Compléments sur les tubes

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

28 novembre 2005

Plan

dup

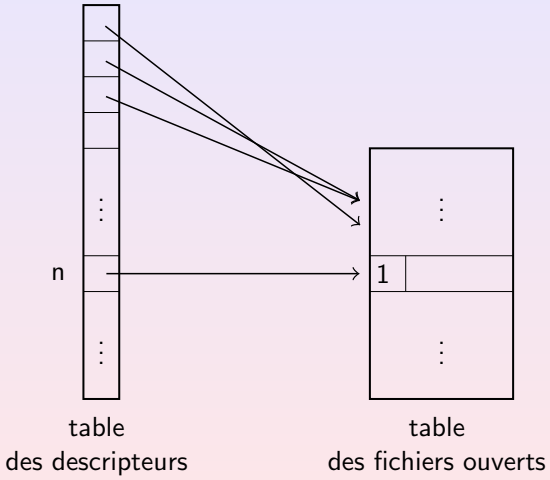
```
#include <unistd.h>
int dup(int desc);
```

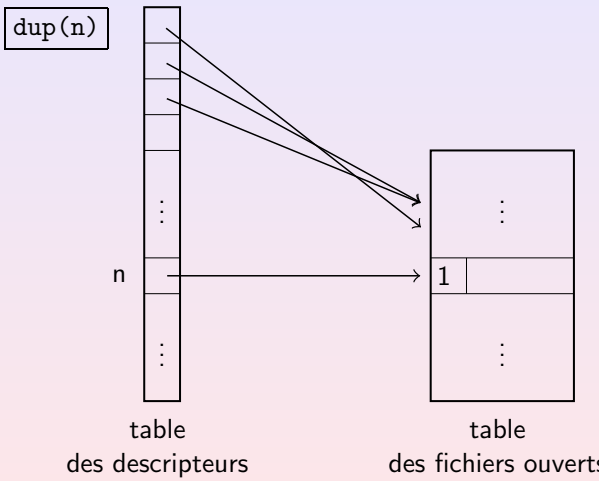
desc : le descripteur qu'on veut dupliquer

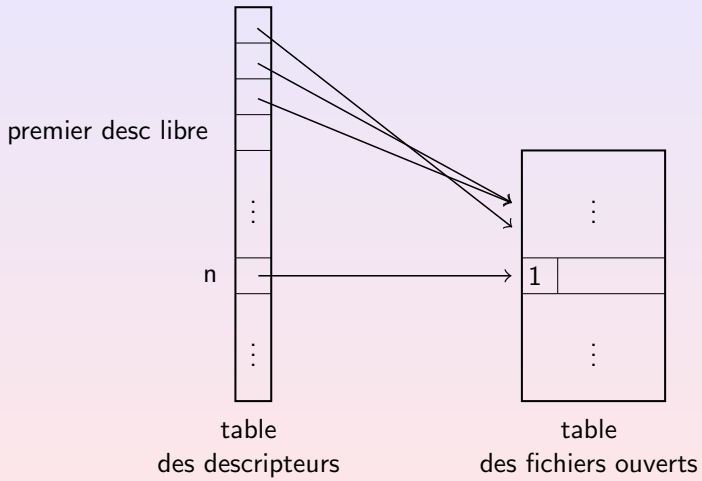
Retourne : un nouveau descripteur (-1 si erreur)

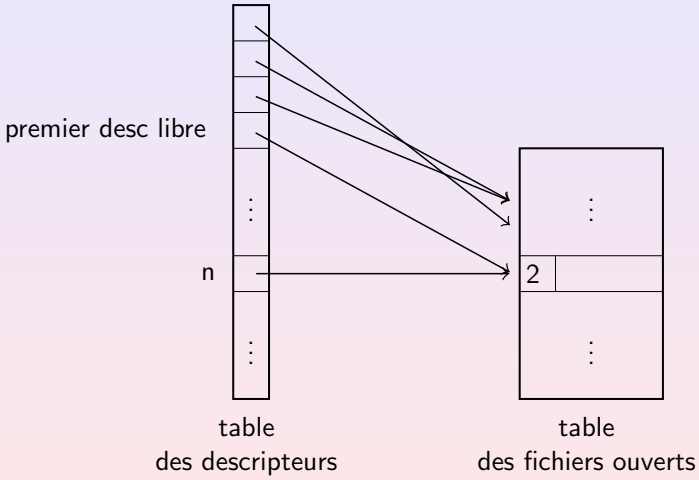
Sommaire : duplication d'un descripteur

Remarques : le nouveau descripteur est le premier disponible



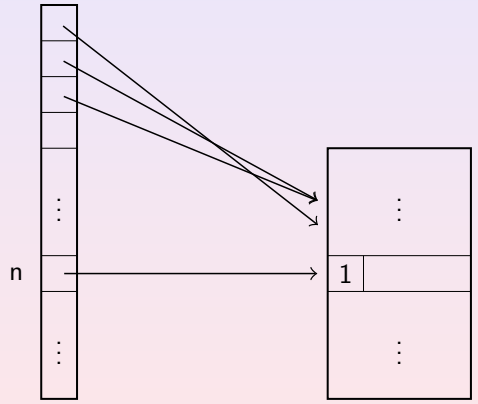




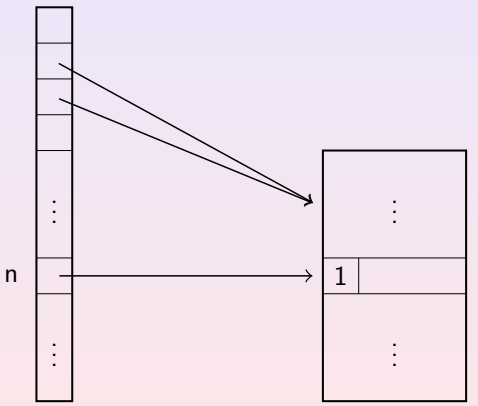


Redirection entrée

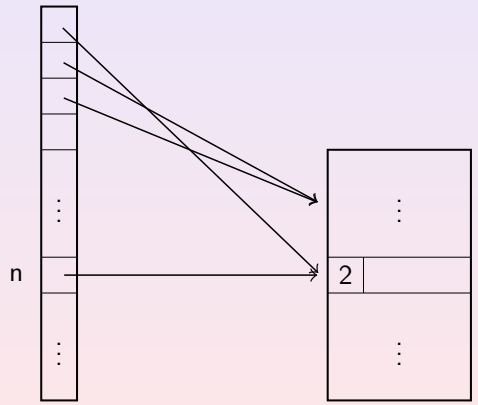
```
{
   ...
   close(STDIN_FILENO);
   dup(n);
   close(n);
   ...
}
```



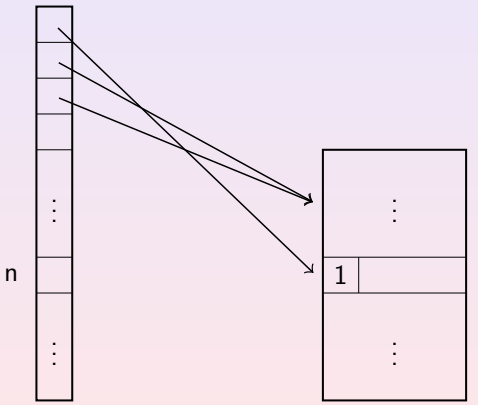

```
close(SDTIN_FILENO);
```




```
close(SDTIN_FILENO); dup(n);
```




```
close(SDTIN_FILENO); dup(n); close(n);
```



Programme : exempledup.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void fils()
6  {
7    printf("%s","abcde\n");
8    exit(EXIT_SUCCESS);
9  }
10
11 void pere()
12 {
13   char tampon[100];
14   fgets(tampon, sizeof(tampon),stdin);
15   printf("%s",tampon);
16   exit(EXIT_SUCCESS);
17 }
```


Programme : exempledup.c (II)

```
20 int main(void)
21 {
22   int tube[2],desc;
23   if(pipe(tube) == -1) exit(EXIT_FAILURE);
24
25   switch(fork())
26     {
27     case -1: exit(EXIT_FAILURE);
28     case 0 : close(tube[0]);
29       /* Redirection stdout vers tube[1] */
30       close(STDOUT_FILENO);
31       desc=dup(tube[1]);
32       fprintf(stderr,"[Fils] Nouveau descripteur : %d\n", desc);
33       close(tube[1]);
34       /* Fin redirection */
35       fils();
36     default : close(tube[1]);
37       /* Redirection stdin vers tube[1] */
38       close(STDIN_FILENO);
39       desc = dup(tube[0]);
40       fprintf(stderr,"[Père] Nouveau descripteur : %d\n", desc);
41       close(tube[0]);
42       /* Fin redirection */
43       pere();
44     }
45   exit(EXIT_FAILURE);
46 }
```


dup2

```
#include <unistd.h>
int dup2(int desc1, int desc2);
```

desc1 : le descripteur à dupliquer

desc2 : le descripteur qu'on aimerait utiliser. S'il est déjà utilisé, le système opère un fermeture préliminaire : `close(desc2)`

Retourne : le descripteur dupliqué (-1 si erreur)

Programme : exempledup2.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void fils()
6  {
7    printf("%s","abcde\n");
8    exit(EXIT_SUCCESS);
9  }
10
11 void pere()
12 {
13   char tampon[100];
14   fgets(tampon, sizeof(tampon),stdin);
15   printf("%s",tampon);
16   exit(EXIT_SUCCESS);
17 }
```

Programme : exempledup2.c (II)

```
20 int main(void)
21 {
22   int tube[2],desc;
23   if(pipe(tube) == -1) exit(EXIT_FAILURE);
24
25   switch(fork()){
26   case -1: exit(EXIT_FAILURE);
27   case 0 : close(tube[0]);
28     /* Redirection stdout vers tube[1] */
29     desc=dup2(tube[1],STDOUT_FILENO);
30     fprintf(stderr,"[Fils] Nouveau descripteur : %d\n", desc);
31     close(tube[1]);
32     /* Fin redirection */
33     fils();
34   default : close(tube[1]);
35     /* Redirection stdin vers tube[1] */
36     desc = dup2(tube[0],STDIN_FILENO);
37     fprintf(stderr,"[Père] Nouveau descripteur : %d\n", desc);
38     close(tube[0]);
39     /* Fin redirection */
40     pere();
41   }
42   exit(EXIT_FAILURE);
43 }
```


Programme : lstubewctirel

```
1  #include <unistd.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    int tube[2];
7    if( pipe(tube) == -1 ) exit(1);
8
9    switch(fork()){
10   case -1: exit(1);
11   case 0 :
12     close(tube[0]); /* Le fils ne lit pas du tube */
13     if( dup2(tube[1],STDOUT_FILENO) == -1) exit(1);
14     close(tube[1]);
15     execlp("ls","Faire la liste",NULL);
16     break;
17   default :
18     close(tube[1]); /* Le pere n'écrit pas dans le tube */
19     if(dup2(tube[0],STDIN_FILENO) == -1) exit(1);
20     close(tube[0]);
21     execlp("wc","Compter lignes", "-l",NULL);
22   }
23   exit(1);
24 }
```


mkfifo

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

ref : nom/chemin à donner au tube
mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-m mode] fichier`

Programme : premier exemple

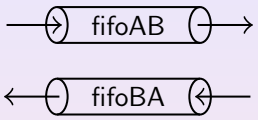
```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8    mode_t mode = S_IRWXU | S_IRWXG | S_IRWXO ;
9
10   if(mkfifo("tube",mode) == -1){
11     perror("mkfifo");
12     exit(EXIT_FAILURE);
13   }
14   /* nécessaire à cause de la masque de création */
15   if(chmod("tube",mode) == -1){
16     perror("chmod");
17     exit(EXIT_FAILURE);
18   }
19   exit(EXIT_SUCCESS);
20 }
```

Ouverture d'un tube

Bloquante :

- `open("tube",O_RDONLY)` :
  bloque si #écrivains = 0
- `open("tube",O_WRONLY)` :
  bloque si #lecteurs = 0

Interbloquage



Code bloquant :

```
/* proc A */                        /* proc B */
    ...                                 ...
d_BA = open("tubeBA",O_RDONLY);   d_AB = open("tubeAB",O_RDONLY);
d_AB = open("tubeAB",O_WRONLY);   d_BA = open("tubeBA",O_WRONLY);
```

Correction :

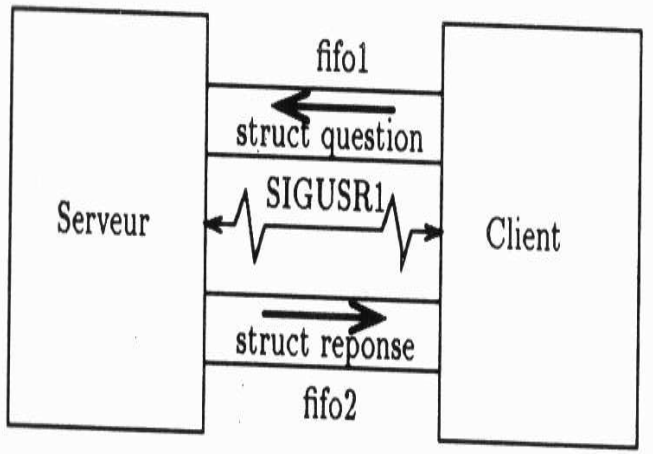
```
/* proc A */                        /* proc B */
    ...                                 ...
d_BA = open("tubeBA",O_RDONLY);   d_BA = open("tubeBA",O_WRONLY);
d_AB = open("tubeAB",O_WRONLY);   d_AB = open("tubeAB",O_RDONLY);
```


Ouverture non bloquante

Obtenue à l'aide des indicateurs `O_NONBLOCK`, `O_NDELAY` :

- `open("tube", O_RDONLY | O_NONBLOCK)` :
  reussit toujours même si #ecrivains = 0
- `open("tube", O_WRONLY | O_NONBLOCK)` :
  renovoit -1 si #lecteurs = 0

Le modèle Serveur-Client (Rifflet 1999)



unlink

```
#include <unistd.h>
int unlink(char * lien);
```

lien : le lien à effacer

Retourne : 0/-1

Remarques : si

- le nombre de lien à l'i-noeud est 0,
- aucun processus utilise cet i-noeud,

alors l'espace sur disque du fichier est libéré.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

Programme : servclififo.h

```
1  #ifndef S_C_FIFO
2  #define S_C_FIFO
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <sys/fcntl.h>
10 #include <signal.h>
11
12 #define NMAX 20
13 #define QUESTION "fifo1"
14 #define REPONSE "fifo2"
15
16 struct question{
17   int pid_client;
18   int question;
19 };
20
21 struct reponse{
22   int pid_serveur;
23   int reponse[NMAX];
24 };
25
26 extern void hand_reveil(int sig);
27 extern void init_action(struct sigaction *action,void (*handler)(int));
28
29 #endif /* S_C_FIFO */
```


Programme : serveurfifo.c

```
1  #include "serv_cli_fifo.h"
2
3  void fin_serveur(int sig)
4  {
5    unlink(QUESTION);
6    unlink(REPONSE);
7    exit(2);
8  }
9
10 int main(void)
11 {
12   struct sigaction action;
13   int d_question, d_reponse; /* Descripteurs sur les tubes
14   int ind,sig;
15   sigset_t ens, ens_vide;
16   struct question question;
17   struct reponse reponse;
18   mode_t mode = S_IRUSR|S_IRGRP|S_IROTH| S_IWUSR|S_IWGRP|S
```

Programme : serveurfifo.c (II)

```
20    if(mkfifo(QUESTION,mode) == -1 || mkfifo(REPONSE,mode) == -1)
21      {
22        fprintf(stderr,"Creation des tubes impossible\n");
23        exit(2);
24      };
25    d_question = open(QUESTION,O_RDONLY);
26    d_reponse = open(REPONSE,O_WRONLY);
27
28    /* Tous les signaux mettront fin au serveur,
29       ... de façon gentile  */
30    for(sig=1;sig < NSIG; sig++)
31      signal(sig,fin_serveur);
32    /* Sauf SIGUSR1 qui nous reveillerai */
33    init_action(&action,hand_reveil);
34    sigaction(SIGUSR1,&action,NULL);
35    /* On bloque SIGUSR1 */
36    sigemptyset(&ens);
37    sigaddset(&ens,SIGUSR1);
38    sigprocmask(SIG_SETMASK,&ens,NULL);
39    sigemptyset(&ens_vide); /* Ensemble vide de signaux */
40
41    srand(getpid());
42    reponse.pid_serveur=getpid();
```


Programme : serveurfifo.c (III)

```
44    /* Boucle principale */
45    while(1){
46      if(read(d_question,&question, sizeof(struct question)) <= 0 )
47        {/* Aucune attente attive :
48            on s'endort en attente de quelque écrivain */
49          close(d_question);
50          d_question = open(QUESTION,O_RDONLY);
51          continue;
52        }
53
54      for(ind=0; ind < question.question; ind++)
55        reponse.reponse[ind]=rand()%10;
56
57      if(write(d_reponse,&reponse,sizeof(struct reponse)) == -1)
58        {
59          perror("write");
60          fin_serveur(SIGUSR2);
61        }
62      kill(question.pid_client,SIGUSR1);
63      /* En attente de tous les signaux */
64      sigsuspend(&ens_vide);
65    }
66  }
```


Programme : clientfifo.c

```
1  #include "serv_cli_fifo.h"
2
3  int main(void){
4    struct sigaction action;
5    int d_question, d_reponse;
6    int ind;
7    sigset_t ens,ens_vide;
8    struct question question;
9    struct reponse reponse;
10
11   d_question = open(QUESTION,O_WRONLY);
12   d_reponse = open(REPONSE,O_RDONLY);
13   if(d_reponse == -1 || d_question == -1)
14     {
15       fprintf(stderr,"Ouverture des tubes impossible\n");
16       exit(2);
17     };
18
19   /* Signaux */
20   init_action(&action,hand_reveil);
21   sigaction(SIGUSR1,&action,NULL);
22   sigemptyset(&ens_vide);
23   sigemptyset(&ens);
24   sigaddset(&ens,SIGUSR1);
25   sigprocmask(SIG_SETMASK,&ens,NULL);
```


Programme : clientfifo.c (II)

```
27   /*  Initialisation */
28   srand(getpid());
29   question.pid_client=getpid();
30   question.question= 1 + rand()%NMAX;
31
32   /* Envoyer message */
33   if(write(d_question,&question,sizeof(struct question)) == -1){
34     perror("write");
35     exit(2);
36   }
37   sigsuspend(&ens_vide);
38
39   /* Lire reponse */
40   if(read(d_reponse,&reponse, sizeof(struct reponse) )<= 0 )
41     {
42       fprintf(stderr,"Problème su read\n");
43       exit(2);
44     };
45   /* ACK */
46   kill(reponse.pid_serveur,SIGUSR1);
47
48   printf("Client : %d nombres reçus :\n",question.question);
49   for(ind=0; ind < question.question; ind++)
50     printf("%d ",reponse.reponse[ind]);
51   printf("\n");
52   exit(0);
53 }
```

Programme : outils.c

```
 1 #include "serv_cli_fifo.h"
 2
 3 void hand_reveil(int sig)
 4 {
 5   return;
 6 }
 7
 8 void init_action(struct sigaction *action,
 9                   void (*handler)(int))
10 {
11   sigemptyset(&action->sa_mask);
12   action->sa_flags=0;
13   action->sa_handler=handler;
14 }
```