

# Programmation système

Ressource R3.05 - Tubes, redirections

---

monnerat@u-pec.fr 

IUT de Fontainebleau

## 1. API fichiers - primitives

## 2. Tubes

- Définition
- Création
- Lecture et écriture
- Fermeture
- Accès non bloquant
- Redirection

## 3. Tubes nommés

## API fichiers - primitives

---

- `open()` : crée une nouvelle description de fichier ouvert dans la table globale des fichiers ouverts. Réserve un nouveau numéro descripteur de fichier pour le processus (et le lui renvoie).
- `read()` : transfère des données depuis le fichier vers la mémoire. Avance la position dans la description de fichier ouvert.
- `write()` transfère des données depuis la mémoire vers le fichier. Avance la position dans la description de fichier ouvert.
- `lseek()` modifie la position dans la description de fichier ouvert.
- `close()` libère le numéro descripteur et, éventuellement, détruit la description de fichier ouvert.
- `fcntl()` modifie les attributs d'un fichier ouvert.

Toutes les primitives travaillent avec des descripteurs de fichiers.

Un descripteur de fichier pointe vers une description d'un fichier ouvert.

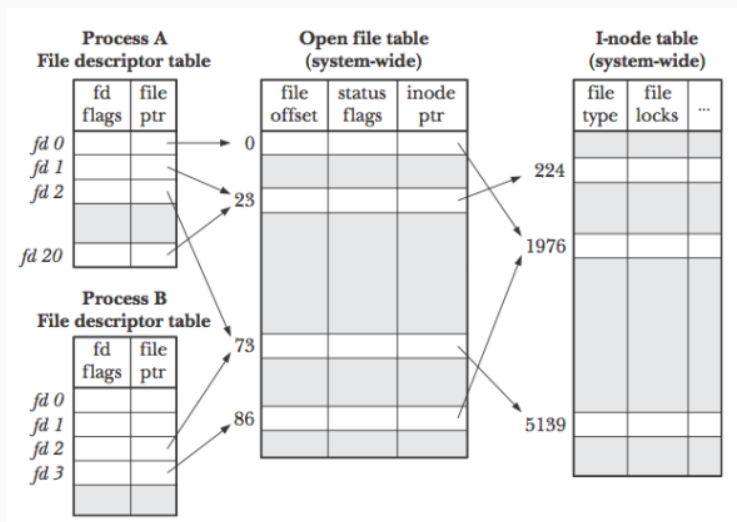


Figure 1 – descripteur de fichier

Plusieurs cas sont possibles :

- Dans un même processus, 2 descripteurs pointent sur la même entrée dans la table des fichiers ouverts :  
`dup` et `dup2`
- Dans 2 processus distincts, 1 descripteur dans chaque processus pointe sur la même entrée :  
`fork`
- Deux entrées différentes de la table des fichiers ouverts font référence au même inode :  
2 processus ont fait `open` sur le même fichier

# Ouverture

```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ...);
int open(const char *path, int oflag, mode_t mode);
```

- path : le fichier à ouvrir.
- oflag : soit O\_RDONLY, O\_WRONLY, O\_RDWR.  
On peut ajouter (ou bit à bit) les masques O\_APPEND, O\_CREAT, O\_EXCL, O\_DIRECT, O\_SYNC, O\_NONBLOCK, etc .
- mode : avec O\_CREAT pour la création, précise les droits d'accès en octal.
- O\_CREAT|O\_EXCL : test et création atomique.

Renvoie le numéro de descripteur attribué au fichier ouvert ( ou -1 si erreur).

1. Le noyau retrouve l'inœud du fichier indiqué par le chemin
  - `O_CREAT` est parmi les drapeaux et le fichier n'existe pas : un nouveau inœud est créé avec les permissions dans mode
  - `O_CREAT` et `O_EXCL` sont parmi les drapeaux et le fichier existe : `open()` échoue
  - `O_WRONLY` / `O_RDWR` et `O_TRUNC` sont parmi les drapeaux : le fichier est remis à vide
2. Une nouvelle description de fichier ouvert est créée :
  - les attributs d'état sont les drapeaux (modifiables par `fcntl()`).
  - la position (le numéro du prochain octet à lire ou à modifier) est mise à 0
  - pointe sur l'inœud du fichier.
3. Le premier descripteur libre dans la table de descripteurs du processus pointe sur la nouvelle description.



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

```
int close(int fd);
```

1. Libère le descripteur
2. S'il n'y a plus de pointeurs sur la description de fichier ouvert, alors l'entrée est détruite.
3. Met à jour les informations dans l'inœud
  - si le nombre de références (hard links) dans l'inœud est 0 et il n'y a plus de descriptions de fichier ouvert pour cet inœud, alors l'inœud est détruit et le fichier n'est plus accessible
  - si `close()` renvoie -1, alors l'état du fichier est inconnu, perte de données possible

```
#include <unistd.h>

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

Lit des données à partir de la position courante.

- `fd` : descripteur de fichier.
- `buf` : tampon où seront stockées les données lues.
- `count` : nombre d'octets à lire.

Renvoie le nombre d'octets lus ( $\leq count$ ), et avance la position associée à `fd`.

- `0` indique la fin du fichier.
- `-1` indique une erreur.

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

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

Si `O_APPEND` est parmi les attributs d'état, met la position à la fin du fichier. Écrit des données à partir de la position courante.

- `fd` : descripteur de fichier
- `buf` : adresse des données à écrire.
- `count` : le nombre d'octets à écrire.
- Renvoie le nombre d'octets écrits (  $\leq count$  ).
- Augmente la valeur de position du même nombre.
- Renvoie `-1` en cas d'erreur.

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

```
off_t lseek(int fildes, off_t offset, int whence);
```

Modifie la position courante dans la description du fichier.

- `fildes` : descripteur de fichier.
- `offset` : déplacement d'octets.
- `whence` :
  - `SEEK_SET` : par rapport au début.
  - `SEEK_CUR` : par rapport à la position courante.
  - `SEEK_END` : par rapport à la fin.

Renvoie la nouvelle position, ou `-1` en cas d'erreur.

**Attention** : tous les fichiers ne sont pas `seekables` (tube ou socket par exemple)

projeter un fichier en mémoire

```
void *mmap(void *addr, size_t length, int prot, int flags,  
           int fd, off_t offset);
```

On ne passe plus par le cache du système.

# Commande cat

```
int cat(int fd){
    char buf[256];
    int nb;
    while(1){
        nb=read(fd,buf,256);
        if (nb <=0) return nb;
        write(STDOUT_FILENO,buf,nb);/*on suppose que tout
                                     est ecrit(sinon coder un full_write)*/
    }
}

int main(int argc,char ** argv){
    int n,fd,ret,i;
    if (argc==1) cat(STDIN_FILENO);
    for(i=1;i<argc;i++){
        fd=open(argv[i],O_RDONLY);
        if (fd == -1) { perror("open error()");
        }else{
            ret = cat(fd);
            if (ret == -1) perror("read() error");
            close(fd);
        }
    }
}
```

# Tubes

---

# Tubes

---

## Définition





pipe : canal de communication **unidirectionnel** entre processus

- flux d'octets entre une entrée et une sortie,
- 2 descripteurs de fichiers,
- un tampon système de capacité limité
- un octet lu disparaît,
- fichier anonyme (pas dans l'aborescence) : partage entre père et descendants.

Outil de **communication** et de **synchronisation** interprocessus.

# Tubes

---

## Création

```
#include <unistd.h>

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

Crée un tube et place dans :

- `fildes[0]` : le descripteur pour la lecture. (analogie avec 0 pour `stdin`)
- `fildes[1]` : le descripteur pour la l'écriture. (analogie avec 1 pour `stdout`)

Renvoie `-1` en cas d'erreur. Les descripteurs d'un tube sont héritées par les fils.

- **lecteur** : tout processus qui possède le descripteur de lecture.
- **écrivain** : tout processus qui possède le descripteur d'écriture.

# Tubes

---

Lecture et écriture

**Lecture** : appel système `read(fd, buf, nBytes)`

Renvoie le nombre d'octets lus  $\leq nBytes$ , ou -1 en cas d'erreur

- si le tampon système du tube est vide, `read()` est bloquant jusqu'à ce qu'un écrivain y mette des données.
- si le tube est vide et si il n'y a plus d'écrivains, `read()` renvoie 0.

**Écriture** : Appel système `write(fd, buf, nBytes)`

- si  $nBytes \leq PIPE_BUF$  : **écriture atomique**. Sinon découpage par le système.
- si le tampon est plein ( $\leq nBytes$  octets libres), `write()` est bloquant jusqu'à ce qu'un lecteur fasse de la place.
- s'il n'y a plus de lecteurs, le processus reçoit le signal SIGPIPE. Si SIGPIPE est bloqué, ignoré ou capté, `write()` renvoie -1.

- En général, un processus utilise un tube donnée soit en lecture, soit en écriture. (pourquoi?)
- Pour communiquer dans les deux sens entre deux processus, il faut deux tubes.

# Tubes

---

Fermeture

Primitive `close()`

- Libère le descripteur
- Pour le descripteur d'écriture, agit comme une fin de fichier.

Fermer tous les descripteurs non utilisés  $\Rightarrow$  évite les situations de blocage



# Interblocage

```
#define BSIZE 1024
int main () {
    int fdts[2], /* to son */
        fdfs[2]; /* from son */
    char bufr[BSIZE], bufw[BSIZE];
    pipe(fdts);
    pipe(fdfs);
    switch (fork()) {
        case -1: exit(EXIT_FAILURE);
        case 0:
            read(fdts[0], bufr, 1);
            write(fdfs[1], bufw, 1);
            break;
        default:
            read(fdfs[0], bufr, 1);
            write(fdts[1], bufw, 1);
    }
    printf("bye\n");
    exit(EXIT_SUCCESS);
}
```

# Tubes

---

Accès non bloquant

Par défaut, les lectures et écritures dans un tube sont bloquantes.  
Possibilité d'accès non bloquants avec le masque `O_NONBLOCK`.

```
/* rend l'écriture dur fd[1] non bloquante */  
fcntl(fd[1], F_SETFL, fcntl(fd[1], F_GETFL) | O_NONBLOCK);
```

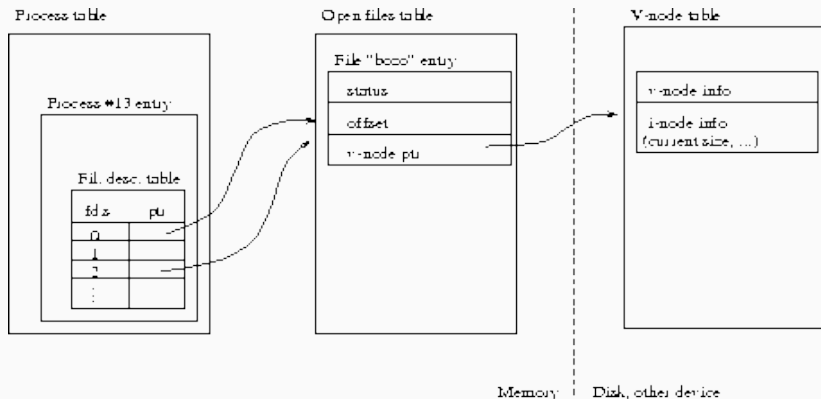
- Écriture non bloquant
  - `write()` retourne -1 si on ne peut pas écrire,
  - positionne `errno` à `EAGAIN`.
- Lecture non bloquante
  - `read()` retourne -1 si le tube est vide et nombre d'écrivains non nul,
  - positionne `errno` à `EAGAIN`.

# Tubes

---

## Redirection

# Dupliquer un descripteur de fichier



# Dupliquer un descripteur de fichier

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

```
int dup(int fildes);
```

dup()

- cherche le plus petit descripteur disponible pour le processus appelant.
- renvoie avec ce descripteur une copie de fildes.

dup() est équivalent à

```
fcntl(fildes, F_DUPFD, 0);
```

```
int dup2(int fildes, int fildes2);
```

dup2()

- ferme fildes2.
- fait fildes2 une copie de fildes.

## Tubes nommés

---



## Inconvénient des tubes anonymes

- les processus communicants doivent avoir un processus parent commun ... qui a créé le tube
- pas de rémanence du tube : détruit, au plus tard, à la terminaison des processus

## Tube nommé ou fifo

- même comportement en lecture/écriture que les tubes anonymes rémanence du tube
- liaison dans le système de fichier : nom du tube

- Commande mkfifo

```
$ mkfifo tube
```

```
$ ls -l tube
```

```
prw-r--r-- 1 denis users 0 12 oct. 09:34 tube
```

- Primitif mkfifo()

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

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

```
int mkfifo(const char *pathname, mode_t mode);
```

POSIX Avec `open()`. Choix exclusif écriture/lecture. **Bloquant** par défaut :

- en lecture
  - si aucun écrivain et
  - aucun processus bloqué en ouverture en écriture.
- en écriture
  - si aucun lecteur et
  - aucun processus bloqué en lecture

Synchronisation des ouvertures en lecture et écriture. **Attention** aux interblocages !

Dans beaucoup de systèmes UNIX (dont linux), l'ouverture d'un tube nommé en lecture/écriture est possible. (mode d'ouverture `O_RDWR`)

- l'ouverture n'est jamais bloquante, et marque chaque extrémité comme ouverte (il y a au moins un lecteur et un écrivain).
- peut être utilisé pour ouvrir un tube en écriture avant qu'un lecteur existe.
- non (Posix) portable. Utiliser un accès non bloquant (`O_NONBLOCK`).