tubes [fils]) == -1) {
      perror ("pipe") ;
      exit (-1) ;
    }

  /* Creation des fils */
  for (fils=0 ; fils < NBFILS; fils++)
    switch (fork()) {
      case -1 :
        perror ("fork") ;
        exit (-1) ;
      case 0 : /* fils */
        /* ferme le tube en ecriture */
        close (tubes [fils][1]) ;
        /* creation du fils */
        creerFils (tubes [fils][0]) ;
        /* terminaison du fils */
        exit (0) ;
      default : /* pere */
        /* ferme le tube en lecture */
        close (tubes [fils][0]) ;
    }

  /* Emmision d'entiers vers les fils */
  for (fils=0, i=0; i < MAXVAL; i++, fils = i%NBFILS)
    write (tubes [fils][1], &i, sizeof (int)) ;

  /* Fermeture des tubes en ecriture */
  for (fils=0 ; fils < NBFILS ; fils++)
    close (tubes [fils][1]) ;

  /* Attente de la terminaison des fils */
  for (fils=0 ; fils < NBFILS; fils++)
    wait (0) ;
} /* main () */
```

FIG. 5 – Programme erroné

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

#define NBFILS  3
#define MAXVAL  9

static void
creerFils (int descr)
{
  int val ;
  int pid ;

  /* mon numero de processus */
  pid = getpid () ;

  /* Lecture de valeur depuis le descripteur et affichage */
  while (read (descr, &val, sizeof (int)) > 0)
    printf ("Valeur %d par le processus %d\n", val, pid) ;

  /* Fermeture du descripteur */
  close (descr) ;

} /* creerfils () */

int
main ()
{
  int i, fils ;

  /* les tubes pour chacun des fils */
  int tubes [NBFILS][2] ;

  /* Creation des tubes et des fils */
  for (fils=0 ; fils < NBFILS; fils++) {

    /* Creation du tube */
    if (pipe (tubes [fils]) == -1) {
      perror ("pipe") ;
      exit (-1) ;
    }

  /* Creation du fils */
    switch (fork()) {
      case -1 :
        perror ("fork") ;
        exit (-1) ;
      case 0 : /* fils */
        /* ferme le tube en ecriture */
        close (tubes [fils][1]) ;
        /* creation du fils */
        creerFils (tubes [fils][0]) ;
        /* terminaison du fils */
        exit (0) ;
      default : /* pere */
        /* ferme le tube en lecture */
        close (tubes [fils][0]) ;
    }
  }

  /* Emmision d'entiers vers les fils */
  for (fils=0, i=0; i < MAXVAL; i++, fils = i%NBFILS)
    write (tubes [fils][1], &i, sizeof (int)) ;

  /* Fermeture des tubes en ecriture */
  for (fils=0 ; fils < NBFILS ; fils++)
    close (tubes [fils][1]) ;

  /* Attente de la terminaison des fils */
  for (fils=0 ; fils < NBFILS; fils++) {
    fprintf (stderr, "Attente d'un fils...") ;
    wait (0) ;
    fprintf (stderr, "ok\n") ;
  }
} /* main () */
```

FIG. 6 – Programme dont la terminaison est à étudier

```
#include <fcntl.h>
#include <unistd.h>

int
main (int argc, char *argv[])
{
  int fd ;
  if (argc != 2) {
    fprintf (stderr, "Usage: %s fichier\n", argv[0]) ;
    exit (EXIT_FAILURE) ;
  }

  fd = open (argv[1], O_RDONLY) ;
  if (fd == -1) { perror ("open") ; exit (EXIT_FAILURE) ; }

  /* redirection de l'entree standard */
  if (dup2 (fd, 0) < 0) { perror ("dup2") ; exit (EXIT_FAILURE) ; }

  system ("wc") ;
  if (errno) { perror ("system") ; exit (EXIT_FAILURE) ; }

  exit (EXIT_SUCCESS) ;
}
% ./redirect redirect.c
       26      91      560
```

Exercice 6
Donnez un programme qui réalise l'équivalent de la commande Shell

```
% ls | wc
```

Exercice 7
Donnez un programme qui réalise l'équivalent de la commande Shell

```
% ps axu | grep -v root | wc -l
```

Exercice 8
Donnez un programme qui réalise l'équivalent de la commande Shell

```
% (echo '#' ; cat ~/.cshrc ; echo '# eof') | sed '/$^/d'
```