TD Communication par tubes

Exercice 1

Ecrire un programme C qui permet la communication entre un processus père et un processus fils en utilisant un tube ordinaire. Le père envoie une chaîne de caractères lue au clavier à son fils qui l'affiche à l'écran.

Exercice 2

Ecrire un programme C qui permet à un processus père de créer 2 processus fils. Le premier processus fils transmet au processus père la chaîne " Je suis le premier fils " en utilisant un tube ordinaire. Le second processus fils transmet au processus père le message " Je suis le second fils " en utilisant un tube ordinaire. Le processus père reçoit les 2 chaînes qui lui sont envoyées par ses fils et les affiche à l'écran.

Exercice 3

Soit le code ci-dessous. Décrire ce que fait ce code et indiquer les problèmes qui peuvent survenir pendant l'exécution de ce code, en considérant que la syntaxe est correcte et que la compilation se déroule bien.

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

int main (void) {

 int p1[2], p2[2] ;
 char * ch ;

 printf(« Debut du processus\n ») ;

 pipe(p1) ;
 pipe(p2) ;
 if (fork()==0) {
       printf("J'effectue mes actions ci-dessous\n");
       read(p1[0], ch, 1) ;
       write(p2[1], ch, 1);
       exit(0);}

 else {
       printf("J'effectue mes actions ci-dessous\n");
       read(p2[0], ch, 1);
       write(p1[1], ch, 1) ;
       exit(0) ;}
 }
```

Exercice 4

Soit le programme en page 3 ci-dessous.

1. Analysez le programme et dites ce qu'il fait.

2. Indiquez l'état des tables système ci-dessous aux points mentionnés (en gras) dans le programme en justifiant votre réponse. Vous supposerez que les points 1, 2 et 3 se déroulement dans cet ordre lors de l'exécution du programme :

   - table des i-nœuds,
   - table des fichiers ouverts,
   - table des descripteurs.

Remarque : la fonction `sizeof()` retourne la taille en octets de l'élément passé en paramètre (variable ou type)

Exercice 5 (optionnel, pour aller un peu plus loin)

Ecrire un programme en langage C équivalent à la commande shell `ps –e | wc –l`
(voir le man de la commande `dup` en annexe).

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

#define NB_ENTIERS 5

void fils (int p1[2], int p2[2])
{ int i, nombre, nb_lus ;

/* Point 2 : état des tables ici */

  close(p1[1]) ;
  close(p2[0]) ;

/* Point 3 : état des tables ici */

  nb_lus = read(p1[0], &nombre, sizeof(int));

  while (nb_lus == sizeof(int)){
      nombre = nombre*2 ;
      printf("Le resultat est :%d\n", nombre) ;
      write(p2[1], &nombre, sizeof(int)) ;
      nb_lus = read(p1[0], &nombre, sizeof(int)) ;
  }
  close(p1[0]);
  close(p2[1]);
  exit(0);
}

int main(void)
{
      int i, nombre, p1[2], p2[2];

      pipe(p1);
      pipe(p2);

/* Point 1 : état des tables ici */

      if (fork() == 0)
            fils(p1, p2);
      else {
            close(p1[0]);
            close(p2[1]);
            for ( i=0 ; i<NB_ENTIERS ; i++) {
                  printf("Entrez un entier\n");
                  scanf("%d", &nombre);
                  write(p1[1], &nombre, sizeof(int));
              }
            close(p1[1]);
            sleep(5);
            printf("Les resultats sont :\n");
            for(i=0; i<NB_ENTIERS; i++) {
                  read(p2[0], &nombre, sizeof(int));
                  printf("%d\n", nombre);
             }
            close(p2[0]);
            exit(0) ;
            }
}
```

Annexe

DUP(2) Linux Programmer's Manual DUP(2)

NAME

dup, dup2, dup3 - duplicate a file descriptor

SYNOPSIS

```
#include <unistd.h>

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

#define _GNU_SOURCE
#include <unistd.h>

int dup3(int oldfd, int newfd, int flags);
```

DESCRIPTION

These system calls create a copy of the file descriptor oldfd.

dup() uses the lowest-numbered unused descriptor for the new descriptor.

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary, but note the following:

* If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.

* If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

After a successful return from one of these system calls, the old and new file descriptors may be used interchangeably. They refer to the same open file description (see open(2)) and thus share file offset and file status flags; for example, if the file offset is modified by using lseek(2) on one of the descriptors, the offset is also changed for the other.

The two descriptors do not share file descriptor flags (the close-on-exec flag). The close-on-exec flag (FD_CLOEXEC; see fcntl(2)) for the duplicate descriptor is off.

dup3() is the same as dup2(), except that:

* The caller can force the close-on-exec flag to be set for the new file descriptor by specifying O_CLOEXEC in flags. See the description of the same flag in open(2) for reasons why this may be useful.

* If oldfd equals newfd, then dup3() fails with the error EINVAL.

RETURN VALUE

On success, these system calls return the new descriptor. On error, -1 is returned, and errno is set appropriately.

ERRORS

EBADF oldfd isn't an open file descriptor, or newfd is out of the allowed range for file descriptors.

EBUSY (Linux only) This may be returned by dup2() or dup3() during a race condition with open(2) and dup().

EINTR The dup2() or dup3() call was interrupted by a signal; see signal(7).

EINVAL (dup3()) flags contain an invalid value. Or, oldfd was equal to newfd.

EMFILE The process already has the maximum number of file descriptors open and tried to open a new one.

VERSIONS

dup3() was added to Linux in version 2.6.27; glibc support is available starting with version 2.9.

CONFORMING TO

dup(), dup2(): SVr4, 4.3BSD, POSIX.1-2001.

dup3() is Linux-specific.

NOTES

The error returned by dup2() is different from that returned by fcntl(..., F_DUPFD, ...) when newfd is out of range. On some systems dup2() also sometimes returns EINVAL like F_DUPFD.

If newfd was open, any errors that would have been reported at close(2) time are lost. A careful programmer will not use dup2() or dup3() without closing newfd first.

SEE ALSO

close(2), fcntl(2), open(2)

COLOPHON

This page is part of release 3.23 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

Linux 2008-10-09 DUP(2)