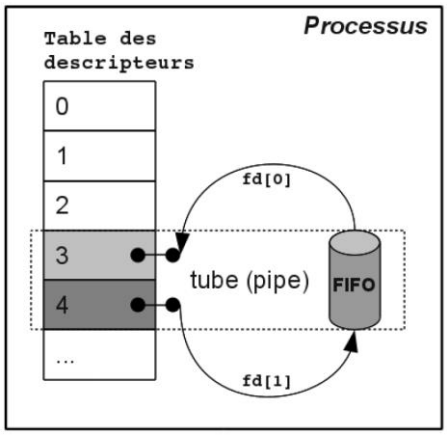
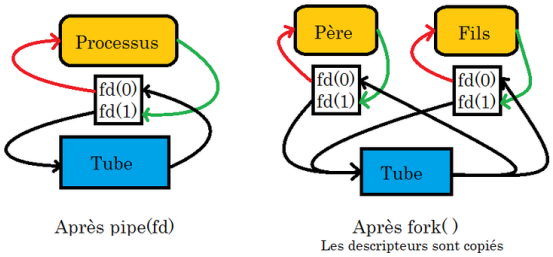




Après pipe(fd)

Après fork( )
Les descripteurs sont copiés

Extrait du référentiel : BTS Systèmes Numériques option A (Informatique et Réseaux)		Niveau(x)
S6. Systèmes d'exploitation S6.2 S.E. multitâche professionnelles	Communications interprocessus (IPC) Règles d'échange de données : modèles producteur/consommateur, modèle lecteur/rédacteur	2 3
S4. Développement logiciel S4.9 Programmation événementielle	Environnement multitâche : traitements parallèles (thread, sémaphores, tubes, …)	3

Objectifs du cours :

Ce cours traitera essentiellement les points suivants :

- Les tubes de communication :
  - les tubes sans nom
  - les tubes nommés

Un autre mécanisme de communication entre processus est l'**échange de messages**. Chaque message véhicule des données. Un processus peut envoyer un message à un autre processus se trouvant sur la même machine ou sur des machines différentes. Unix offre plusieurs mécanismes de communication pour l'envoie de messages : les **tubes de communication** sans nom, nommés et les **sockets**.

LES TUBES DE COMMUNICATION

Les **tubes de communication** ou « pipes » permettent à deux ou plusieurs processus d'échanger des informations. On distingue deux types de tubes :

- les **tubes sans nom** ou anonymes (unamed pipe) ;
- et **les tubes nommés** (named pipe).

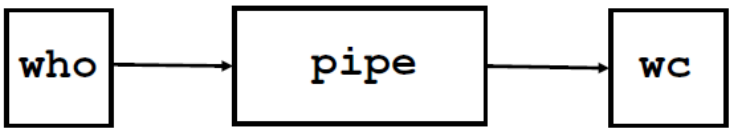
LES TUBES SANS NOM

Les **tubes sans nom** sont des liaisons **unidirectionnelles** de communication.
La taille maximale d'un tube sans nom varie d'une version à une autre d'Unix, mais elle est approximativement égale à 4 Ko. Les tubes sans nom peuvent être créés par le shell ou par l'appel système pipe().

Les tubes sans nom du shell sont créés par l'opérateur binaire « | » qui dirige la sortie standard d'un processus vers l'entrée standard d'un autre processus. Les tubes de communication du shell sont supportés par toutes les versions d'Unix.

Exemple (en shell) : La commande shell ci-dessous crée deux processus qui s'exécutent en parallèle et qui sont reliés par un tube de communication pipe (voir figure ci-dessous). Elle détermine le nombre d'utilisateurs connectés au système en appelant who, puis en comptant les lignes avec wc :

```
who | wc -l
```



Processus en parallèle reliés par un tube de communication

Le premier processus réalise la commande who. Le second processus exécute la commande wc -l. Les résultats récupérés sur la sortie standard du premier processus sont dirigés vers l'entrée standard du deuxième processus via le tube de communication qui les relie.

Le processus réalisant la commande who ajoute dans le tube une ligne d'information par utilisateur du système. Le processus réalisant la commande wc -l récupère ces lignes d'information pour en calculer le nombre total. Le résultat est affiché à l'écran. Les deux processus s'exécutent en parallèle, les sorties du premier processus sont stockées sur le tube de communication.

Lorsque le tube devient plein, le premier processus est suspendu jusqu'à ce qu'il y ait libération de l'espace nécessaire pour stocker une ligne d'information. De façon similaire, lorsque le tube devient vide, le second processus est suspendu jusqu'à ce qu'il y ait au moins une ligne d'information sur le tube.

Exemple (par appel système) : Un tube de communication sans nom est créé par l'appel système pipe, auquel on passe un tableau de deux entiers :

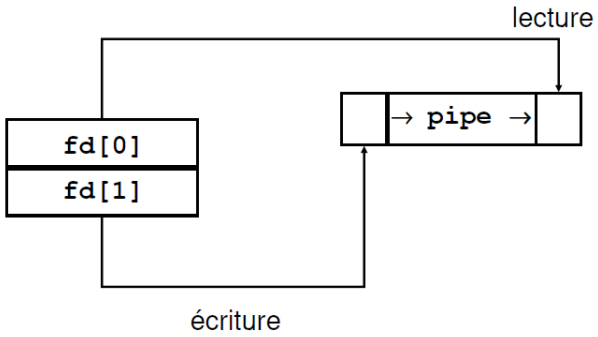
```
int pipe(int descripteur[2]);
```

Au retour de l'appel système pipe(), un tube aura été créé, et les deux positions du tableau passé en paramètre contiennent deux descripteurs de fichiers. Nous considérerons pour le moment un descripteur comme une valeur entière que le système d'exploitation utilise pour accéder à un fichier. Dans le cas du tube on a besoin de deux descripteurs : le descripteur pour les lectures du tube et le descripteur pour les écritures dans le tube. L'accès au tube se fait via les descripteurs. Le descripteur de l'accès en lecture se retrouvera à la position 0 du tableau passé en paramètre, alors que le descripteur de l'accès en écriture se retrouvera à la position 1. Seul le processus créateur du tube et ses descendants (ses fils) peuvent accéder au tube. Si le système ne peut pas

créer de tube par manque d'espace, l'appel système pipe() retourne la valeur -1, sinon il retourne la valeur 0.
Les déclarations suivantes créent le tube de l'exemple précédent :

```
int fd[2];
pipe(fd);
```

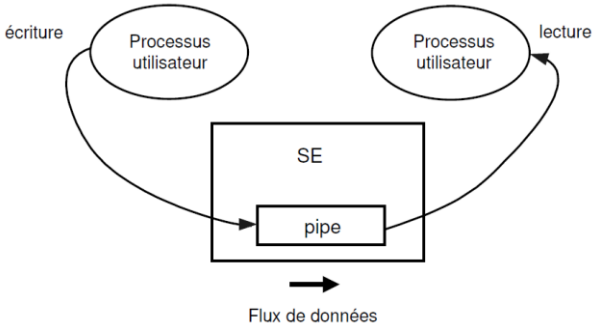
Les tubes sans nom sont, en général, utilisés pour la communication entre un processus père et ses processus fils, avec un d'entre eux qui écrit sur le tube, appelé **processus écrivain (ou rédacteur)**, et l'autre qui lit à partir du tube, appelé **processus lecteur** (voir figure ci-dessous).



Pipe de lecture/écriture

La séquence d'événements pour une telle communication est comme suit :

- Le processus père crée un tube de communication sans nom en utilisant l'appel système pipe() ;
- Le processus père crée un ou plusieurs fils en utilisant l'appel système fork() ;
- Le processus écrivain ferme l'accès en lecture du tube ;
- De même, le processus lecteur ferme l'accès en écriture du tube ;
- Les processus communiquent en utilisant les appels système write() et read() ;
- Chaque processus ferme son accès au tube lorsqu'il veut mettre fin à la communication via le tube.



Création d'un tube de communication entre deux processus

Exemple 1 : Dans le programme ci-dessous le processus père crée un tube de communication pour communiquer avec son processus fils. La communication est unidirectionnelle du processus fils vers le processus père.

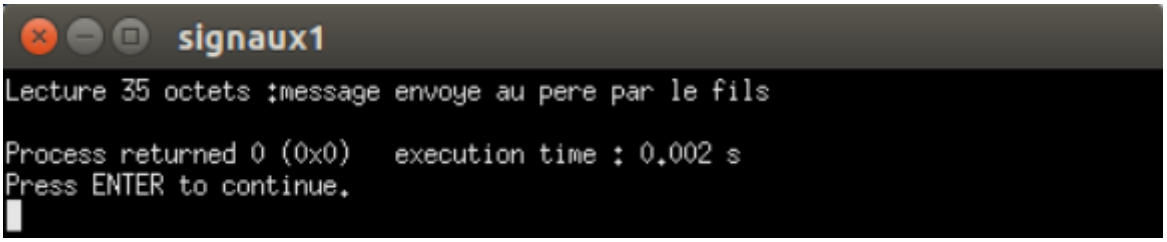
```
#include <sys/types.h>   // types
```

```
#include <unistd.h>      // fork, pipe, read, write, close
#include <stdio.h>
#include <string.h>      // strlen
#define R 0
#define W 1

int main ()
{
int fd [2];
char message [100]; // pour recuperer un message
int nboctets;
char *phrase = "message envoye au pere par le fils";
pipe (fd); // creation d'un tube sans nom
if (fork() == 0) // creation d'un processus fils
{
// Le fils ferme le descripteur non utilise de lecture
close (fd[R]);
// depot dans le tube du message
write (fd[W], phrase, strlen(phrase)+1);
// fermeture du descripteur d'ecriture
close (fd[W]);
}
else
{
// Le pere ferme le descripteur non utilise d'ecriture
close (fd[W]) ;
// extraction du message du tube
nboctets=read(fd[R], message, 100);
printf("Lecture %d octets :%s\n", nboctets, message);
// fermeture du descripteur de lecture
close (fd[R]);
}
return 0;
}
```

Contenu du fichier : upipe.c

```
# gcc -o upipe upipe.c
# ./upipe
```



Compilation et exécution du fichier « upipe.c »

Le processus fils de l'exemple a inclus le caractère nul dans le message envoyé au processus père. Le père peut facilement le supprimer. Si le processus écrivain envoie plusieurs messages de

longueurs variables sur le tube, il est nécessaire d'établir des règles qui permettent au processus lecteur de déterminer la fin d'un message, c'est-à-dire, d'établir un **protocole de communication**.

Par exemple, le processus écrivain peut précéder chaque message par sa longueur ou bien terminer chaque message par un caractère spécial comme retour chariot ou le caractère nul.

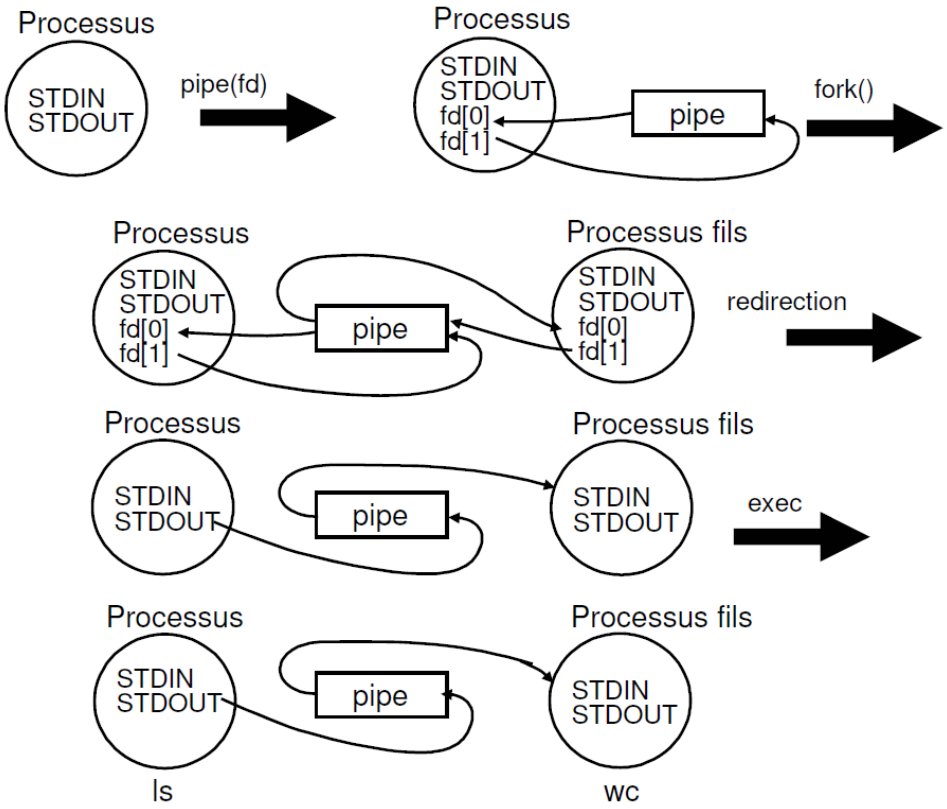
La figure ci-dessous illustre comment un shell exécute une commande comme **ls | wc**.

Dans un premier temps, le tube est créé avec la commande pipe().

Puis un processus fils est créé. Ce processus fils héritera des accès en entrée et en sortie du tube.

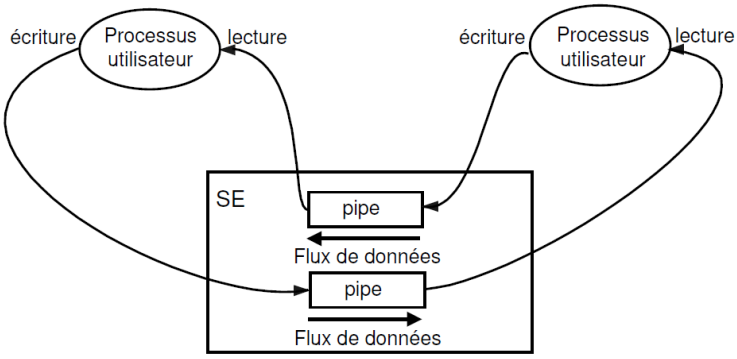
Ensuite, le processus parent ferme son accès à la sortie du tube et fait pointer STDOUT sur l'entrée du tube. Dans le même temps, le processus fils ferme son accès à l'entrée du tube et redirige STDIN sur la sortie du tube.

Finalement, les deux processus sont remplacés par les programmes ls et wc.



Usage des descripteurs de fichiers pendant la communication par tube unidirectionnel entre deux processus.

La **communication bidirectionnelle** entre processus est possible en utilisant deux tubes : un pour chaque sens de communication, comme illustré ci-dessous.



Création d'un tube de communication bidirectionnelle entre deux processus

LES TUBES NOMMÉS

Linux supporte un autre type de tube de communication, beaucoup plus performants. Ils s'agit des **tubes nommés** (named pipes). Les tubes de communication nommés fonctionnent aussi comme des files du type FIFO (First In First Out). Ils sont plus intéressants que les tubes sans nom car ils offrent, en plus, les avantages suivants :

- ils ont chacun un nom qui existe dans le système de fichiers (une entrée dans la table des fichiers) ;
- ils sont considérés comme des fichiers spéciaux ;
- ils peuvent être utilisés par des processus indépendants, à condition qu'ils s'exécutent sur une même machine ;
- ils existeront jusqu'à ce qu'ils soient supprimés explicitement ;
- ils sont de capacité plus grande, d'environ 40 Ko.

Ainsi, deux processus sans relation parent-enfant peuvent échanger des données au moyen des tubes nommés.

Les tubes de communication nommés sont créés par la commande shell **mkfifo** ou par l'appel système **mkfifo()** :

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

où nom contient le nom du tube et mode indique les permissions à accorder au tube. Ainsi, le segment de code suivant permet de créer un tube nommé avec les permissions en lecture et écriture pour tous :

```
if(mkfifo(nom_tube,0666,0) != 0)
{
printf("Impossible de créer %s\n",nom_tube);
exit (1);
}
```

On peut aussi utiliser la commande mkfifo du shell pour créer un tube nommé :

```
# mkfifo tube
#
```

On peut voir l'affichage des attributs du tube créé comme on le fait avec les fichiers :

```
# ls -l tube
prw-r--r-- 1 root root 0 mar 6 11:15 tube
#
```

De la même façon, il est possible de modifer des permissions d'accès :

```
# chmod g+rw tube
# ls -l tube
# prw-rw-r-- 1 root root 0 mar 6 11:15 tube
#
```

Dans les permissions, « **p** » indique que tube (donc « **p**ipe ») est un tube.

Une fois le tube créé, il peut être utilisé pour réaliser la communication entre deux processus. Pour ce faire, chacun des deux processus ouvre le tube, l'un en mode écriture et l'autre en mode lecture.

Exemple 2 : Le programme **ecrivain.c** écrit un message sur le tube **montube**. De son coté, le programme **lecteur.c** lit un message à partir du même tube.

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

int main (void)
{
int fd;
char message [26];
sprintf (message,"bonjour de l'ecrivain [%d]\n",getpid());
//Ouverture du tube montube en mode écriture
fd=open( "montube",O_WRONLY);
printf("ici ecrivain [%d] \ n",getpid());
if (fd!=-1) {
write(fd, message,strlen(message));
}
else
printf("desole, le tube n'est pas disponible\n");
// Fermeture du tube
close (fd);
return 0;
}
```

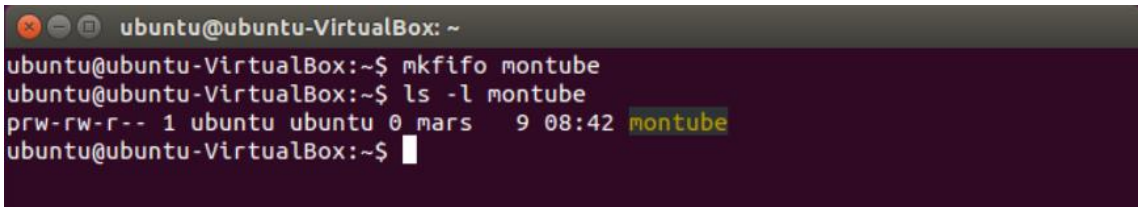
Contenu du fichier : ecrivain.c

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

int main ( )
{
int fd,n;
char input;
// ouverture du tube montube en mode lecture
fd=open("montube",O_RDONLY);
printf("ici lecteur [%d] \n" ,getpid());
if(fd !=-1 ) {
printf ("Recu par le lecteur :\n");
while ((n=read(fd,&input,1))>0) {
printf("%c ",input);
}
printf("Le lecteur se termine !\n",n);
}
else
printf("desole, le tube n'est pas disponible\n");
close(fd);
return 0;
}
```

Contenu du fichier : lecteur.c

Après avoir crée le tube « montube » avec **mkfifo**, on peut compiler séparément les deux programmes :



Affichage en console

Puis on lance en « background » ecrivain et lecteur qui communiqueront via le tube.

```
# ./ecrivain& ./lecteur&
ici lecteur[4851]
ici ecrivain[4850]
Recu par le lecteur: bonjour de l'ecrivain [4850]
Le lecteur se termine!
[1]- Done          ./ecrivain
[2]+ Done          ./lecteur
```