

# Programmation Système

Les tubes





# Sommaire

- Introduction
- Définition
- Fonctions standard de manipulation de fichiers
- Les tubes anonymes
- Création d'un tube anonyme
- Exemple 1
- Lecture dans un tube anonyme
- Écriture dans un tube anonyme
- Duplication des descripteurs
- Les tubes nommés
- Création d'un tube en langage C

# Programmation Système: les tubes

La sortie standard d'une commande peut servir d'entrée standard à une autre commande

Par exemple : **ls -l/etc** | **wc -l**

Si on exécute cette commande, le résultat de **ls -l/etc** servira d'entrée à la commande **wc -l**

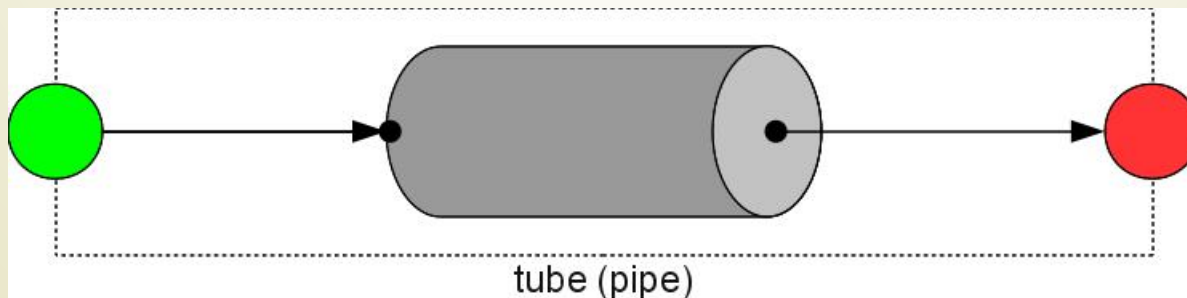
Le système crée un tube de communication entre les deux commandes.

La sortie de **ls -l/etc** est écrite dans le tube

La commande **wc -l** lit son entrée dans le tube

Les processus vont donc pouvoir s'échanger des données par l'intermédiaire des tubes.

**Les tubes sont donc un mécanisme de communication entre processus résidant sur une même machine.**





# Programmation Système: les tubes

## Définition :

Un tube de communication est un canal ou tuyau (en anglais pipe) dans lequel un processus peut écrire des données (le producteur, écrivain ou rédacteur) et un autre processus peut les lire (consommateur ou lecteur).

- C'est un moyen de communication **unidirectionnel** inter-processus. C'est le moyen de communication le plus simple entre deux processus.
- Pour avoir une communication **bidirectionnelle** entre deux processus, il faut créer deux tubes et les employer dans des sens opposés.

# Programmation Système: les tubes

- Les tubes appartiennent au système de fichiers.
- Il sera donc possible d'utiliser les fonctions standard d'accès aux fichiers :
  - open
  - read
  - write
  - close
- Il existe deux types de tubes :
  - les tubes anonymes ou ordinaires ou non nommés
  - les tubes nommés

# Programmation Système: les tubes

## *Rappels sur les fonctions standard d'accès aux fichiers:*

**int open(const char \*pathname, int flags)**

- `pathname` : le nom du fichier
- `flags` : les options **O\_RDONLY**, **O\_WRONLY** ou **O\_RDWR**

Valeur de retour : "un file" descriptor (fd).

**ssize\_t write(int fd, const void \*buf, size\_t count)**

- `fd` : retour de `open`
- `buf` : données à écrire
- `count` : nbre d'octets à écrire

Valeur de retour : le nombre d'octets écrits.

# Programmation Système: les tubes

## **fonctions standard** (suite) :

**ssize\_t read(int *fd*, void \**buf*, size\_t *count*)**

- *fd* : retour de open
- *buf* : où écrire les données lues
- *count* : nbre d'octets que l'on peut stocker  
(taille de *buf*)

Valeur de retour : nombre d'octets lus.

**La lecture est destructrice pour un tube.**

**int close(int *fd*)**

- *fd* : retour de open

# Programmation Système: les tubes

## Tubes anonymes

- Un tube anonyme a un inode mais un compteur de référence nul et pas de nom.
- On ne peut donc pas utiliser la fonction `open`.
- Pour utiliser le tube il faut donc recevoir un *file descriptor* (en réalité deux) :
  - soit à la création : fonction `pipe`
  - soit par héritage
- La gestion des tubes se fait en mode FIFO



# Programmation Système: les tubes

- Un file descriptor est destiné à la lecture
- L'autre est destiné à l'écriture.

## Conséquences

- La communication via ces tubes est réservée à des processus qui ont des filiations.
- Quand on ferme le file descriptor d'accès en lecture, on ne peut plus lire dans ce tube.
- Même raisonnement pour l'écriture.

# Programmation Système: les tubes

## Création d'un tube

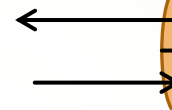
Exécution

Processus

```
int main (int argc, char**  
argv)  
{int fd[2];  
pipe(fd);  
  
}
```

Tube anonyme

fd[0]



fd[1]

fd[0] et fd[1] n'existent plus

Le tube anonyme n'a  
plus de file descriptor

Le processus se termine ...

Le tube disparaît aussi !

# Programmation Système: les tubes

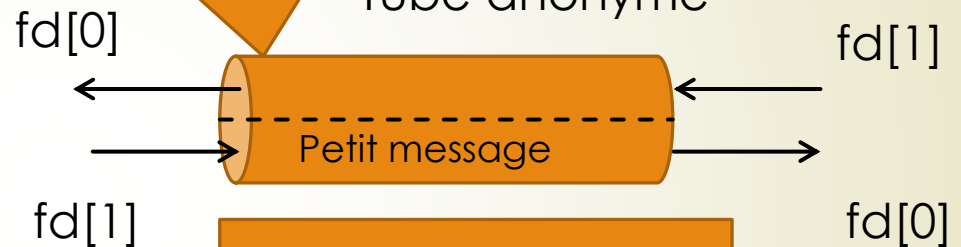
## Dialoguer avec un tube anonyme

Exécution

Processus

Tube anonyme

```
int main (int argc, char**  
argv)  
{int fd[2]; Le fils a toujours  
pipe(fd); les fd du tube  
fork();      Donc le tube  
              existe toujours  
write(...);  Le fils se termine  
}  
              Le tube anonyme n'a  
              plus de file descriptor
```



read(...)

Il hérite des  
variables du  
père

Le processus se termine ...

Le tube disparaît aussi !

# Programmation Système: les tubes

## Caractéristiques d'un tube anonyme

On peut accéder aux caractéristiques du tube anonyme créé:

La fonction *int fsat(int fildes, struct stat \*buf);* permet d'extraire des informations relatives à l'inode créé :

- Le numéro d'inode,
- L'UID du propriétaire,
- Le GID du propriétaire,
- Le nombre d'octets contenus dans le tube,
- L'heure du dernier accès,

La fonction *fcntl* permet de modifier le comportement du tube, il est par exemple possible de rendre la lecture non bloquante.



# Programmation Système: les tubes

```
int main()
{
    int fd[2];
    struct stat info;
    struct tm *ptrtm;
    char heure[20];
    int status;
    if(pipe(fd)==-1)
    {
        perror("pipe");
        return EXIT_FAILURE;
    }

    if( fstat(fd[0],&info)==-1)
    {
        return EXIT_FAILURE;
    }
    printf("\n ID propriétaire %d\n", info.st_uid);
    printf(" ID groupe %d\n",info.st_gid);
    ptrtm localtime(&info.st_atime);
    strftime(heure,100,"%d %B %Y (%H:%M:%S",ptrtm);
    printf("Heure de dernier acces : %s\n",heure);
    printf(" Il y a %lu octet(s) dans le tube\n ",info.st_size);
    status = fcntl(fd[0],F_GETFL);
    printf("\n lecture bloquante : (status = %d)\n",status);
    fcntl(fd[0],F_SETFL,status|O_NONBLOCK);
    status= fcntl(fd[0],F_GETFL);
    printf("\n lecture non bloquante : (status = %d)\n",status);
    close(fd[0]);
    close(fd[1]);
    return EXIT_SUCCESS;
}
```

*Exemple1: → TubeAnonymeInfos*


# Programmation Système: les tubes

## Lecture dans un tube anonyme :

Tube non vide (contient  $n$  caractères):

- `read(fd, buffer, nb)`
  - $nb \leq n$  : lecture de  $nb$  caractères (lecture partielle)
  - $nb > n$  : lecture de  $n$  caractères (lecture totale)

Tube vide :

- pas de rédacteur :
  - `read` retourne 0.  Pas de données et il n'y en aura jamais donc ne bloque pas.
- au moins un rédacteur :
  - lecture non bloquante : `read` retourne -1,
  - lecture bloquante : `read` est bloqué jusqu'à ce qu'il y ait des données dans le tube.

# Programmation Système: les tubes

- ▶ La lecture dans un tube non vide qui contient  $n$  caractères se fait par l'intermédiaire de la fonction :

***ssize\_t read(int fd, void\* buf, size\_t count);***

- ▶ permet de lire le minimum entre  $n$  et *count* et les écrit dans *buf*
- ▶ Si le tube est vide :
  - Si le nombre de rédacteurs est nul (tous les descripteurs fermés) la fonction *read* renvoie 0 ;
  - Si le nombre de rédacteurs n'est pas nul :
    - Si la lecture est bloquante le processus est mis en sommeil jusqu'à ce que le tube ne soit plus vide.
    - Si la lecture n'est pas bloquante la fonction *read* renvoie -1.

# Programmation Système: les tubes

## Écriture dans un tube anonyme :

### ➤ Pas de lecteur :

Le processus reçoit un signal SIG\_PIPE qui par défaut provoque sa destruction

Il n'y a pas de lecteur. Il n'y en aura pas d'autres. Pas la peine d'écrire des données !

### ➤ Au moins 1 lecteur :

- il y a assez de place dans le tube : écriture atomique du message,
- il n'y a pas assez de place dans le tube :

➤ Écriture non bloquante : write retourne -1 sans écriture,

➤ Écriture bloquante : attente de place puis écriture atomique du message.



# Programmation Système: les tubes

- Si un processus est le dernier ou le seul rédacteur, une lecture le bloquera !!

```
int p[2];  
pipe(p);  
read(p[0],buffer,1); /*blocage*/  
write(p[1],texte,strlen(texte));
```

- Deux processus utilisent des tubes pour communiquer mais attendent tous les deux des données :

```
int p1[2],p2[2];  
pipe(p1);  
pipe(p2);  
if (fork()==0)  
{  
    read(p1[0],buffer,1); /*blocage sur p1 vide*/  
    write(p2[1],texte,strlen(texte));  
}  
else  
{  
    read(p2[0],buffer,1); /*blocage sur p2 vide*/  
    write(p1[1],texte,strlen(texte));  
}
```

# Programmation Système: les tubes

## Attention aux blocages !

- ▶ On se bloque soi-même :

```
int p[2] ;  
pipe(p) ;  
read(p[0],buffer,1) ;  
write(p[1],texte,strlen(texte));
```

Blocage !

On est seul rédacteur  
et on attend des données.

# Programmation Système: les tubes

## Attention aux blocages (suite) !

➤ Interblocage :

```
int p1[2];  
pipe(p1);  
if (fork() == 0)  
{  
    read(p1[0],buffer,1);  
    write(p1[1],texte,strlen(texte));  
}  
else  
{  
    read(p1[0],buffer1,1);  
    write(p1[1],texte,strlen(texte));  
}
```

Interblocage !  
Deux rédacteurs  
mais les deux processus  
attendent des données.

# Programmation Système: les tubes

## Attention aux blocages (fin) !

### ➤ Interblocage :

```
int p1[2],p2[2] ;  
pipe(p1) ;  
pipe(p2) ;  
if (fork() == 0)  
{  
    read(p1[0],buffer,1) ;  
    write(p2[1],texte,strlen(texte));  
}  
else  
{  
    read(p2[0],buffer1,1) ;  
    write(p1[1],texte,strlen(texte));  
}
```

Interblocage !  
Deux rédacteurs  
mais les deux processus  
attendent des données.



# Programmation Système: les tubes

## Duplication des descripteurs:

- Il est possible de dupliquer les file descriptors.
- On aura plus de rédacteurs et/ou de lecteurs.
- Mais surtout on pourra rediriger sur les entrées sorties standards (stdin et stdout).

`int dup(int oldfd);`

- Duplique oldfd sur le plus petit numéro disponible.

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

- Duplique oldfd sur newfd en fermant newfd avant si nécessaire.
- Si on duplique sur l'entrée standard, scanf lit dans le tube !
- Si on duplique sur la sortie standard, printf écrit dans le tube !

# Programmation Système: les tubes

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

int main()
{
    int fd[2];
    pid_t pid;
    if(pipe(fd)==-1)
    {
        perror("pipe");
        return EXIT_FAILURE;
    }
    pid= fork();
    switch(pid)
    {
        case -1: close(fd[0]);
                close(fd[1]);
                return EXIT_FAILURE;

        case 0: close (fd[1]);
                close(STDIN_FILENO);
                dup(fd[0]);
                close(fd[0]);
                execlp("wc", "wc", "-l", NULL);
                perror("execlp");
                return EXIT_FAILURE;

        default: close(fd[0]);
                 close(STDOUT_FILENO);
                 dup(fd[1]);
                 close(fd[1]);
                 fprintf(stderr, "%s", "Nombre de fichiers du repertoire \n");
                 execlp("ls", "ls", "-l", NULL);
                 perror("execlp");
                 return EXIT_FAILURE;
    }
}
```

Exemple2: → TubeAnonymeDup

# Programmation Système: les tubes

## Tube anonyme : Exemple complet

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    int    pipefd[2];
    char   buf;
    pid_t  cpid;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    cpid = fork();
    if (cpid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (cpid == 0) { /* lecture dans le pipe par le fils */
        close(pipefd[1]); /* fermeture du descripteur en écriture car non utilisé */
        while (read(pipefd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);
        write(STDOUT_FILENO, "\n", 1);
        close(pipefd[0]);
        exit(EXIT_SUCCESS);
    } else { /* écriture dans le pipe de argv[1] */
        close(pipefd[0]); /* fermeture du descripteur en lecture car non utilisé */
        write(pipefd[1], argv[1], strlen(argv[1]));
        close(pipefd[1]);
        wait(NULL);
        exit(EXIT_SUCCESS);
    }
}
```

*Exemple: → TubeAnonyme*

# Programmation Système: les tubes

## Les tubes nommés

- ✓ Un tube nommé contrairement à un tube anonyme possède un nom et une référence dans le système de fichiers.
- ✓ Il est donc vu par tous les processus. Tous les processus même sans lien de parenté pourront s'échanger des données par l'intermédiaire des tubes nommés, il suffit pour cela de connaître le nom du tube.
- ✓ Le type d'un tube nommé est indiqué par un **P** devant les droits.
- ✓ Un tube nommé peut donc être ouvert grâce à sa référence.
- ✓ Avant de pouvoir ouvrir un tube nommé, il faut le créer au préalable.

*Un tube nommé est donc un fichier spécial permettant à des processus quelconques d'échanger des données en mode FIFO.*



# Programmation Système: les tubes

Création d'un tube nommé sous le shell:

- Commande : **mknod**

**mknod canal1 p**

- Commande : **mkfifo**

**mkfifo canal2**

sous réserve d'avoir les droits !

```
pi@raspberrypi:~/Documents/ExCoursTubes $ mkfifo canal1
pi@raspberrypi:~/Documents/ExCoursTubes $ ls -l
total 76
prw-r--r-- 1 pi pi    0 22 sept. 17:24 canal1
-rwxr-xr-x 1 pi pi 8436 22 sept. 16:54 ExTube
-rw-r--r-- 1 pi pi 1476 22 sept. 16:51 ExTubeAnonyme.c
-rwxr-xr-x 1 pi pi 8428 22 sept. 10:22 infosTube
-rwxr-xr-x 1 pi pi 8436 22 sept. 17:07 tubeanonyme
-rw-r--r-- 1 pi pi  943  8 sept.  2022 TubeAnonymeDup.c
-rw-r--r-- 1 pi pi 1056 22 sept. 10:22 TubeAnonymeInfos.c
-rwxr-xr-x 1 pi pi 8328 22 sept. 11:20 Tubedup
-rwxr-xr-x 1 pi pi 8296 22 sept. 17:00 tubeplein
-rw-r--r-- 1 pi pi 1177 22 sept. 16:59 tube_plein.c
pi@raspberrypi:~/Documents/ExCoursTubes $
```

# Programmation Système: les tubes

## Écriture/lecture d'un tube nommé:

- La lecture d'un tube nommé sans rédacteur est bloquante.
- L'écriture dans un tube nommé sans lecteur est bloquante.

Vérification sous le shell:

### Exemple:

- le tube canal1 étant créé, on peut l'ouvrir et lire dedans:

```
pi@raspberrypi:~/Documents/ExCoursTubes $ cat canal1 &  
[1] 3228  
pi@raspberrypi:~/Documents/ExCoursTubes $
```

Comme le tube est encore vide, l'ouverture sera suspendue d'où le "&" pour garder la main.

# Programmation Système: les tubes

Suite:

```
pi@raspberrypi:~/Documents/ExCoursTubes $ ls -l > canal1
total 80
-rw-r--r-- 1 pi pi 561 22 sept. 17:34 cab
prw-r--r-- 1 pi pi 0 22 sept. 17:24 canal1
-rwxr-xr-x 1 pi pi 8436 22 sept. 16:54 ExTube
-rw-r--r-- 1 pi pi 1476 22 sept. 16:51 ExTubeAnonyme.c
-rwxr-xr-x 1 pi pi 8428 22 sept. 10:22 infosTube
-rwxr-xr-x 1 pi pi 8436 22 sept. 17:07 tubeanonyme
-rw-r--r-- 1 pi pi 943 8 sept. 2022 TubeAnonymeDup.c
-rw-r--r-- 1 pi pi 1056 22 sept. 10:22 TubeAnonymeInfos.c
-rwxr-xr-x 1 pi pi 8328 22 sept. 11:20 Tubedup
-rwxr-xr-x 1 pi pi 8296 22 sept. 17:00 tubeplein
-rw-r--r-- 1 pi pi 1177 22 sept. 16:59 tube_plein.c
[1]+  Fini                cat canal1
pi@raspberrypi:~/Documents/ExCoursTubes $
```

Écriture dans le tube du résultat de `ls -l` et déblocage de la lecture (commande `cat`)

# Programmation Système: les tubes

## Exemple 2:

Le processus écrivain doit attendre un éventuel lecteur : écriture dans le tube canal

```
pi@raspberrypi:~/Documents/ExCoursTubes $ ls -l > canal1 &  
[1] 3270  
pi@raspberrypi:~/Documents/ExCoursTubes $
```

Le tube canal est toujours vide

```
pi@raspberrypi:~/Documents/ExCoursTubes $ ls -l canal1  
prw-r--r-- 1 pi pi 0 22 sept. 17:40 canal1  
pi@raspberrypi:~/Documents/ExCoursTubes $
```

Lecture dans le tube canal

```
pi@raspberrypi:~/Documents/ExCoursTubes $ cat canal1  
total 80  
-rw-r--r-- 1 pi pi 561 22 sept. 17:34 cab  
prw-r--r-- 1 pi pi 0 22 sept. 17:40 canal1  
-rwxr-xr-x 1 pi pi 8436 22 sept. 16:54 ExTube  
-rw-r--r-- 1 pi pi 1476 22 sept. 16:51 ExTubeAnonyme.c  
-rwxr-xr-x 1 pi pi 8428 22 sept. 10:22 infosTube  
-rwxr-xr-x 1 pi pi 8436 22 sept. 17:07 tubeanonyme  
-rw-r--r-- 1 pi pi 943 8 sept. 2022 TubeAnonymeDup.c  
-rw-r--r-- 1 pi pi 1056 22 sept. 10:22 TubeAnonymeInfos.c  
-rwxr-xr-x 1 pi pi 8328 22 sept. 11:20 Tubedup  
-rwxr-xr-x 1 pi pi 8296 22 sept. 17:00 tubeplein  
-rw-r--r-- 1 pi pi 1177 22 sept. 16:59 tube_plein.c  
pi@raspberrypi:~/Documents/ExCoursTubes $
```

*Un tube nommé étant référencé dans le système de fichiers, on peut le détruire grâce à la commande **rm** ou la commande **unlink***

# Programmation Système: les tubes

## Création d'un tube en langage C :

➤ Deux possibilités :

**int mknod(const char \*pathname, mode\_t mode, dev\_t dev)**

- ❖ pathname : le nom du tube,
- ❖ mode : droits et type,
- ❖ dev : sans importance pour un tube.

S\_IFIFO pour un tube

**int mkfifo ( const char \*pathname, mode\_t mode)**

- ❖ pathname : le nom du tube,
- ❖ mode : droits.

### Attention :

Les droits sont limités par umask.

# Programmation Système: les tubes

## Ouverture d'un tube nommé en C :

Utilisation de la fonction open :

**int open(const char \*pathname, int flags)**

- Pour l'ouvrir en lecture, il faut avoir le droit r. Pour l'ouvrir en écriture, il faut avoir le droit w.
- L'ouverture en lecture seule est bloquante tant qu'il n'y a pas de rédacteur.
- L'ouverture en écriture seule est bloquante tant qu'il n'y a pas de lecteur.
- Comme avec les tubes anonymes : **attention à l'inter-blocage.**



# Programmation Système: les tubes

## Lecture dans un tube nommé:

Tube non vide (contient n caractères)

- `read(fd, buffer, nb)`
  - ❖  $nb \leq n$  : lecture de nb caractères (lecture partielle)
  - ❖  $nb > n$  : lecture de n caractères (lecture totale)

Tube vide :

- lecture non bloquante : `read` retourne -1,
- lecture bloquante : `read` est bloqué jusqu'à ce qu'il y ait des données dans le tube.

# Programmation Système: les tubes

## Écriture dans un tube nommé :

### ➤ Pas de lecteur :

Le processus reçoit un signal SIG\_PIPE qui par défaut provoque sa destruction

### ➤ Au moins 1 lecteur :

- ❖ il y a assez de place dans le tube : écriture atomique du message,
- ❖ il n'y a pas assez de place dans le tube :
  - Écriture non bloquante : write retourne -1 sans écriture,
  - Écriture bloquante : attente de place puis écriture atomique du message.

# Programmation Système: les tubes

## Destruction d'un tube nommé en C :

- Utilisation de la fonction unlink :

**int unlink(const char \*pathname)**

- ❖ pathname : le nom du tube.

## Mise en œuvre d'un exemple :

- un processus crée un tube, l'ouvre en lecture seule, lit deux fois et détruit le tube,
- l'autre processus ouvre le tube en écriture seule, écrit deux messages puis se termine.

# Programmation Système: les tubes

## Processus 1

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(void)
{
    mode_t mode;
    int fd;
    int nbre;
    char Lecture[100];
    mode = S_IRUSR | S_IWUSR;
    if (mkfifo("Tubel",mode) == -1)
    {
        perror("mkfifo");
        return EXIT_FAILURE;
    }
    if ((fd = open("./Tubel",O_RDWR)) == -1)
    {
        perror("open");
        unlink("./Tubel");
        return EXIT_FAILURE;
    }
    printf("\nprocl : lecture dans le tube \n");
    nbre = read(fd,Lecture,100);
    if (nbre > 0)
    {
        Lecture[nbre] = 0;
        printf("\nprocl: Message reçu (%d caractères): %s\n",nbre,Lecture);
        nbre = read(fd,Lecture,100);
        Lecture[nbre] = 0;
        printf("\nprocl: Nouveau message reçu (%d caractères):%s\n",nbre,Lecture);
        close(fd);
        unlink("Tubel");
        printf("Fini\n");
        return EXIT_SUCCESS;
    }
    perror("read");
    close(fd);
    unlink("./Tubel");
    return EXIT_FAILURE;
}
```

# Programmation Système: les tubes

## Processus 2

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <nom du tube>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int fd;
    char Texte[60] = "Ecriture d'un message dans le tube pour un autre processus \n";
    if ((fd = open(argv[1], O_RDWR)) == -1)
    {
        perror("open");
        return EXIT_FAILURE;
    }

    printf("\nproc2 : Ecriture dans le tube \n");
    write(fd, Texte, strlen(Texte));
    sleep(2);
    strcpy(Texte, "Un autre message de quelques caractères\n");
    printf("\nproc2 : Ecriture du second message dans le tube \n");
    write(fd, Texte, strlen(Texte));
    close(fd);
    return EXIT_SUCCESS;
}
```

# Programmation Système: les tubes

## Exécution :

```
pi@raspberrypi:~/Documents/ExCoursTubes $ ./proc1

proc1 : lecture dans le tube

proc1: Message reçu (61 caractères): Ecriture d'un message dans le tube pour un autre proces
sus

proc1: Nouveau message reçu (41 caractères):Un autre message de quelques caractères

Fini
pi@raspberrypi:~/Documents/ExCoursTubes $
```

```
pi@raspberrypi:~/Documents/ExCoursTubes $ ./proc2 Tube1

proc2 : Ecriture dans le tube

proc2 : Ecriture du second message dans le tube
pi@raspberrypi:~/Documents/ExCoursTubes $
```