I2181-B
LINUX
APPELS SYSTÈME

PIPE

PIPES : GENERALITES

- En ligne de commande : ls -l | wc -l

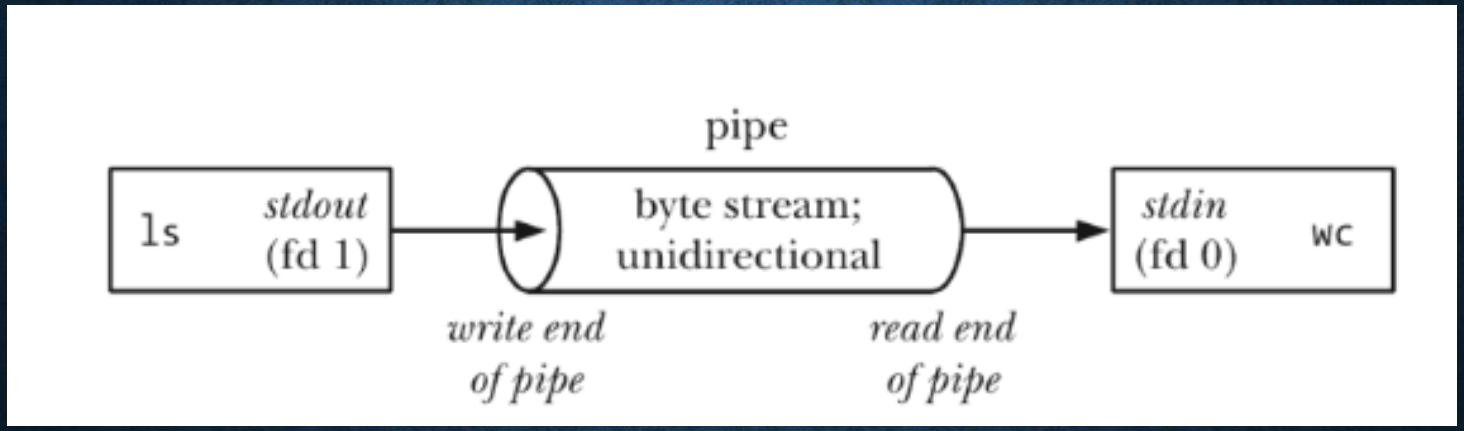


Image source : The Linux Programming Interface, Michael Kerrisk, 2010

PIPES : GENERALITES

- En ligne de commande : ls -l | wc -l
- Un *pipe* (tube) est un moyen de communication entre processus
- Utilisé généralement de manière FIFO et unidirectionnel :
  - un seul processus lit
  - un seul processus écrit

PIPE

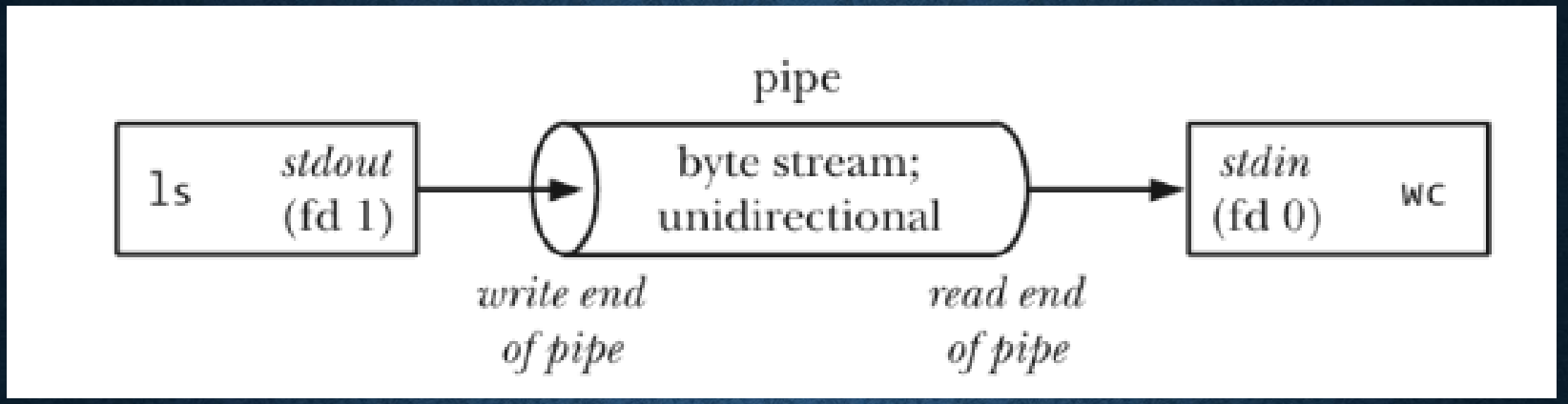
Création d'un pipe par programmation :

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

Où: filedes : tableau vide qui sera initialisé par pipe()

filedes[0] : permettra de lire dans le pipe

filedes[1] : permettra d'écrire dans le pipe




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

Où: filedes : tableau vide qui sera initialisé par pipe()

filedes[0] : permettra de lire dans le pipe

filedes[1] : permettra d'écrire dans le pipe

REMARQUE

- Tubes anonymes
  - Communication entre un processus père et un fils

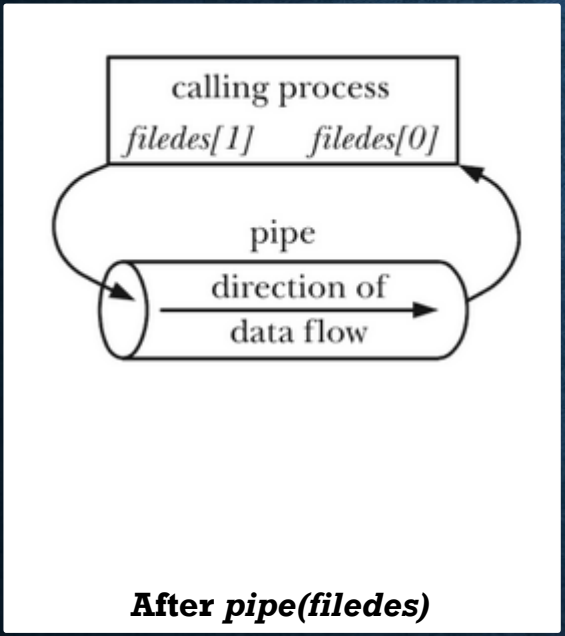
  → Objet de ce cours
- Tubes nommés
  - Communication entre processus quelconques
  - `int mkfifo(char* name, mode_t mode)`

  → Pas l'objet de ce cours

CONFIGURATION D'UN PIPE

- Création du pipe AVANT la création du processus fils
- Les 2 processus ont ainsi accès aux descripteurs de fichiers du pipe
- Dans chaque processus, fermer le descripteur de fichier non utilisé

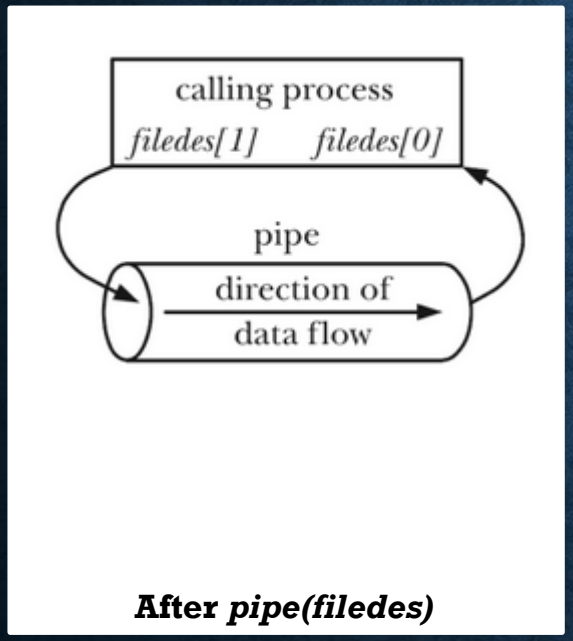
CONFIGURATION D'UN PIPE



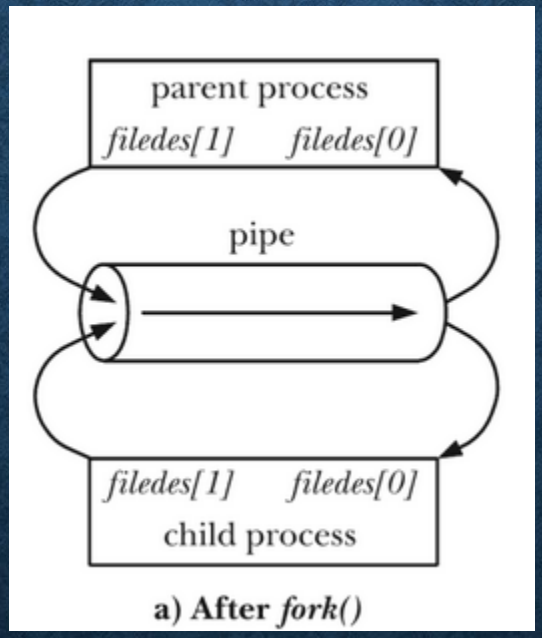
After *pipe(filedes)*

Image source: The Linux Programming Interface, Michael Kerrisk, 2010

CONFIGURATION D'UN PIPE



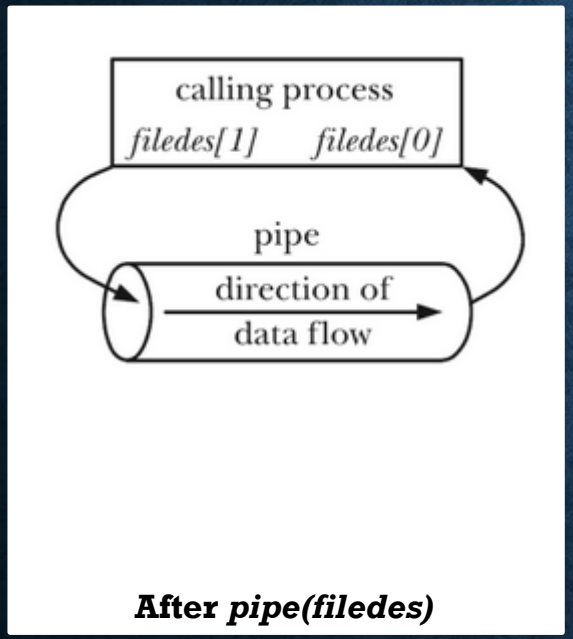
After *pipe(filedes)*



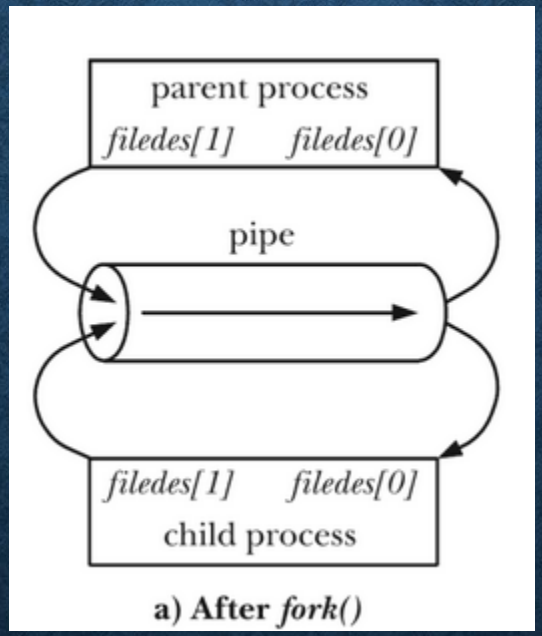
a) After *fork()*

Image source: The Linux Programming Interface, Michael Kerrisk, 2010

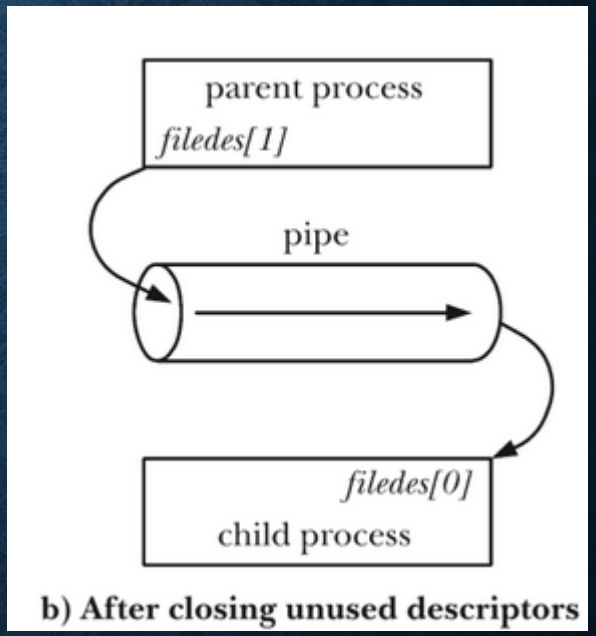
CONFIGURATION D'UN PIPE



After *pipe(filedes)*



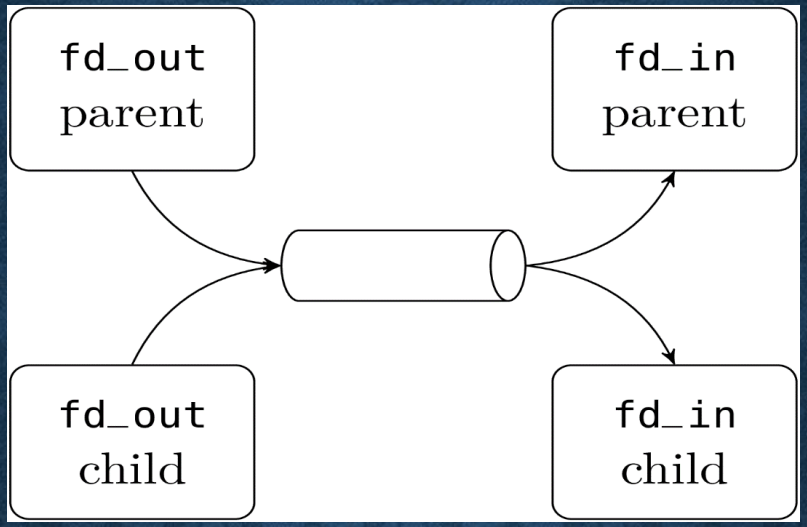
a) After *fork()*



b) After closing unused descriptors

Image source: The Linux Programming Interface, Michael Kerrisk, 2010

CONFIGURATION D'UN PIPE



- La fermeture d'un descripteur de fichier (*close*) permet d'indiquer au système d'exploitation qu'un processus n'utilise plus ce descripteur.
- Une lecture sur un descripteur de fichier fd[0] dont l'autre extrémité fd[1] est totalement fermée en écriture (plus d'écrivain) renvoie 0 (EOF)
- Une écriture sur un descripteur de fichier fd[1] dont l'autre extrémité fd[0] est totalement fermée en lecture (plus de lecteur) génère une erreur (SIGPIPE → voir signaux)

EXEMPLE 1

- Envoi d'un entier depuis le processus père vers son fils

EXEMPLE 2

- Envoi d'un entier depuis le processus père vers son fils
- Version fork_and_run

EXEMPLE 3

- Envoi d'un entier depuis le processus père vers son fils
- Terminaison incorrecte: on continue à écrire alors que personne n'écoute.

GESTION DES FILE DESCRIPTORS

- La gestion des file descriptors (fd) utilise trois tables :
  - Table des fd : nb entiers positifs, une table par processus
  - Table des fichiers ouverts : permissions et offset de chaque fd utilisé, table globale
  - Table des I-nodes : "path" des fichiers, table globale

GESTION DES FILE DESCRIPTORS

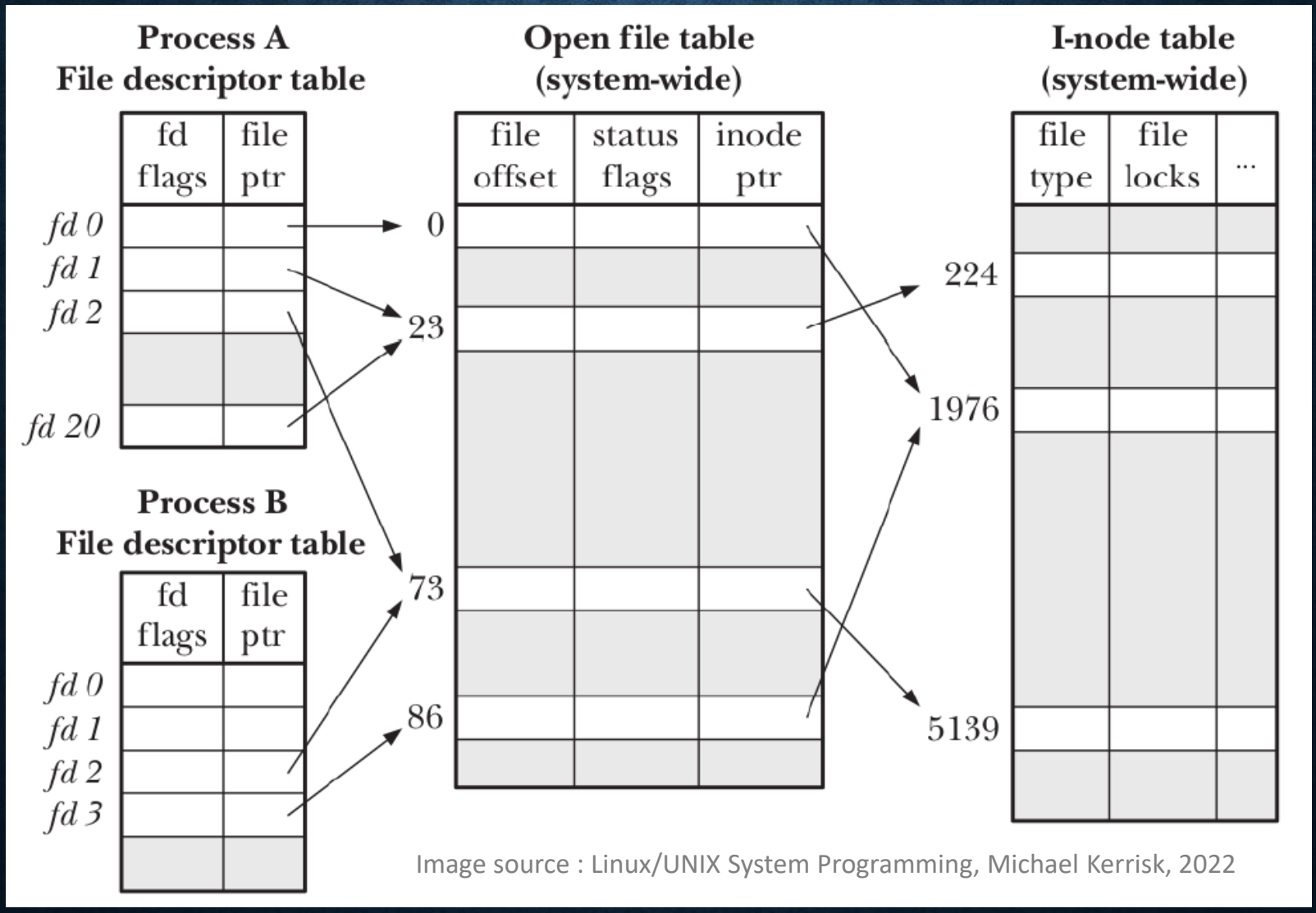


Image source : Linux/UNIX System Programming, Michael Kerrisk, 2022

GESTION DES FILE DESCRIPTORS

- L'appel système open associe un fd à une ressource (table des fichiers)
- Plusieurs fd peuvent pointer vers la même ressource dans la table des fichiers ouverts (mécanisme de duplication – appels système `dup/dup2`)
- L'appel système `close` libère le fd. Si c'est la dernière référence vers une ressource, celle-ci est libérée

REDIRECTION

- Exemples de redirections :
  - redirection de la sortie standard dans un fichier

```
ls -l > sortie.txt
```

  - redirection de la sortie standard dans un pipe

```
ls -l | grep ".sh"
```

  - …

REDIRECTION

REDIRECTION


```
close(1);
```

REDIRECTION


```
close(1);
dup(pipe[1]);
```

REDIRECTION


```
close(1);
dup(pipe[1]);
```

ou

```
dup2(pipe[1],1)
```

REDIRECTION

- Ce qu'on appelle redirection est en fait de la duplication de fd
- `dup` : le plus petit fd disponible pointe la même ressource que le fd en paramètre – il le duplique
- `dup2` : un fd spécifié est fermé si nécessaire, puis pointe vers la même ressource qu'un autre fd spécifié

DUP / DUP2

```
int dup(int fd)
```

- **dup**() makes the lowest available fd be the copy of *fd*

```
int dup2(int oldfd, int newfd)
```

- **dup2**() makes *newfd* be the copy of *oldfd*, closing *newfd* first if necessary