Exemples d'utilisation tubes

Un exemple de communication unidirectionnelle

Nous donnons un premier exemple de communication entre processus par l'intermédiaire d'un tube. Dans cet exemple, le processus fils écrit à destination de son père la chaîne de caractères « bonjour ». Les étapes du programme sont les suivantes :

1. Le processus père ouvre un tube en utilisant la fonction pipe().
2. Le processus père crée un fils en utilisant la fonction fork().
3. Les descripteurs en lecture et écriture du tube sont utilisables par les 2 processus. Chacun des deux processus ferme le descripteur qui lui est inutile : ainsi, le processus père ferme le descripteur en écriture et le processus fils ferme le descripteur en lecture.
4. Le fils envoie un message à son père.
5. Le père lit le message.

```
/*********************************************************************/
/*    Communication unidirectionnelle entre un père et son fils    */
/*********************************************************************/
main () {
int pip[2], status;
pid_t retour;
char chaine[7];


pipe(pip);
retour = fork();
if (retour == 0)
{ /* le fils écrit dans le tube */
        close (pip[0]); /* pas de lecture sur le tube */
        write (pip[1], "bonjour", 7);
        close (pip[1]); exit(0);
}
else
{ /* le père lit le tube */
        close (pip[1]); /* pas d'écriture sur le tube */
        read(pip[0], chaine, 7);
        close (pip[0]);
        wait(&status);
}
```

Un exemple de communication bidirectionnelle

L'exemple suivant illustre une communication bidirectionnelle. Comme le tube est un outil de communication unidirectionnelle, la réalisation d'une communication bidirectionnelle nécessite l'utilisation de deux tubes, chaque tube étant utilisé dans le sens inverse de l'autre.

```
/*********************************************************************/
/*    Communication bidirectionnelle entre un père et son fils    */
/*********************************************************************/
#include <stdio.h>

int pip1[2];       /*  descripteurs pipe 1  */
int pip2[2];       /*  descripteurs pipe 2  */
int status;

main()
{
```

```
int idfils;
char rep[7], mesg[5];

/* ouverture tubes */
if(pipe(pip1))
  {
  perror("pipe 1");
  exit(1);
  }
if(pipe(pip2))
  {
  perror("pipe 2");
  exit(2);
  }
/* création processus */
if((idfils=fork())==-1)
  {
  perror("fork");
  exit(3);
  }
if(idfils) {
        /*le premier tube sert dans le sens père vers fils
          il est fermé en lecture */
       close(pip1[0]);
       /*le second tube sert dans le sens fils vers père
          il est fermé en écriture*/
       close(pip2[1]);
       /* on envoie un message au fils par le tube 1*/
       if(write(pip1[1],"hello",5)!=5)
         {
           fprintf(stderr,"père: erreur en écriture\n");
           exit(4);
         }
        /* on attend la réponse du fils par le tube 2 */
        if(read(pip2[0],rep,7)!=7)
          {
           fprintf(stderr,"fils: erreur lecture\n");
           exit(5);
          }
        printf("message du fils: %s\n",rep);
        wait(&status);
else   {
        /*fermeture du tube 1 en écriture */
        close(pip1[1]);
        /* fermeture du tube 2 en lecture */
        close(pip2[0]);
        /* attente d'un message du père */
        if(read(pip1[0],mesg,5)!=5)
        {
           fprintf(stderr,"fils: erreur lecture\n");
           exit(6);
         }
        printf("la chaine reçue par le fils est: %s\n",mesg);
        /* envoi d'un message au père */
        if(write(pip2[1],"bonjour",7)!=7)
```

```
        {
          fprintf(stderr,"fils: erreur ecriture\n");
          exit(7);
        }
        exit(0)
      }
}
```

Exemples d'utilisation des tubes nommés

Communication unidirectionnelle

Ce premier exemple illustre une communication unidirectionnelle entre deux processus. Un premier processus crée un tube nommé appelé fictub, puis écrit la chaîne «0123456789» dans ce tube. L'autre processus lit la chaîne et l'affiche. À l'issue de l'exécution de ces deux processus, la commande ls –l montre le tube nommé fictub.

```
/*********************************************************************/
/*            Processus écrivain sur le tube nommé                   */
/*********************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main ()
{
mode_t mode;
int tub;
mode = S_IRUSR | S_IWUSR;

/*création du tube nommé */
mkfifo ("fictub", mode);

/* ouverture du tube */
tub = open ("fictub", O_WRONLY);

/* écriture dans le tube */
write (tub, "0123456789", 10);

/* fermeture du tube */
close(tub);
}
/*********************************************************************/
/*            Processus lecteur sur le tube nommé                    */
/*********************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main ()
{
char zone[11];
```

```
int tub;

/* ouverture du tube */
tub = open ("fictub", O_RDONLY);

/* lecture dans le tube */
read (tub, zone, 10);
zone[10] = 0;

printf ("processus lecteur du tube fictub: j'ai lu %s", zone);

/* fermeture du tube */
close(tub);
}
```

Communication bidirectionnelle

Ce second exemple illustre une communication bidirectionnelle à travers deux tubes nommés tube1 et tube2. Un processus client envoie deux nombres entiers à un processus serveur au travers du premier tube tube1. Le processus serveur effectue l'addition de ces deux nombres et renvoie le résultat au processus client à travers le tube tube2. Le processus serveur a lui-même créé les deux tubes nommés.

Les données véhiculées dans les tubes étant de type chaîne de caractères, les entiers doivent être convertis dans ce format au moment de l'envoi et reconvertis dans l'autre sens à la réception.

```
/*********************************************************************/
/*                    Processus client                               */
/*********************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main ()
{
int tub1, tub2;
int nb1, nb2, res;
char ent[2];

/* ouverture du tube tube1 en écriture  */
tub1 = open ("tube1", O_WRONLY);
/* ouverture du tube tube2 en lecture */
tub2 = open ("tube2", O_RDONLY);

printf ("Donnez moi deux nombres à additionner:");
scanf ("%d %d", &nb1, &nb2);

sprintf (ent, "%d", nb1);
/* écriture dans le tube */
write (tub1, ent, 2);

sprintf (ent, "%d", nb2);
/* écriture dans le tube */
write (tub1, ent, 2);

/* lecture du résultat */
read (tub2,ent,2);
```

```
res = atoi(ent);

printf ("le résultat reçu est: %d", res);

/* fermeture et destruction des tubes */
close(tub1);
close(tub2);
unlink(tub1);
unlink(tub2);
}
/*********************************************************************/
/*                    Processus serveur                    */
/*********************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main ()
{
char ent[2];
int tub1, tub2;
int nb1, nb2,res;
mode_t mode;

mode = S_IRUSR | S_IWUSR;

/*création de deux tubes nommés */
mkfifo ("tube1", mode);
mkfifo ("tube2", mode);


/* ouverture du tube 1 en lecture */
tub1 = open ("tube1", O_RDONLY);
/*ouverture du tube 2 en ecriture */
tub2 = open ("tube2", O_WRONLY);


/* lecture dans le tube */
read (tub1, ent, 2);
nb1 = atoi(ent);
read (tub1, ent, 2);
nb2 = atoi(ent);


res = nb1 + nb2;
sprintf (ent, "%d", res);

write (tub2,ent,2);

/* fermeture du tube */
close(tub1);
close(tub2);
}
```

Un exemple d'utilisation des files de messages

Nous donnons ici un exemple de communication clients-serveur par le biais d'une file de messages. Comme pour l'exemple concernant les tubes nommés, le serveur effectue l'addition de deux nombres entiers envoyés par un client et renvoie le résultat. La communication s'effectue à présent par l'intermédiaire d'une file de messages unique de clé égale à 314. Cette file de message contient donc tout à la fois les requêtes en provenance des clients et les réponses du serveur.

Dans cette application, le serveur doit uniquement consommer les requêtes depuis la file de messages. Les clients de leur côté doivent uniquement lire la réponse qui les concerne.

Pour parvenir à ce schéma, on utilise le champ type dans la structure des messages véhiculés par la file de messages. Ainsi, le serveur désigne le client auquel s'adresse la réponse en remplissant le champ type de la réponse avec la valeur du pid du client. Ce pid est indiqué par le client au niveau de sa requête. Le client quant à lui désigne le serveur comme destinataire de la requête en initialisant le champ type de sa requête avec une valeur positive qui ne peut pas correspondre à un pid de processus utilisateur, par exemple la valeur 1 (cette valeur 1 correspond obligatoirement au processus init).

Ainsi une requête et une réponse ont le format suivant :

```
struct requete {
        long letype; /* prend la valeur 1 */
        int nb1; /* premier membre de l'addition */
        int nb2; /* second membre de l'addition */
        pid_t mon_pid;  /*le client indique ici son pid */
        };
struct reponse {
        long letype; /* prend la valeur du pid du client */
        int res; /* le résultat de l'addition */
        };
/*********************************************************************/
/*          Processus serveur: crée la MSQ              */
/*   et additionne les deux nombres reçus dans chaque message   */
/*********************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define CLE 314

struct requete {
        long letype;
        int nb1;
        int nb2;
        pid_t mon_pid;
       };
struct reponse {
        long letype;
        int res;
        };

main()
{
int msqid,l;
struct requete la_requete;
struct reponse la_reponse;
/*  allocation  MSQ */
```

```
if((msqid=msgget((key_t)CLE,0750|IPC_CREAT|IPC_EXCL))==-1)
  {
   perror("msgget");
   exit(1);
  }

while (1)
{
/* lecture d'une requête */
if((l=msgrcv(msqid,&la_requete,sizeof(struct requete)-4,1,0))==-1)
  {
   perror("msgrcv");
   exit(2);
  }
la_reponse.res = la_requete.nb1 + la_requete.nb2;
la_reponse.letype=la_requete.mon_pid;
/* type associé au message; le pid du client  */
if(msgsnd(msqid,&la_reponse,sizeof(struct reponse) - 4,0)==-1)
  {
   perror("msgsnd");
   exit(2);
  }
}
exit(0);
}
/*********************************************************************/
/*      Processus client: envoi de deux nombres à additionner      */
/*********************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define CLE 314

struct requete {
              long letype;
              int nb1;
              int nb2;
              pid_t mon_pid;
             };
struct reponse {
              long letype;
              int res;
              };
main()
{
int msqid, l, nb1, nb2;
struct requete la_requete;
struct reponse la_reponse;

/* récupération du msqid */
if((msqid=msgget((key_t)CLE,0))<0)
```

```
  {
   perror("msgget");
   exit(1);
  }

/* préparation de la requête et envoi */
printf ("Donnez moi deux nombres à additionner:");
scanf ("%d %d", &nb1, &nb2);

la_requete.letype = 1;
la_requete.nb1 = nb1;
la_requete.nb2 = nb2;
la_requete.mon_pid = getpid();

if(msgsnd(msqid,&la_requete,sizeof(struct requete)-4,0)==-1)
  {
   perror("msgsnd");
   exit(2);
  }

/* réception de la réponse */
if((l=msgrcv(msqid,&la_reponse,sizeof(struct reponse)-4,getpid(),0)==-1))
  {
   perror("msgrcv");
   exit(2);
  }

printf ("le resultat reçu est: %d", la_reponse.res);
exit(0);
}
```