



Copyright 2010 tv <tvaira@free.fr>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License,

Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover.

You can obtain a copy of the GNU General Public License : write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

SOMMAIRE

Les Tubes.....	3
Les tubes anonymes.....	4
Création d'un tube.....	4
Communication.....	7
Obtention d'un flux.....	7
Obtention d'un descripteur.....	8
Redirections.....	8
Les tubes nommés.....	10
Création d'un tube nommé.....	10
Ouverture d'un tube nommé.....	10
Suppression d'un tube nommé.....	10
Manipulations.....	13

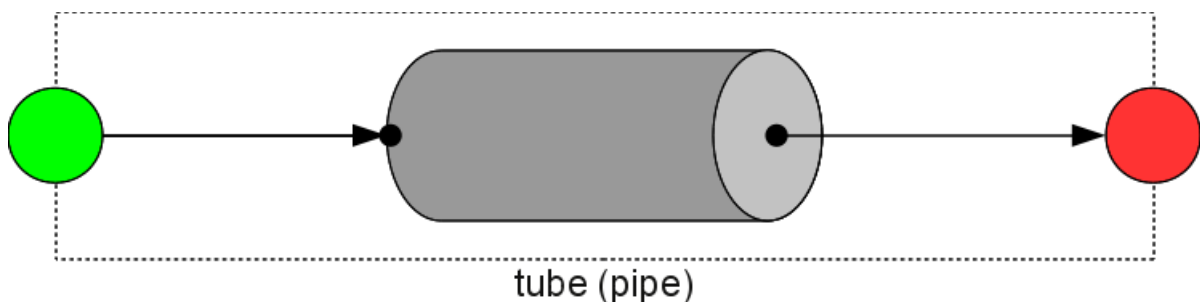
LES TUBES

Les tubes sont un mécanisme de communication entre processus résidant sur une même machine. On peut en distinguer 2 catégories :

- les **tubes anonymes** (volatiles) : Ils concernent des processus issus de la même application
- les **tubes nommés** (persistants) : Ils concernent des processus totalement indépendants

Les principales caractéristiques des tubes sont :

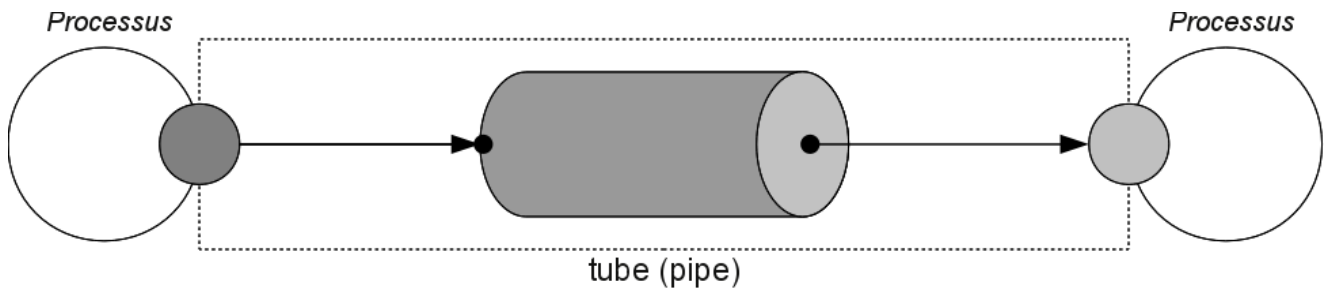
- intra-machine
- communication synchrone
- communication unidirectionnelle
- communication en mode flux (stream)
- limité en taille
- stratégie de remplissage et de vidage en FIFO
- volatilité ou persistance (tube anonyme ou tube nommé)



LES TUBES ANONYMES

Création d'un tube

Un tube peut être vu comme un tuyau dans lequel un processus, à une extrémité, produit des informations, tandis qu'à l'autre extrémité un autre processus les consomme.



Un tube est créé par l'appel système `pipe()` :

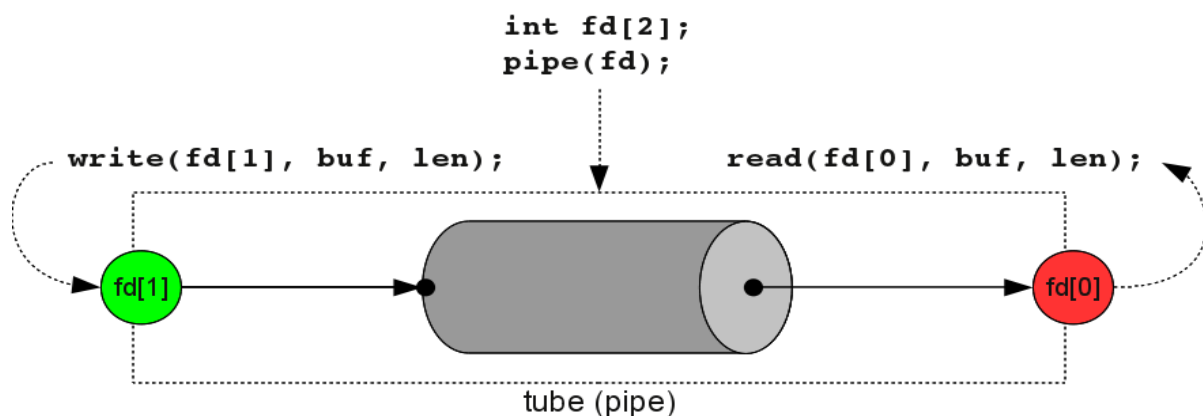
```
#include <unistd.h>

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

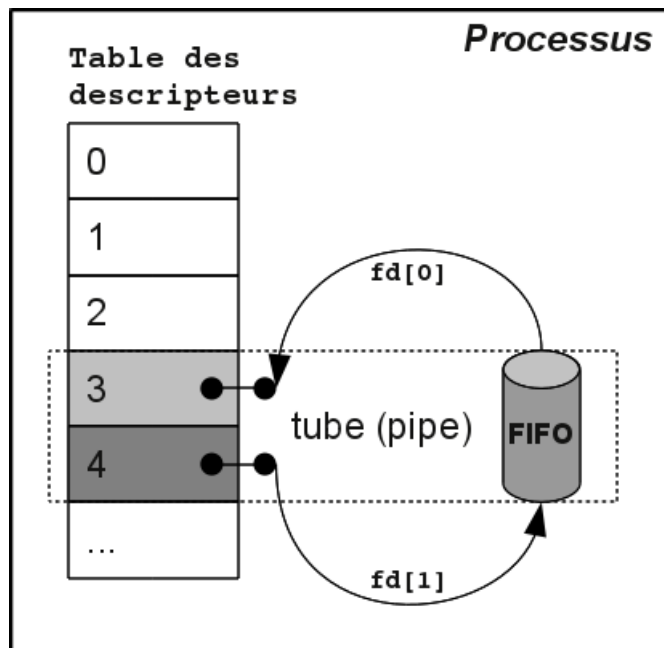
Cette fonction, lorsqu'elle réussit retourne 0 et remplit le tableau descripteur avec les descripteurs de chacune des extrémités du tube : l'entrée et la sortie.

- Le descripteur 0 correspond à la sortie du tube. Il est ouvert en lecture seule.
- Le descripteur 1 correspond à l'entrée du tube. Il est ouvert en écriture seule.

En cas d'échec, `pipe()` retourne -1.



La fonction `pipe()` a pour effet de créer un inode dans le système de gestion de fichier et d'ajouter 2 entrées dans la table des descripteurs de fichiers.

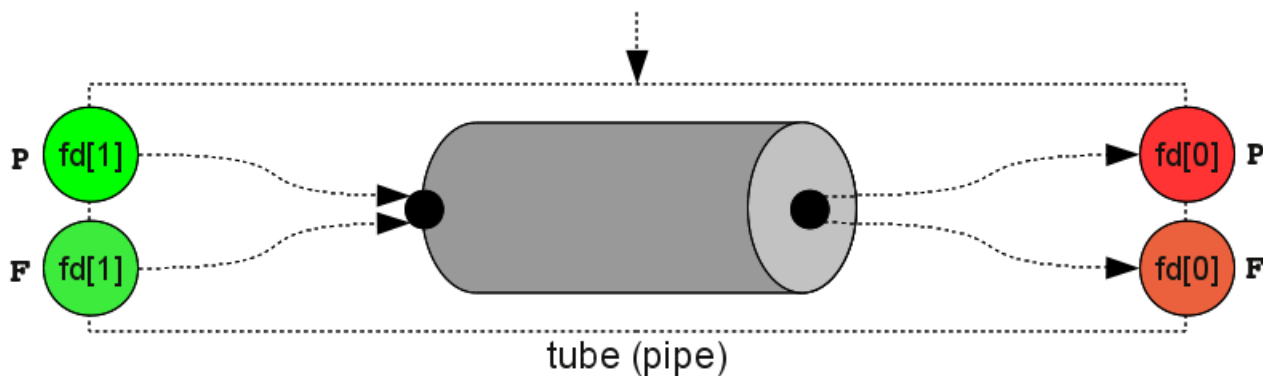


```
# lsof -p 22050
COMMAND    PID  USER   FD   TYPE    DEVICE  SIZE/OFF      NODE NAME
tube1      22050  tv     0u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22050  tv     1u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22050  tv     2u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22050  tv     3r   FIFO     0,8      0t0 20133335 pipe
tube1      22050  tv     4w   FIFO     0,8      0t0 20133335 pipe
```

// Et pour son fils (après un fork) :

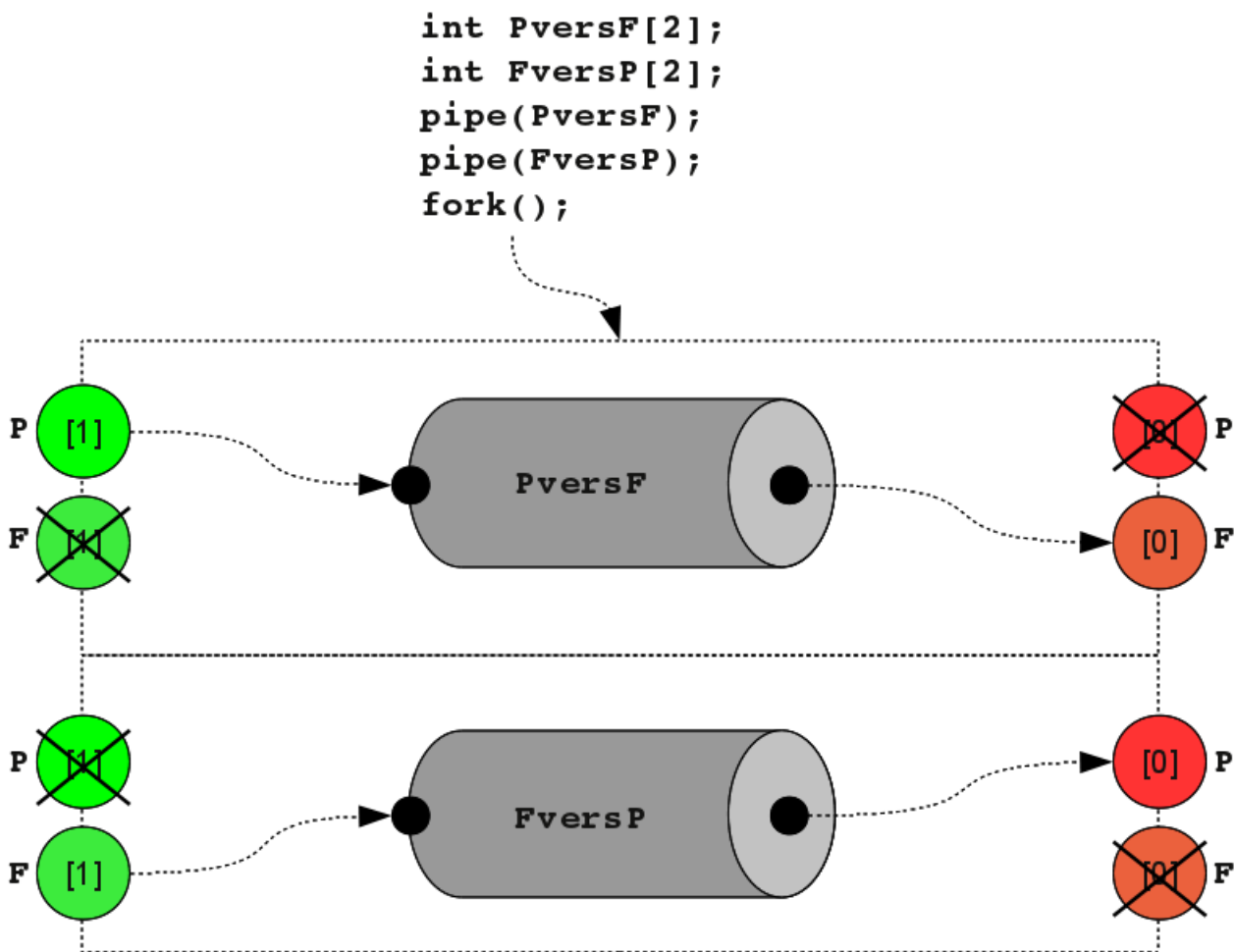
```
# lsof -p 22051
COMMAND    PID  USER   FD   TYPE    DEVICE  SIZE/OFF      NODE NAME
tube1      22051  tv     0u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22051  tv     1u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22051  tv     2u   CHR     136,3    0t0        6 /dev/pts/3
tube1      22051  tv     3r   FIFO     0,8      0t0 20133335 pipe
tube1      22051  tv     4w   FIFO     0,8      0t0 20133335 pipe
```

```
int fd[2];
pipe(fd);
fork();
```



Les tubes étant des systèmes de communication unidirectionnels, ils faut créer deux tubes et les employer dans le sens opposé si l'on souhaite réaliser une communication bidirectionnelle entre 2 processus.

La création de ces deux tubes est sous la responsabilité du processus père. Ensuite, la primitive `fork()` réalisant une copie des données dans le processus fils, il suffit pour le père et le fils de fermer les extrémités de tubes qu'ils n'utilisent pas :



```

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

int main()
{
    int tubePversF[2], tubeFversP[2];

    if ((pipe(tubePversF) != 0) || (pipe(tubeFversP) != 0))
    {
        perror("Création du tube impossible");
        exit(-1);
    }
}

```

```
switch (fork())
{
case -1 : perror("Erreur du fork"); exit(-1);
case 0  : /* fils */
        close(tubePversF[1]);
        close(tubeFversP[0]);
        ...
default : /* pere */
        close(tubePversF[0]);
        close(tubeFversP[1]);
        ...
}
```

Communication

L'écriture dans un tube se fait grâce à la primitive `write()`.

La lecture dans un tube se fait grâce à la primitive `read()`.

La fermeture d'un flux se fait à l'aide de `close()`.

Remarque :

Il est à noter que ne pas fermer un flux peut entraîner un blocage.

Obtention d'un flux

```
#include <stdio.h>

FILE *fdopen(int descripteur, const char *mode);
FILE *popen(const char *commande, const char *mode);
int pclose(FILE *flux);
```

La fonction `fdopen()` retourne un flux associé au descripteur passé en paramètre. Le mode du flux doit être compatible avec celui du descripteur.

La fonction `popen()` exécute la commande passée en paramètre, crée un tube entre les deux processus et transforme une extrémité du tube en un flux qu'il retourne. L'écriture dans un flux ouvert avec `popen()` correspond à écrire sur le `stdin` de la commande. Symétriquement, la lecture dans le flux correspond à lire sur le `stdout` de la commande.

Un flux ouvert avec `popen()` doit être fermé avec `pclose()`.

Exemple : fdopen

```
int tube[2];
FILE *flux;
...
pipe(tube);
```

```
close(tube[0]);
flux = fdopen(p[1], "w");
fprintf(flux, "Une autre façon d'écrire dans un tube\n");
```

Exemple : popen

```
FILE *flux;
int heure, minutes;
...
flux = popen("date +%Hh%D", "r");
fscanf(flux, "%dh%d",&heure, &minutes);
pclose(flux);
```

Obtention d'un descripteur

```
#include <stdio.h>

int fileno(FILE *flux);
```

La fonction `fileno()` retourne le numéro de descripteur correspondant au flux passé en paramètre.

Redirections

Les redirections ont pour effet de rediriger un flux vers un autre flux, typiquement les flux standards d'entrée/sortie : `stdin`, `stdout` et `stderr`.

La mise en œuvre d'une redirection peut se faire simplement à l'aide des primitives `dup()` et `dup2()` :

```
#include <unistd.h>

int dup(int descripteur);
int dup2(int nouveau, int ancien);
```

La fonction `dup()` a pour effet d'effectuer une copie du descripteur passé en paramètre, pointant sur la même structure que l'original. Cet appel système garantit que la copie sera effectuée dans la première entrée disponible dans la table des descripteurs. C'est ce numéro d'entrée qui est retourné par la fonction.

Le défaut de ce procédé réside dans le risque qu'un signal interrompe le processus entre la fermeture du descripteur à rediriger et la duplication du descripteur cible.

La fonction `dup2()` vient pallier ce problème.

Elle permet de fermer le descripteur ancien s'il est ouvert, puis de dupliquer nouveau en lui associant une nouvelle entrée à la position ancien dans la table des descripteurs.

Cette fonction réalise l'opération de façon atomique sans présumer de la numérotation effective des flux.

Exemple : redirection de stdout

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

int main() {
    int fd;

    // ouverture du fichier
    FILE *f = fopen("resulDup.txt", "w+");

    // obtention de son numéro de descripteur
    fd = fileno(f);

    if (fd < 0) {
        perror("open");
        exit(-1);
    }

    // fermeture de stdout
    close(STDOUT_FILENO);

    // duplication du descripteur de fd dans celui de stdout (1)
    if (dup(fd) < 0) {
        perror("dup");
        exit(-1);
    }

    // fermeture du descripteur inutilisé
    close(fd);

    // commutation d'image
    execlp("ls", "ls", NULL);
    perror("execlp");
    exit(-1);
}
```

LES TUBES NOMMÉS

Un tube nommé est simplement un nœud dans le système de fichiers. Le concept de tube a été étendu pour disposer d'un nom dans ce dernier. Ce moyen de communication disposant d'une représentation dans le système de fichier, il peut être utilisé par des processus indépendants.

Création d'un tube nommé

La création d'un tube nommé se fait à l'aide de la fonction `mkfifo()`.

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

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

Cette fonction retourne 0 si elle réussit et -1 sinon.

Le premier paramètre est le nom du tube (chemin). Le second paramètre `mode` représente les droits d'accès.

Ouverture d'un tube nommé

Une fois le nœud créé, on peut l'ouvrir avec `open()` avec les restrictions dues au mode d'accès.

Si le tube est ouvert en lecture, `open()` est bloquante tant qu'un autre processus ne l'a pas ouvert en écriture. Symétriquement, une ouverture en écriture est bloquante jusqu'à ce que le tube soit ouvert en lecture.

Suppression d'un tube nommé

La suppression d'un tube nommé s'effectue avec `unlink()`.

Exemple : Création d'un tube nommé et lecture dedans

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

int main() {
    int tubeRD;
    FILE *f;
    char message[100];

    if (mkfifo("tn", 0644) != 0) {
        perror("Erreur mkfifo");
        exit(-1);
    }
    printf("Création du noeud tn\n");

    tubeRD = open("tn", O_RDONLY);
    if (tubeRD < 0) {
        perror("open");
        exit(-1);
    }
    printf("Ouverture du noeud tn en lecture\n");

    f = fdopen(tubeRD, "r");
    printf("Ouverture d'un flux en lecture sur tn\n");

    fscanf(f, "%s", message);
    printf("Message reçu : %s\n", message);

    fclose(f);
    printf("Fermeture du flux\n");

    close(tubeRD);
    printf("Fermeture du tube\n");

    sleep(1);
    unlink("tn");
    printf("Suppression du noeud\n");

    return 0;
}
```

Exemple : Ouverture d'un tube nommé et écriture dedans

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

int main() {
    int tubeRD;
    FILE *f;

    tubeRD = open("tn", O_WRONLY);
    if (tubeRD < 0) {
        perror("open");
        exit(-1);
    }
    printf("Ouverture du noeud tn en ecriture\n");

    f = fdopen(tubeRD, "w");
    printf("Ouverture d'un flux en ecriture sur tn\n");

    fprintf(f, "OK\n");
    printf("Message envoye : OK\n");

    fclose(f);
    printf("Fermeture du flux\n");

    close(tubeRD);
    printf("Fermeture du tube\n");

    return 0;
}
```

MANIPULATIONS

1) Compiler et tester les exemples fournis.

Tubes anonymes

- `tube1.c` : communication bidirectionnelle (père-fils) avec un seul tube
- `tube2.c` : communication bidirectionnelle (père-fils) avec deux tubes
- `tube3.c` : exécute " `who | cut -c 1-8 | sort -u` "

Tubes nommés

`tubeN_lecture.c` : création d'un tube nommé et lecture dedans

`tubeN_ecriture.c` : ouverture d'un tube nommé et écriture dedans

Exécution :

```
$ ./tubeN_lecture &

$ ls -l
prw-r--r-- 1 tv tv    0 2011-01-17 16:04 tn|

$ ./tubeN_ecriture &
...
```