Chapitre 5: Tubes

Principales caractéristiques

- Communication synchrone
- Unidirectionnel
- Limitations: Taille, sérialisation, …
- Mécanismes: voir cours systèmes de fichiers
- Intra-machine
- Volatiles ou persistants (FIFO ou tube nommé)

➔ Implémentation du shell: >, <, |

Bidirectionnel :

- Syst. V R.4
- implantation par les sockets sur linux

pipe: tube de communication

Syntaxe :

```
#include <unistd.h.h>
int pipe ( int fd[2]);
```

Retour :

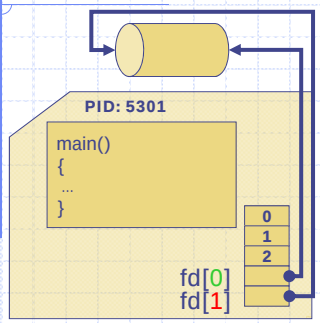
- 0, si *OK*,
- -1, si *erreur*

Descripteurs en lecture et écriture

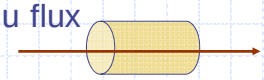
- ❑ Paramètre résultat
- ❑ utilisable directement avec
  - ✓ write
  - ✓ read
  - ✓ ...

- ❑ Création d'un tube
- ❑ Disparait si plus aucun processus ne peut l'utiliser
- ❑ Utilisable par la descendance (exclusivement)
  - ✓ fork
  - ✓ execve

pipe: avec un processus (1)

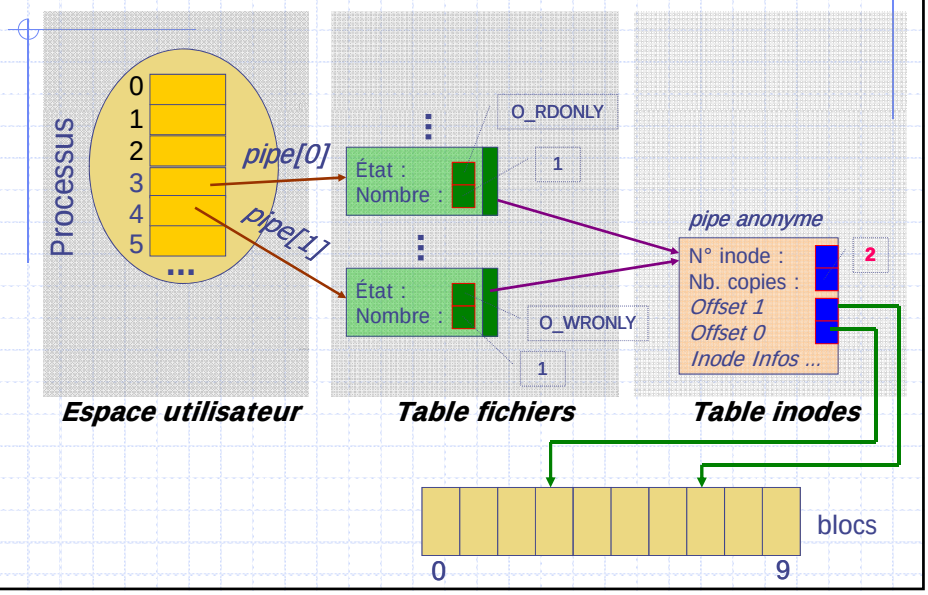


Sens du flux

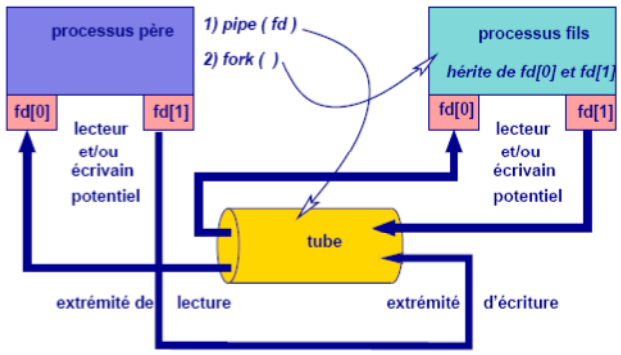



```
#include <unistd.h>
int main () {
  int fd[2];
  int err;
  char *s1, *s2;
  s1 = "chaine1";
  s2 = (char *) malloc (10);
  err = pipe (fd);
  write (fd[1], s1, 8);
  read (fd[0], s2, 8);
      ...
  }
```

pipe: avec un processus (1)

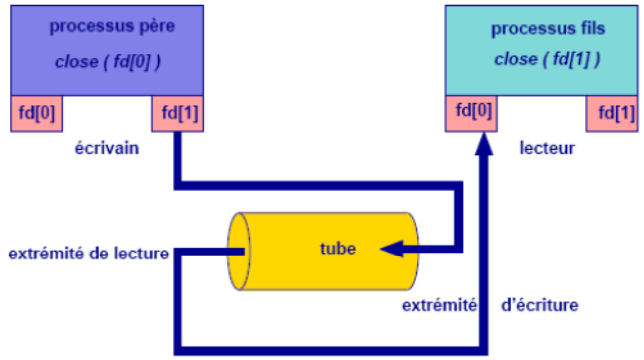


pipe : entre deux processus (1)



- Le fils « hérite » des descripteurs du père
- les deux processus : lecteur et/ou écrivain
- Echange : un seul sens mais n'importe lequel

pipe : entre deux processus (2)



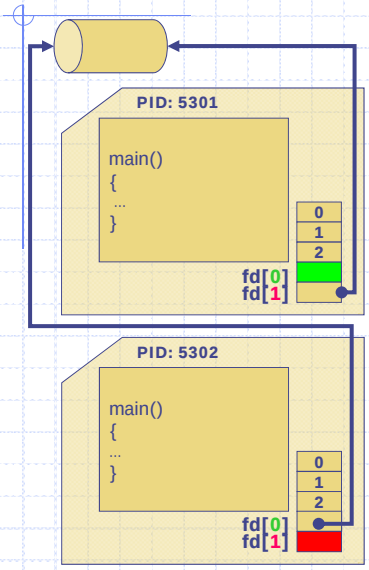
- Fermeture d'une des extrémités dans chaque processus
- Un seul « écrivain » et un seul « lecteur »
- Choix arbitraire d'un sens « père » vers « fils »

pipe : entre deux processus (3)

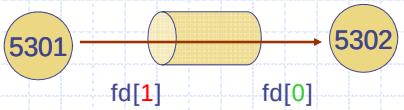
```
#include <unistd.h>
int main () {
        int fd[2], pid;
        char *s1, *s2;
        int err;
        ...
        s1 = "chaine1";
        err = pipe (fd);
        if ( (pid = fork ()) < 0) erreur;
        if (pid == 0) { /* Traitement fils */
                s2 = (char *) malloc (10);
                close (fd[1]);
                read (fd[0], s2, 8);}
        else { /* Traitement père */
                close (fd[0]);
                write (fd[1], s1, 8);}
        ... }
```

?

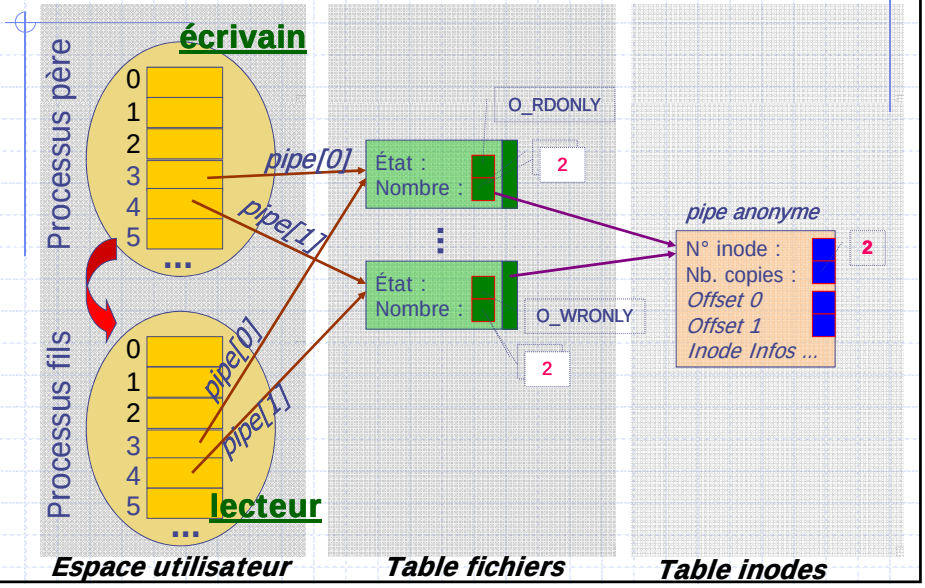
pipe : entre deux processus (4)



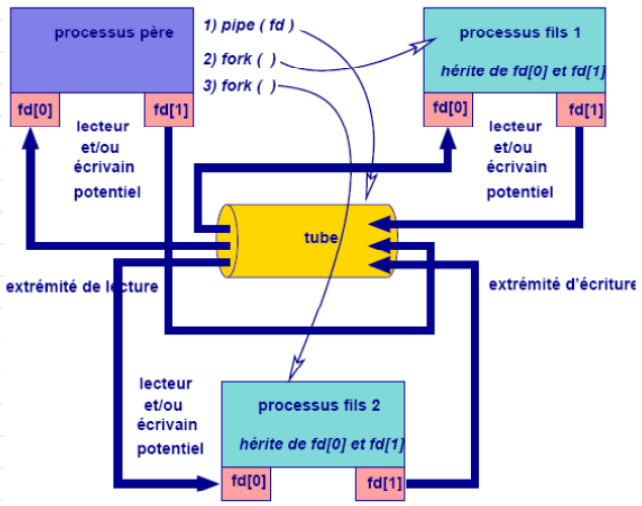
Sens du flux



pipe : entre deux processus (5)

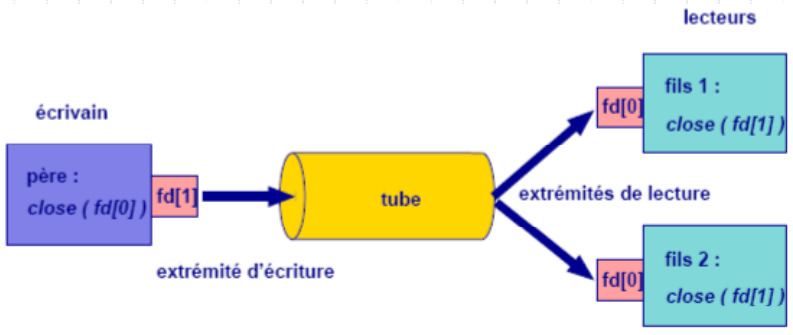


pipe : entre trois processus (1)



Situation après création du tube et de 2 processus fils

pipe : entre trois processus (2)



- Fermeture d'une des extrémités dans chaque processus
- Un seul « écrivain » : le père
- et deux « lecteurs » : les fils
- Toute combinaison est possible

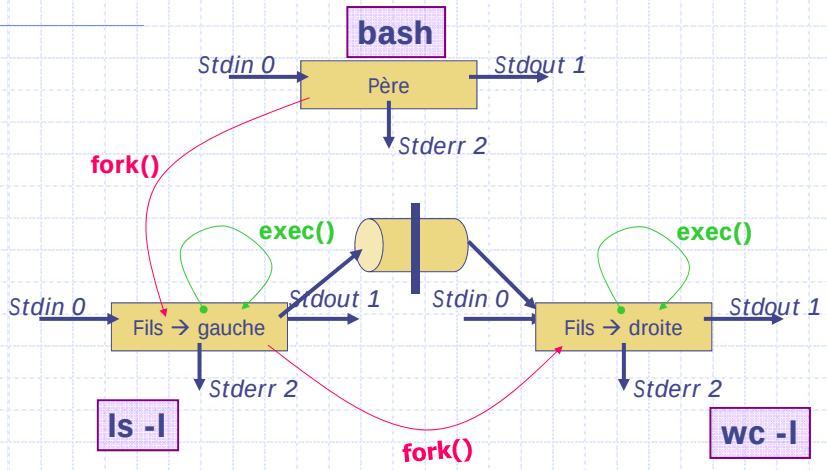
pipe: compléments (1)

- Informations complémentaires
  - Utilisation de blocs directs
  - Structure FIFO (premier écrit, premier lu)
    - → Gestion d'une file circulaire
  - Position de déplacement: dans l'inode
    - → Fichier ordinaire: dans la table des fichiers
  - Lecture (non) bloquante: *no_delay*
- Cas limites
  - Tube vide ou sans « écrivains » : lecture ⇒ EOF
    read retourne 0 - *sinon si rien à lire: blocage*
    ⇒ fermer « fd[1] » si processus pas « écrivain »
  - Tube sans « lecteurs » : écriture ⇒ signal SIGPIPE

pipe: compléments (2)

- Echange par flots ou messages (programmation)
  En général si « message » : lecture/écriture atomique
- Ecriture :
  - interruptible ⇒ Ecriture non atomique
  - Atomique : nb octets à écrire **<** capacité tube
    Voir *fpathconf*
  - Attente possible si pas assez de place
- Lecture :
  - les octets lus sont retirés du tube
  - aucune garantie d'atomicité ou complétude
    Tester le nb octets lu et itérer

pipe/dup: exemple du pipe shell (1)



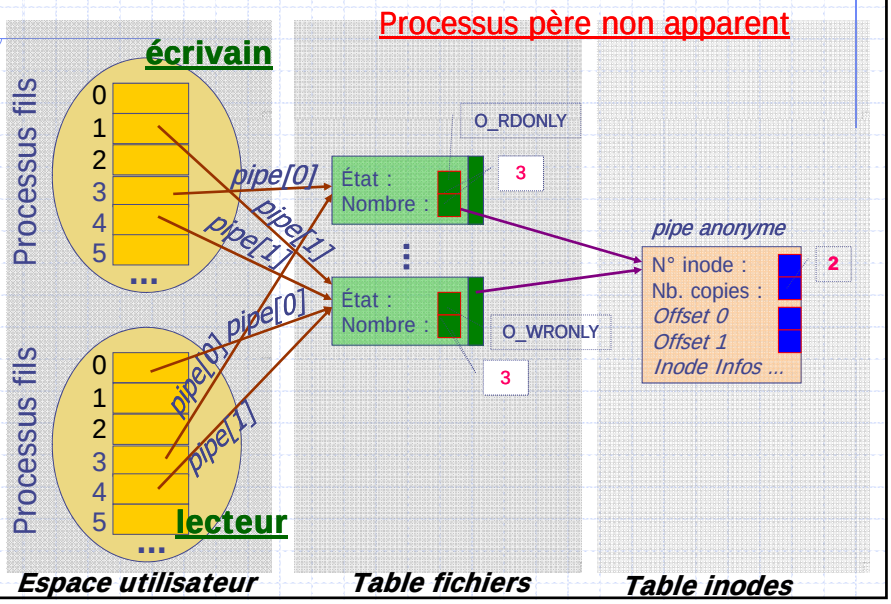
pipe/dup: exemple du pipe shell (2)

```
#include <unistd.h>
int main () {
    int pid, nfd;
    int tab[2];
    ... /* analyse commande */
    if ( (pid = fork ()) == 0) { /* (début 1) */
            ... /* Traitement redirection */ ...
            if (/* mise en tube */) { /* (début 2) */
                    pipe (tab);
                    if ( (pid = fork ()) == 0) {
                            /* partie gauche pipe: stdout --> tube */
                            close (stdout);
                            nfd = dup (tab[1]); /* nfd = stdout */
                            close (tab[1]);
                            close (tab[0]);
                            execlp(gauche, ...); }
```

pipe/dup: exemple du pipe shell (3)

```
#include <unistd.h>
int main () {
   /* début sur T. précédent */
        if (/* mise en tube */) { /* (suite 2) */
                /* partie droite pipe: stdin --> tube */
                close (stdin);
                nfd = dup (tab[0]); /* nfd = stdin */
                close (tab[0]);
                close (tab[1]);
        } /* fin 2 */
        /* suite 1 */
        execve(droit, ...);
   } /* fin 1 */
   if (/* & */) wait (&status);
}
```


pipe/dup: exemple du pipe shell (4)

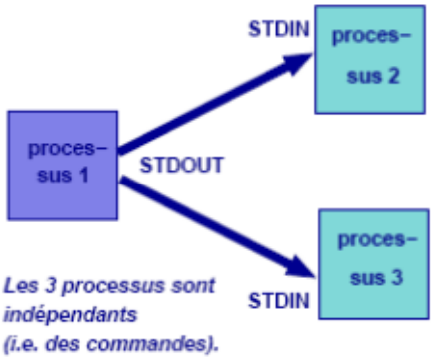


Tube nommé: Principaux aspects

- Points commun avec « pipe »
  - Unidirectionnel
  - Toujours intra-machine
  - Toujours FIFO
- différence avec « pipe » : l'accès
  - Un fichier avec un nom
  - Partage par tout processus (aussi ¬ descendant)
  - Moment de création des processus communicants

Tube nommé: exemple d'utilisation (1)

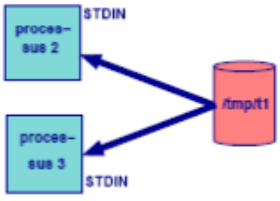
duplication d'informations pour plusieurs processus



Les 3 processus sont indépendants (i.e. des commandes).

a) `proc1 > /tmp/t1`

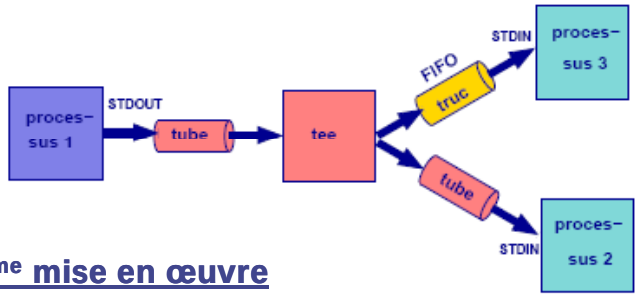
b) `proc2 < /tmp/t1 &`
`proc3 < /tmp/t1 &`



1ère mise en œuvre

Tube nommé: exemple d'utilisation (2)

```
mkfifo truc
proc1 | tee truc | proc2 &
proc3 < truc &
```



2ème mise en œuvre

mkfifo: tube nommé

Syntaxe :

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

Nom fichier

Protection

Retour :

- 0, si *OK*,
- -1, si *erreur*

- ❑ Création d'un tube nommé (FIFO)
- ❑ Masque de protection: mode & ¬ umask
- ❑ Utilisation appels systèmes pour fichier ordinaire

Exemple : *dup2*, *read*, *write*, *fcntl*, *etc.*

Mais pas *lseek*

- ❑ Fréquent: création du tube avec *mkfifo(1)* mis en œuvre avec *mkfifo(2)*

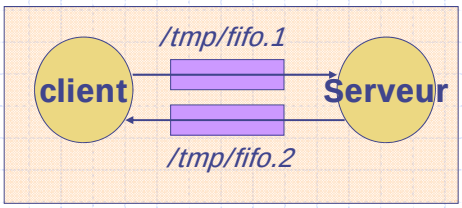
Utilisation d'un tube nommé

- **open** :
  - une ouverture en mode O_WRONLY (resp. O_RDONLY) est bloquante jusqu'à ce qu'un autre processus ouvre le même FIFO en lecture (resp. écriture).
  - Si l'ouverture est en mode O_RDWR ou si l'option est O_NONBLOCK, alors l'ouverture est non bloquante.
  - ⇒l'ouverture d'un FIFO peut provoquer un « rendez-vous » entre 2 processus !
- **write** et **read** : même comportement que pour un tube anonyme

Tube nommé: Exemple d'utilisation

- **Fichier fifo.h:**

```
#include type.h stat.h errno.h
extern in errno;
#define FIFO1 "/tmp/fifo.1"
#define FIFO2 "/tmp/fifo.2"
#define PERMS 0666
```



- **Fichier client:**

```
#include "fifo.h"
int main () {
        int readfd, writefd;
        if ( (writefd = open (FIFO1, 1)) < 0) erreur;
        if ( (readfd = open (FIFO2, 0)) < 0) erreur;
        client (readfd, writefd);
        close (readfd); close (writefd);
}
```

Tube nommé: Exemple d'utilisation

Fichier serveur:

```
#include "fifo.h"
int main () {
        int readfd, writefd;
        if ( (mkfifo (FIFO1, PERMS)) < 0
                && (errno != EEXIST)) unlink + erreur;
        if ( (mkfifo (FIFO2, PERMS)) < 0
                && (errno != EEXIST)) erreur;
        if ( (writefd = open (FIFO1, 1)) < 0) erreur;
        if ( (readfd = open (FIFO2, 0)) < 0) erreur;
        server (readfd, writefd);
        close (readfd); close (writefd);
        if (unlink (FIFO1) < 0) erreur;
        if (unlink (FIFO2) < 0) erreur;
}
```