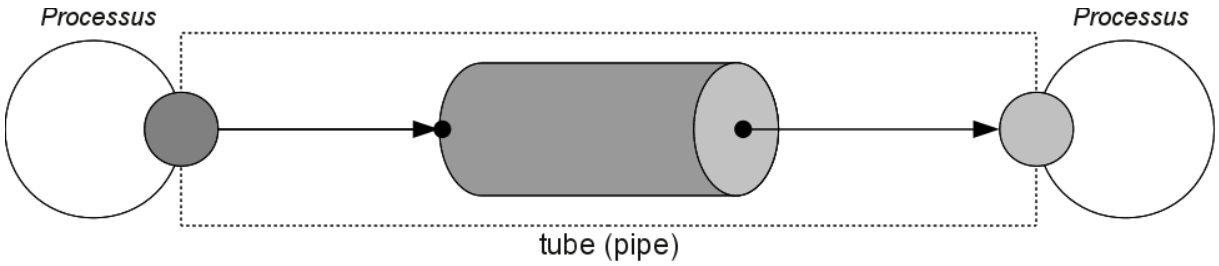
III. TUBES

Un tube est un canal unidirectionnel qui fonctionne en mode flot d'octets. Mécanisme d'échange bien adapté à l'envoi de caractères. Les tubes sont utilisés pour la communication entre processus.

1. Les Tubes Anonymes



Un tube est implémenté, sous Unix, comme un fichier. Ainsi, la plupart des fonctions sur les fichiers sont utilisables sur les tubes. Cependant, un tube n'a pas de nom physique. Il n'existe donc que pendant le temps d'exécution du processus.

Propriétés :

- Un tube n'a pas de nom physique
- Il n'existe que pendant le temps d'exécution du processus
- Les processus communicant avec un tube doivent avoir un lien de parenté
- Un tube est une file de type FIFO
- La communication est unidirectionnelle
- Ce qui est lu quitte définitivement le tube et ne peut être relu

1.1 Création d'un tube

La création d'un tube se fait à l'aide de la fonction pipe :

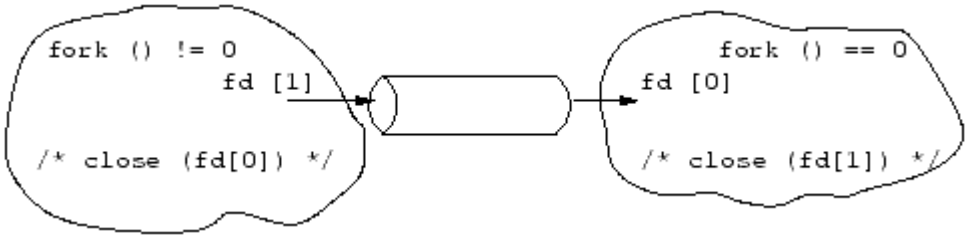
```
#include <unistd.h>

        int pipe(int *fd)
```

```
int p[2]; // déclaration de deux descripteurs de type int
pipe(p);  // appel de la fonction pipe
```

- Retourne automatiquement deux descripteurs en cas de création
  - p[0] est le descripteur par lequel on peut lire dans le tube
  - p[1] est le descripteur par lequel on peut écrire dans le tube
- -1 en cas d'erreur

```
if  (pipe(p)==-1)
{
 perror("pipe");
        exit(0);
}
```

Remarque :
Si le tube a été créé par un processus avant la création de ses fils, tous ces fils héritent alors des descripteurs du père



PÈRE **FILS**

Utilisation typique d'un tube : communication entre processus avec lien de parenté

- Un tube est créé par un processus père
- Le processus père crée un processus fils
- Le père et le fils partagent et communiquent avec le tube
- Le processus fils hérite des descripteurs de lecture et d'écriture dans le tube.

Les descripteurs retournés par l'appel système *pipe()* fonctionnent de la même manière que les descripteurs de fichiers.

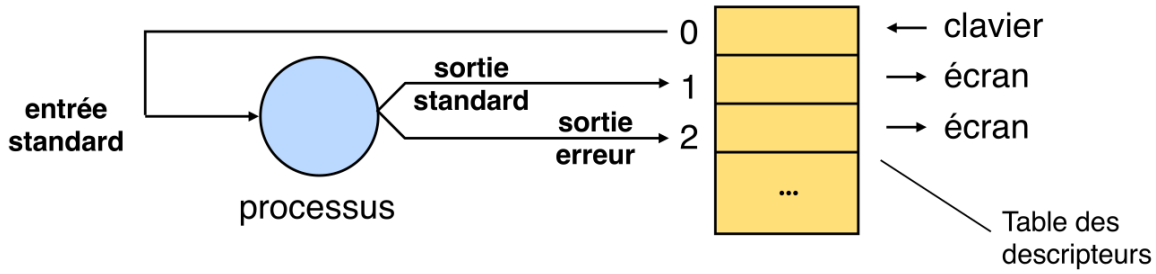
Rappel sur les descripteurs du système de gestion de fichier

Chaque fichier ouvert est représenté par un descripteur de fichier

Tout processus possède trois descripteurs de fichiers ouverts par défaut :

- 0 : descripteur de l'entrée standard (clavier)
- 1 : descripteur de la sortie standard des résultats (écran)
- 2 : descripteur de sortie standard des erreurs (écran)

et une table de descripteurs de fichiers ouverts



Tous les processus partagent une table de fichiers ouverts

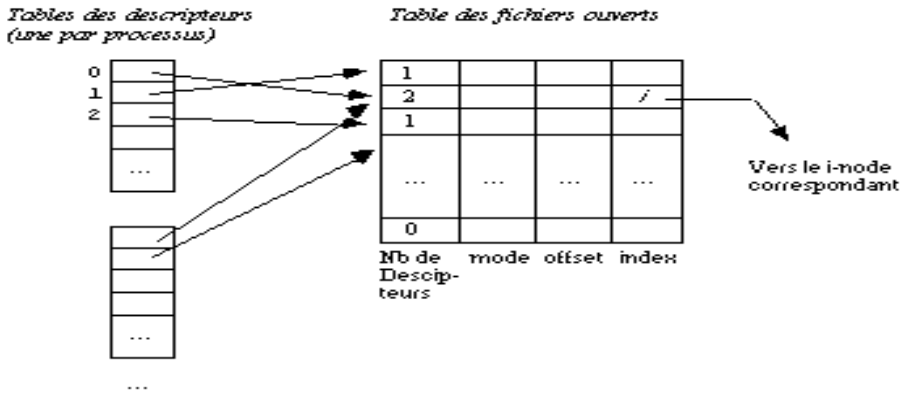


Table des fichiers ouverts

- Le nombre total de descripteurs correspondants au fichier disque ouvert appelé aussi compteur de référence
- Le mode d'ouverture (R, W, RW)
- Position (dite *offset*) courante en lecture/écriture
- Pointeur vers un *I_Node* (*Index_Node*), Contient des informations sur le fichier:

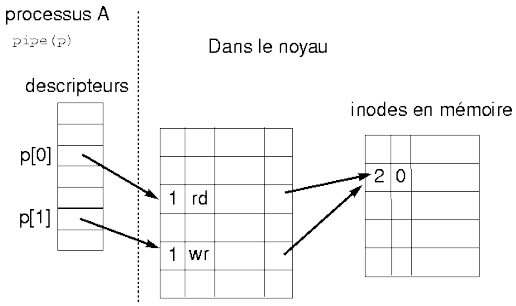
(Droits d'accèss associées, Taille, Position sur le support physique, …..)

Pipe, fork() et descripteurs

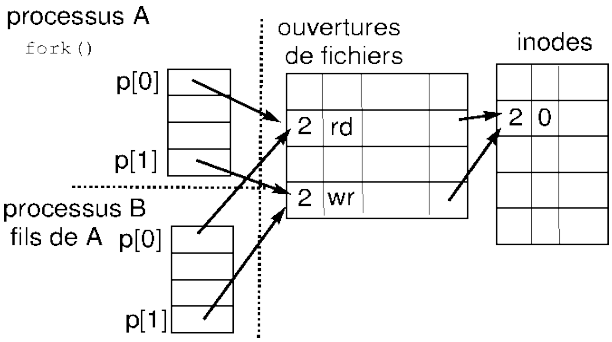
Le programme suivant permet à un processus (père) de créer un tube de communication et un processus fils.

```
#include <stdio.h>
#include <unistd.h>
main()
{
        int pid, p[2];
        pipe(p); /* création des descripteurs p[1], p[0] */
        if  (pipe(p)==-1)
                {
                perror("pipe");
                exit(0);
        }
        pid=fork() ;
        if (pid ==0) { /* fils */

          /* Héritage de p[1], p[0]  par duplication */
        else {
           /* pere */

         }
 }
```

La figure suivante montre ce qui se passe au niveau noyau après les appels système pipe() et fork()



Avant le fork()



Pipe() et fork()

Après Création du processus fils avec fork()

- Le nouveau processus a une copie identique de la mémoire et des descripteurs de fichiers
- Il hérite donc des descripteurs de lecture et d'écriture du tube créés par le père

Un seul processus accède en lecture et un seul accède en écriture (les descripteurs non utilisés doivent être fermés)

1.2 Écriture/Lecture dans un tube

Dans le cas général les appels systèmes read() et write() ont la syntaxe suivante :

*ssize_t read(int fd, void *buf, size_t count);*

read() lit jusqu'à *count* octets depuis le fichier associé au descripteur *fd* dans le tampon pointé par *buf*.

*ssize_t write(int fd, const void *buf, size_t count);*

write() écrit jusqu'à *count* octets dans le fichier associé au descripteur *fd* depuis le tampon pointé par *buf*.

Pour un tube de communication p crée par pipe(p) :

```
char buf[TAILLE_BUF] ;
Nb_écrit = write(p[1], buf, n);
```

L'appel système *write* demande l'écriture dans le tube de descripteur *p[1]* des n caractères accessibles par l'adresse *buf* dans l'espace d'adressage du processus.
L'appel système write() retourne le nombre de caractères réellement écrit *Nb_écrit*, -1 en cas d'erreurs.

```
char buf[TAILLE_BUF]
Nb_lu = read(p[0], buf ,n)
```

L'appel système read() lit *n* octets du tube *p* référencé par le descripteur *p[0]* et transfert le contenu lu dans *buf*.
L'appel système read() retourne le nombre de caractères réellement écrit *Nb_lu*, -1 en cas d'erreurs.

***TAILLE_BUF** doit être <= **PIPE_BUF** (limite définie dans <limits.h>.*

1.3 Fermeture d'un flux associé à un descripteur de tube

```
close(P[0]) ;
……
……
close(P[1]) ;
```

La commande close permet de fermer le descripteur d'un tube. L'accès à la partie entrante (resp. sortante) du tube se fait par la commande *close(p[1]) (resp. close(p[0]))*. Il est fortement conseillé de fermer les descripteurs non utilisés au niveau d'un processus, si celui ci ne compte pas les utilisés.

Exemple : Ecrire le programme C sous unix qui permet à un processus père de créer un processus Fils. Le fils récupère une chaine de caractère l'écrit dans un tube. Le père lit du tube caractère par caractère et l'affiche en majuscule.

Appels Systemes : fork(), pipe(), read(), write(), close();

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


main(int argc, char *argv[] ){
char c;
int pid,i, tube[2];
if(pipe(tube)==-1){ perror("pipe");
                    exit(0); }
pid=fork();
if(pid==-1){perror("fork");
          exit(0);   }
if(pid==0){
          close(tube[0]);  // On ferme le descripteur de lecture pour le fils
          write(tube[1],argv[1],strlen(argv[1])*sizeof(char));
          close(tube[1]);
          }
else{

     close(tube[1]);    // On ferme le descripteur d'écriture pour le père
     for(i=0;i< strlen(argv[1]); i++)
        {
     read(tube[0],&c, sizeof(char));
      printf("%c",toupper(c));
         }
       close(tube[0]);

   }
 printf("\n");
}
```

3. Communication et synchronisation en utilisant read() et write()

Un tube est un tampon qui sert de moyen de communication entre un processus Ecrivain et un processus Lecteur. Les tubes sont des Mécanisme de communication entre processus avec lien de parenté Fils/Père.

Une synchronisation type Ecrivain / Lecteur est nécessaire pour réaliser la communication. Un lecteur peut parfois attendre qu'il y est quelque chose d'écrite avant de lire, et un écrivain peut attendre qu'il y ait de la place dans le tube avant de pouvoir l'écriture. Les appels systèmes read() et write() sont utilisés pour réaliser de tels synchronisations.

Comportement de la fonction read()

Communication et synchronisation en utilisant read() et write()

ssize_t read(int fd, void *buf, size_t nbyte);

Si Tube non vide (contient tbyte caractères)
alors
Extrait min(tbyte, nbyte) caractères
Ecrit à l'adresse buf
retourne la valeur lue *nb_lu*= min(tbyte, nbyte)
Sinon (Tube vide)
Si *nombre d'écrivains = 0 (la fin est atteinte)*
alors
aucun caractère n'est lu et la primitive renvoie la valeur *nb_lu=0;*
Fsi
Si *nombre d'écrivains != 0*
alors
Si *lecture bloquante*
alors
processus bloqué jusqu'à écriture par un écrivain;
Fsi;
Si lecture non bloquante alors
le retour est immédiat et la valeur de retour est −1 et errno=EAGAIN
Fsi;
Fsi
Fsi

Réveil d'un processus bloqué en lecture sur un tube vide

- si un écrivain réalise une écriture, read() retourne une valeur non nulle
- si le dernier écrivain ferme son descripteur d'accès au tube par close() alors read() retourne 0

Rendre read() non-bloquante :

Une fois un fichier ouvert, associé à un descripteur de fichier, on peut gérer certaines de ses caractéristiques par l'appel système fcntl()

#include <fcntl.h>

int fcntl(int fd, int cmd, ... [arg]);

Fd : Descripteur du fichier
Cmd : Opération à effectuer
-F_GETFL : Obtenir les options du fichier ouvert
-F_SETFL : Modifier les options du fichier ouvert
Arg : Argument optionnel, dépendant de l'opération à effectuer

Exemple : *fcntl(p[0], F_SETFL, O_NONBLOCK);*

Dans ce cas read() dans un tube vide (avec écrivains potentiels) ne bloque pas et renvoie -1.

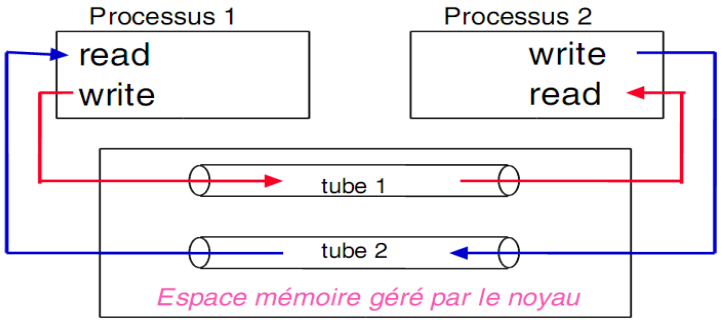
Comportement de la fonction write()

ssize_t write(int fd, const void *buf, size_t nbyte)

Si nombre de lecteurs dans le tube = 0
alors Erreur, le processus écrivain se termine par le signal SIGPIPE
Si le nombre de lecteurs != 0
alors
si écriture bloquante
alors
Ecriture atomique,
bloquant tant qu'il n'y a pas la place pour écrire les nbyte caractères
sinon /* non_bloquant */
si nbyte **> PIPE_BUF**
alors le retour est un nombre < nbyte **éventuellement -1**
fsi
si nbyte **<= PIPE_BUF et nbre emplacements libres >=** nbyte
alors une écriture atomique est réalisé;
fsi
si nbyte **<= PIPE_BUF et nbre emplacements libres <** nbyte
alors le retour est immédiat *sans écriture avec la valeur de retour*
0 ou -1
fsi
fsi

4. Communication bidirectionnelle

Deux processus avec lien de parenté peuvent réaliser une communication bidirectionnelle en utilisant deux tubes (un tube communication seulement dans un seul sens).



Les synchronisations peuvent être bien gérées en utilisant les appels systèmes read() et write(). Cependant, il faut faire très attention au situation d'interblocage illustrée, par l'exemple, deux processus Fils/ Père qui communiquent dans les deux sens à travers deux tubes 1 et 2.

PROCESSUS 1 : Fils		**PROCESSUS 2 : Père**
read(tube 2[0], buf, n);		read(tube 1 [0], buf, n);
...	...	
write(tube 1[1], buf, n);		write(tube 2 [1], buf, n);

Les deux processus se bloquent sur le read() alors que les deux tubes sont vides au départ.

5. Redirection des Entrée et Sorties standard

La fonction système *dup* permet de rediriger les entrées/sorties standards dans un tube.

Les descripteurs suivants sont ouverts automatiquement pour chaque processus :
0 : entrée standard (clavier)
1 : sortie standard des résultats (écran)
2 : sortie standard des erreurs (écran)

Il est possible d'effectuer des redirections entre les entrées/sorties standards et un tube.

2.1 Redirection de l'entrée standard (0) vers le tube p :

```
close(0);
close(p[1]);
dup(p[0]);
close(p[0]);
```

Fermeture de l'entrée standard
Interdiction de l'écriture
On duplique l'entrée correspondant au descripteur de lecture du tube dans la table des descripteurs (le descripteur de l'entrée standard est libre on le relie au tube pour lecture)

2.2 Redirection de la sortie standard (1) vers le tube p :

```
close(1);
close(p[0]);
dup(p[1]);
close(p[1]);
```

Fermeture de la sortie standard
Interdiction de lecture
On duplique l'entrée correspondant au descripteur d'écriture du tube dans la table des descripteurs (le descripteur de la sortie standard est libre on le relie au tube pour écriture)

Exemple : la commande ps –e | wc –l

```
main()
{ int p[2] ;

if  (pipe(p)==-1)
          { perror("creation du tube");
           exit(2);
          }
switch(fork()){

case –1 : /* Erreur */
           perror("Fork");
           exit();
case 0 : close(1);
          close(p[0]);
           dup(p[1]);
           close(p[1]);
           execlp("ps", "ps", "-e", NULL);
           exit();
case 1 : close(0);
          close(p[1]);
           dup(p[0]);
           close(p[0]);
           execlp("wc", "wc", "-l", NULL);
           exit();
}
```

6. Les Tubes Nommées

La définition d'un tube nommé vise à permettre à des processus sans lien de parenté particulier dans le système, de communiquer en mode flot. Ils ont toutes les caractéristiques des tubes que nous avons mentionnées et ont la propriété supplémentaire de posséder des références dans le système de gestion de fichiers.

- Permettre à des processus sans lien de parenté particulier dans le système, de communiquer en mode flot.
- Ils ont toutes les caractéristiques des tubes (FIFO,…)
- Propriété supplémentaire de posséder des références dans le système de gestion de fichiers
- Tube nommées Résident alors que les tubes temporaires)

6.1 Création

Pour créer un tube nommé, il faut créer une référence dans le système de fichiers. Depuis le *shell*, on peut utiliser la commande *mkfifo*. Il existe cependant, la primitive *mkfifo()* qu'il est possible d'appeler depuis un programme. Le prototype de cette fonction se trouve dans le fichier d'entête <sys/stat.h>

```
int mkfifo(char *ref, mode_t mode);
```

`ref` : nom physique du tube sur le disque
`mode` : Ce paramètre permet d'indiquer les permission du mode de création (Lecture, Ecriture et Exécution). Mode de création qui est conjugué avec le `umask` et les informations sur le propriétaire

6.2 Ouverture d'un tube nommé

La primitive open utilisée pour l'ouverture des fichiers, permet aussi d'ouvrir un tube de communication avec des modes en écriture/lecture. La fonction open retourne un descripteur de fichier. Cette primitive a la propriété d'être bloquante lorsque elle est appliquée à un tube nommé.

```
int open(const char *pathname, int flags);
```

open() renvoie le nouveau descripteur de fichier , ou -1 s'il échouent, auquel cas errno contient le code d'erreur.

pathname : le chemin complet d'accès au tube (fichier)
fllags : Le paramètre flags doit inclure l'un des mode d'accès suivants :
O_RDONLY, O_WRONLY : l'ouverture en lecture seule ou écriture seule.

6.3 Écriture/Lecture dans un tube nommé

La lecture et l'écriture sur le tube nommé s'effectuent de la même manière en utilisant les primitives read() et write() utilisées pour les pipes et les fichiers.

Exemple : Deux processus P1, P2
Le processus P1 récupère une chaine de caractère en paramètre et l'écrit dans le tube
Le processus P2 lit dans le tube nommé caractère par caractère et l'affiche en majuscule

Processus P1

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


main (int argc , char* argv[])
{ int i,fd;


if( mkfifo("tube",0666) ==-1)
      {perror("mkfifo"),
      exit(0);}

fd=open("tube",O_WRONLY);
if(fd==-1)
      {perror("open"),
      exit(0); }

write(fd,argv[1],strlen(argv[1])*sizeof(
char));
close(fd);
}
```

Processus P2

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

main (int argc , char* argv[])
{int fd;
char c;
fd=open("tube",O_RDONLY);

if(fd==-1)
{
perror("open");
exit(0);
}

while (read(fd,&c,1)!=0)
printf("%c",toupper(c));
printf("\n");
close(fd);

}
```