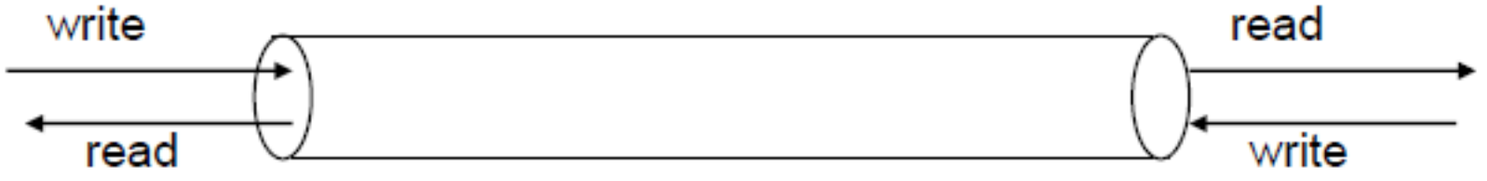
Tubes (pipes)

- canal de communication bidirectionnel FIFO (Linux: unidirectionnel)
- permet d'échanger une séquence d'octets
- tamponnage éventuel sur disque (transparent)
- plusieurs producteurs possibles / plusieurs consommateurs possibles
- un producteur écrit dans le tube; un consommateur lit dans le tube



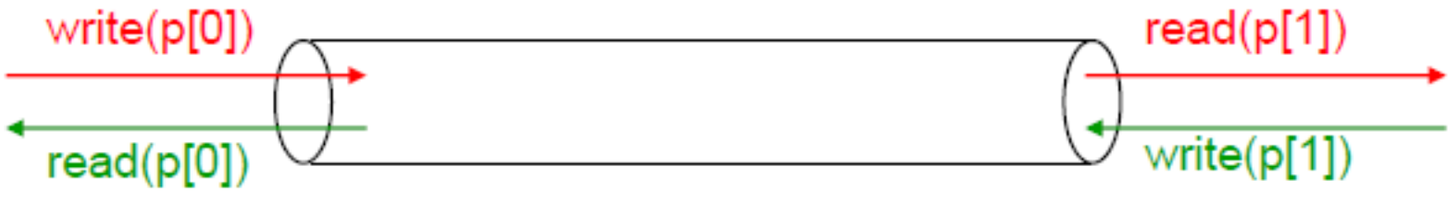
- distinguer:
  - tubes *anonymes*
  - tubes *nommés*

Tubes anonymes

Création d'un tybe anonyme:

```
int pipe (int p[2])
```

- pas d'erreur: résultat de l'appel = 0
- erreur: résultat de l'appel = -1
- l'appel retourne dans `p[0]` et `p[1]` des indices dans la table des descripteurs de fichiers du processus
- l'information écrite dans `p[0]` est lue dans `p[1]`
- l'information écrite dans `p[1]` est lue dans `p[0]`
- ne permet la communication qu'entre processus liés par filiation.



Lecture / écriture d'un tube

`int read(p[1], buf, bufsize)`

- `buf` est l'adresse du tampon
- `bufsize` est la taille du tampon
- l'appel retourne en résultat le nombre d'octets lus

`int write(p[0], buf, bufsize)`

Synchronisation et cas particuliers:

- la lecture est blocante
- le lecteur est bloqué si le tube est vide mais ouvert en écriture par un autre processus
- `read` retourne 0 (EndOfFile) si le tube est vide mais ouvert en écriture par aucun autre processus
- le signal SIGPIPE est envoyé au processus qui exécute `write` si le tube n'a été ouvert en lecture par aucun autre processus

Autoblocage / interblocage

Autoblocage

```
int tube[2];
. . .
pipe(tube);
/* le tube est vide et le processus en est le seul écrivain
read (tube [0] , bufLecture, 1);  // Blocage
. . .
write (tube [1], bufEcriture, 1);
```

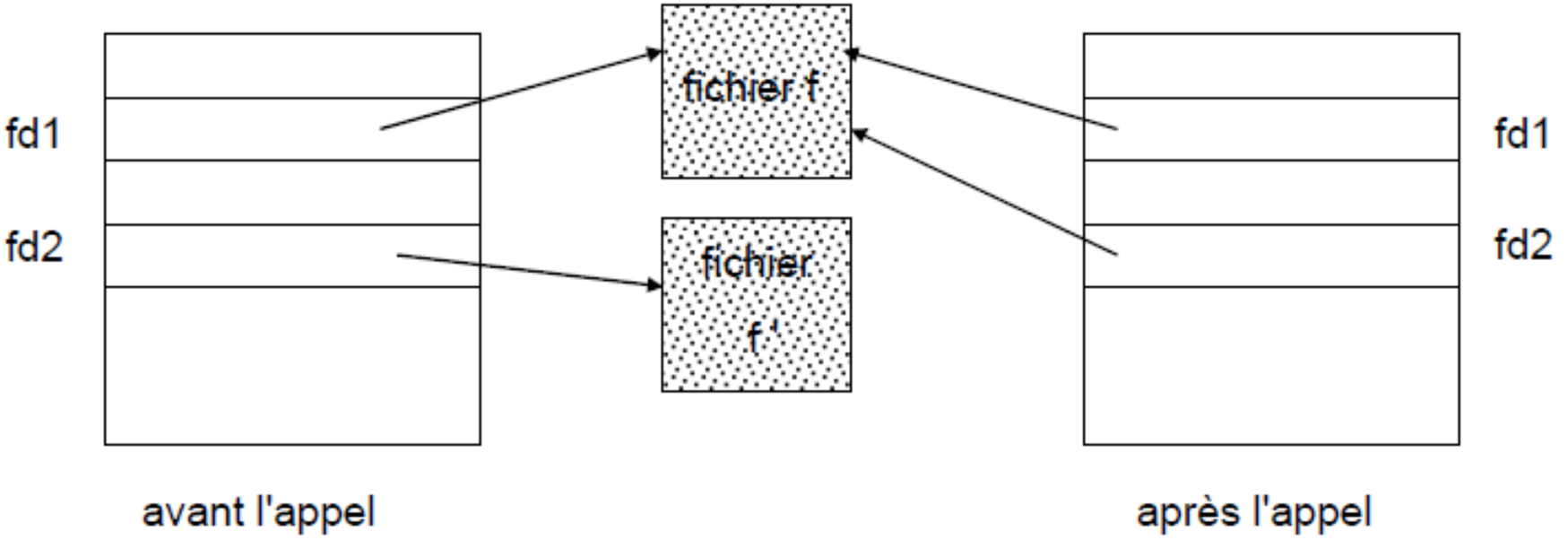
Interblocage

```
int tube1[2], tube2[2];
. . .
pipe (tube1); pipe (tube2);
/* deux processus conservant les deux descripteurs tube1 et tube2
if (fork() == 0) {
  read(tube1[0], buf1, 1);  // Blocage sur tube1 qui est vide
  write(tube2[1], buf2, 1);
}
else {
  read (tube2[0] , buf1, 1); // Blocage sur tube2 qui est vide
  write(tube1[1], buf2, 1);
}
. . .
```

Fonction `dup2`

```
int dup2 (int fd1, int fd2)
```

- copie l'entrée `fd1` de la table des descripteurs de fichiers dans l'entrée `fd2`
- après l'appel, `fd1` et `fd2` désignent le même fichier, à savoir le fichier désigné par `fd1` avant l'appel



Fonction `pipe`: exemple

Processus-père

(1) `pipe(p);`

(2) `fork( );`

(3) `close(p[1]);`

(4) `dup2(p[0], Stdin);`

(5) `close(p[0]);`

Lecture de Stdin:

- obtient les caractères écrits par le processus-fils dans Stdout

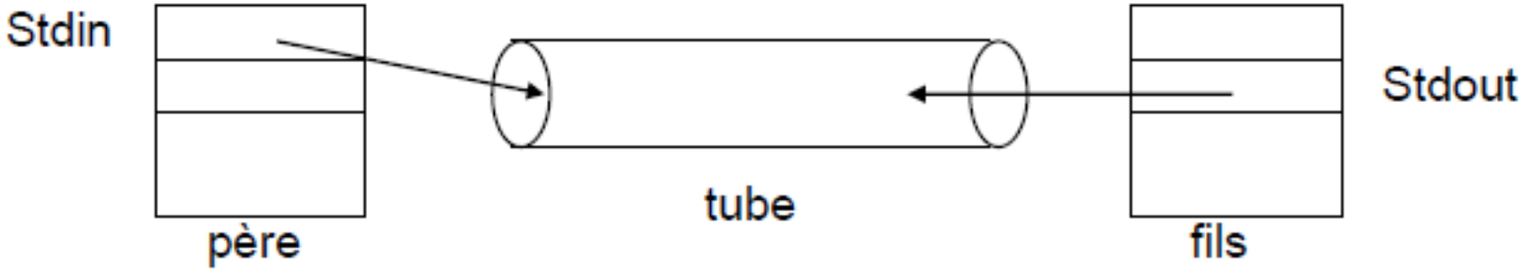
Processus-fils

(3) `close(p[0]);`

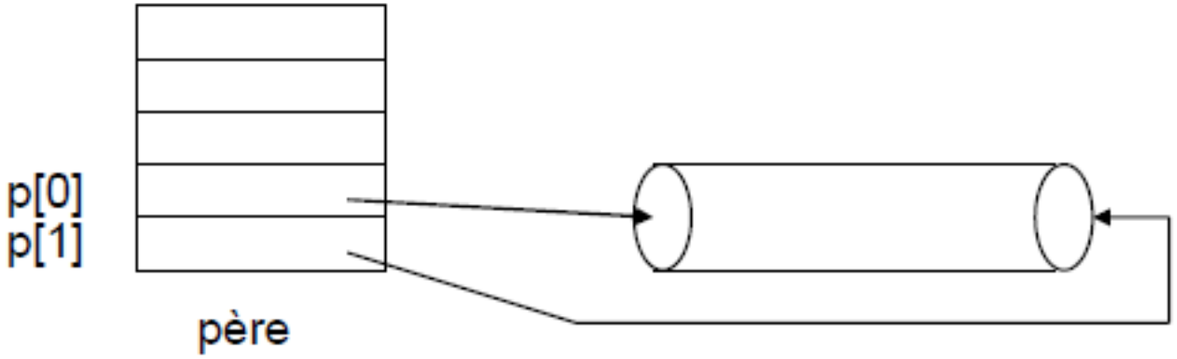
(4) `dup2(p[1], Stdout);`

(5) `close(p[1]);`

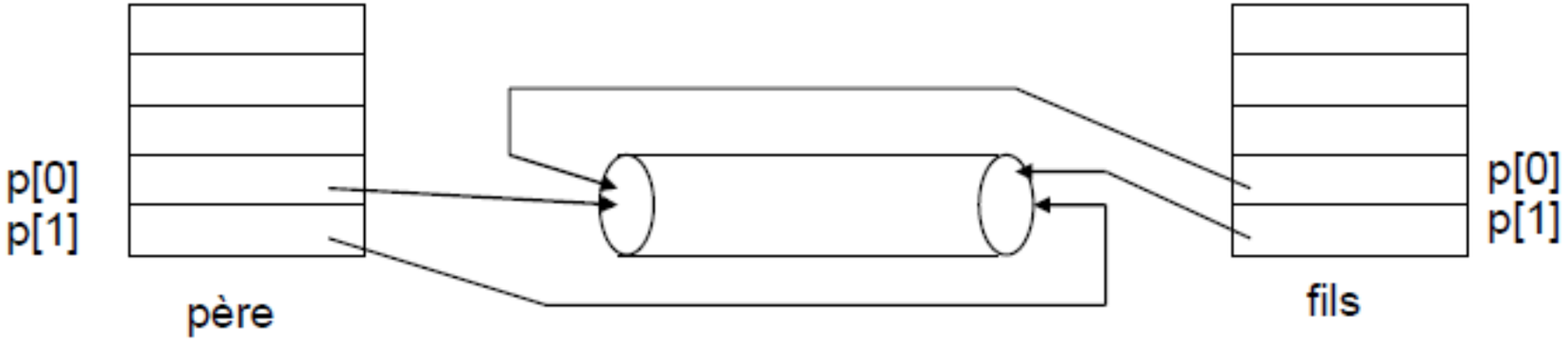
Ecriture dans Stdout



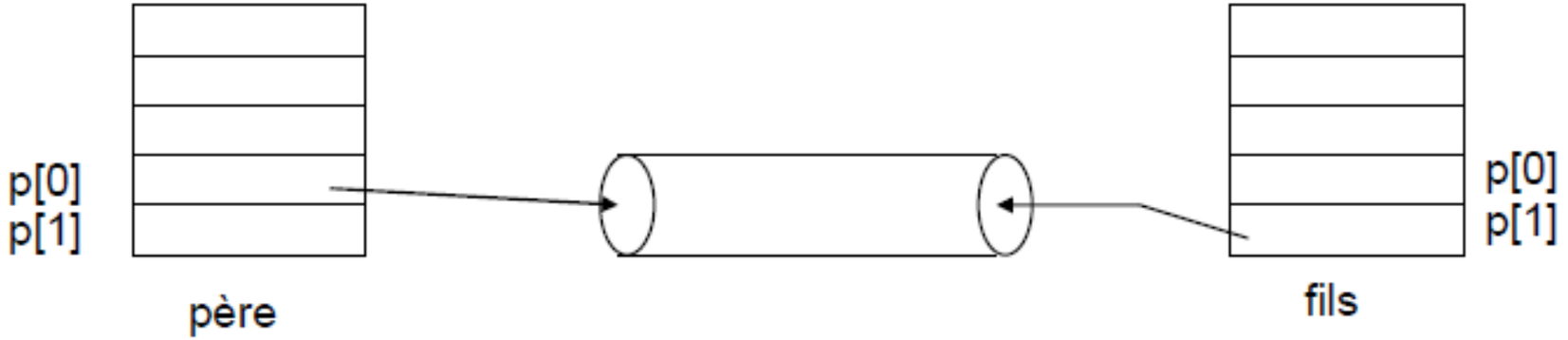
après (1)



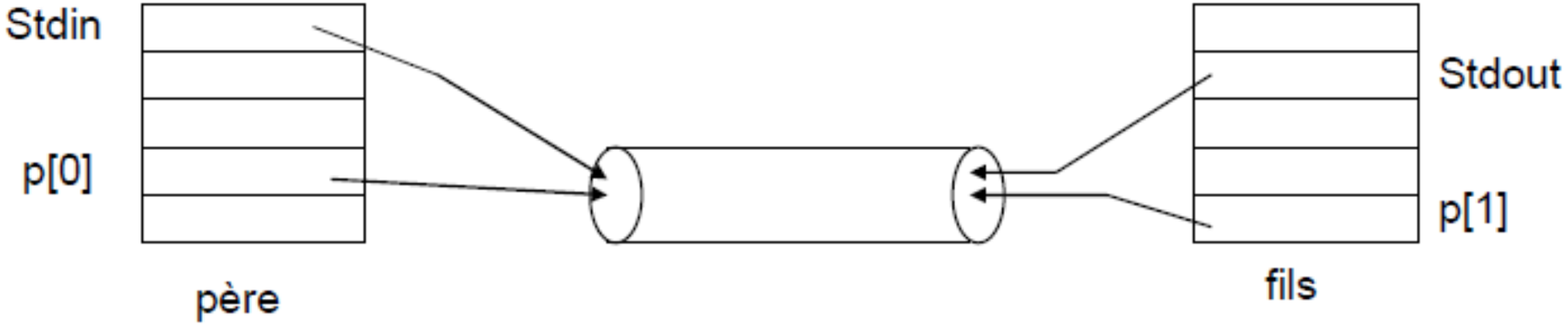
après (2)



après (3)



après (4)



Tubes nommés (FIFO)

- tube unidirectionnel (aussi appelé FIFO)
- fichier d'un type particulier
- création (langage de commande): `/etc/mknod/toto p`
  (crée le tube nommé `toto`; `p` spécifie "pipe")
- création (appel système):

```
mkfifo(const char *ref, mode_t droits);
```

- utilisé comme un fichier normal (`open` / `close`, `read` / `write`)
- désignation et protection: identique aux fichiers
- synchronisation
  - ouverture en lecture: le processus est bloqué jusqu'à ce qu'un processus ait ouvert le tube en écriture
  - ouverture en écriture: le processus est boqué jusqu'à ce qu'un processus ait ouvert le tube en lecture

Tube nommé: exemple

Emetteur (fichier *emet*)

```
#include <stdio.h>
#include <sys/fcntl.h>
main( ) {
  int fd;

  fd = open(''toto'', O_WRONLY);
  write(fd, ''un message'', 10);
  printf(''%d]> fin ecriture'',
         getpid() );
}
```

Récepteur (fichier *recoit*)

```
#include <stdio.h>
#include <sys/fcntl.h>
main( ) {
  int fd; char buf[50];

  fd = open(''toto'', O_RDONLY);
  read(fd, buf, sizeof(buf));
  printf(''%d]> recu'', %s \n'',
         getpid(), buf);
}
```

```
% /etc/mknod toto p

% recoit &
[1] 2190

% emet
2191]> fin ecriture
[1] Done recoit
2190]> recu: un message
```