Cours "Programmation Système"

R3.05

2ème année BUT de Caen

Département d'informatique

les tubes

Table des matières

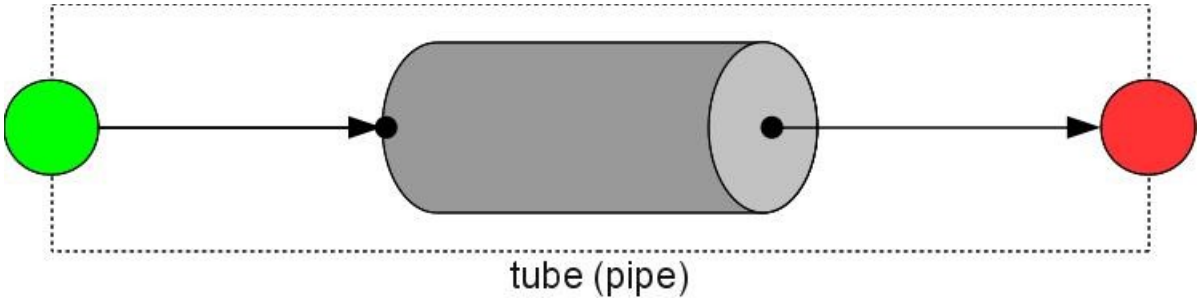
I/ Introduction : 3
II/caractéristiques d'un tube 3
II-1/ Tubes de processus 4
II-2/ tubes anonymes (ou ordinaires ou non nommés) 5
II-2-1/ Création d'un tube anonyme 5
II-2-2/Accès aux caractéristiques du tube anonyme créé 6
II-2-3/ Lecture dans un tube anonyme 7
II-2-4/ Ecriture dans un tube anonyme 8
II-2-5/ Duplication des descripteurs 9
II-3/Les tubes nommés 10
II-3-1/Gestion des tubes nommés sous le shell 11
II-3-2/Création d'un tube nommé en C 13
II-3-3/ Ouverture d'un tube nommé en C 14
II-3-4/ Lecture-écriture-fermeture et suppression d'un tube nommé en C 15

Partie 2 :

I/ Introduction :

La sortie standard d'une commande peut servir d'entrée standard à une autre commande. Par exemple si on exécute la commande ***ls -l | more*** deux processus sont créés et le résultat de ***ls -l*** sert d'entrée pour ***more***. En effet, le système créé un tube de communication (***pipe***) dans lequel le résultat du premier processus est écrit et dans lequel le second processus lit. 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.



II/caractéristiques d'un tube

On distingue deux catégories de tubes :

- 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 :

- La gestion des tubes est intégrée dans le système de fichiers. On peut donc accéder aux tubes par des descripteurs de fichiers ; un descripteur pour la lecture et un descripteur pour l'écriture.
- Les tubes disposent de deux extrémités une pour la lecture l'autre pour l'écriture (d'où la nécessité des deux descripteurs).
- Les tubes permettent de gérer un flot continu de données : une seule lecture peut extraire les données de plusieurs écritures.
- La lecture est destructrice : une information ne peut être lue qu'une seule fois.
- La gestion des tubes est assurée en mode FIFO : la première donnée écrite sera la première lue.
- Un tube a une taille limitée, il pourra donc être plein et rendre les opérations d'écriture bloquantes.

- Un tube peut avoir plusieurs lecteurs et plusieurs rédacteurs. Si un tube n'a pas de lecteur, l'écriture dedans est interdite.

II-1/ Tubes de processus

Le lancement d'une commande est une première utilisation facile des tubes. La fonction ***FILE *popen(const char* commande, const char* type);*** lance un processus exécutant la commande ***commande*** et relie ce processus par un tube unidirectionnel (lecture seule ou écriture seule selon le paramètre ***type***). Les deux processus s'exécutent en parallèle.

Exemple1 :

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

int main()
{
   FILE *Lire=NULL;
   char Valeur[BufferSize];
   int nbr;
   Lire = popen("cat /etc/passwd | wc -l","r");
   if(Lire != NULL)
   {
      nbr= fread(Valeur, sizeof(char),BufferSize,Lire);
      if(nbr>0)
      {
         printf("Nombre de lignes du fichier /etc/passwd : %s\n",Valeur);
      }
      pclose(Lire);
      return EXIT_SUCCESS;
   }
   return EXIT_FAILURE;
}
```

*La fonction **popen** a lancé un processus qui a exécuté la commande **cat /etc/passwd | wc -l** qui compte le nombre de lignes du fichier **/etc/passwd** et qui écrit le résultat dans le tube. La fonction **read** a lu le résultat dans le tube puis ce résultat a été affiché. Sous le shell la commande donne bien le même résultat.*

Exemple2 :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   FILE *Ecrire=NULL;
   char Texte[BufferSize]="41*94\n";
   Ecrire = popen("bc","w");
   if(Ecrire != NULL)
   {
```

```
        fwrite(Texte, sizeof(char),strlen(Texte),Ecrire);
        fwrite("quit",sizeof(char),strlen("quit"),Ecrire);
        pclose(Ecrire);
        return EXIT_SUCCESS;
    }
    return EXIT_FAILURE;
}
```

Dans cet exemple, la fonction ***popen*** a lancé un processus qui a exécuté la commande ***bc*** (qui lance une calculatrice) et qui attend des données dans le tube. La fonction ***fwrite*** a écrit les données dans le tube puis la calculatrice a affiché le résultat.

L'appel de la fonction ***popen*** *lance le programme cité en invoquant un shell auquel elle passe la chaine de commande. Il est donc possible d'utiliser les capacités du shell mais cette utilisation est très couteuse en ressources système.*

II-2/ tubes anonymes (ou ordinaires ou non nommés)

Les tubes ordinaires sont associés par un inode au système de fichiers, mais cet inode a un compteur de lien nul ce qui implique qu'aucun répertoire ne contient de référence à ce fichier. Un tube ordinaire se comporte comme un fichier sans nom ; il sera libéré lorsque plus aucun processus ne l'utilisera. Comme il n'a pas de nom, on ne pourra pas l'ouvrir avec la fonction ***open***. La connaissance de l'existence de ce fichier que par la possession de descripteurs de fichiers (un pour l'écriture et un pour la lecture). Cette connaissance peut être réalisée de deux manières :

- ✓ Le processus est créateur du tube,
- ✓ Le processus hérite des descripteurs car son processus père les possède.

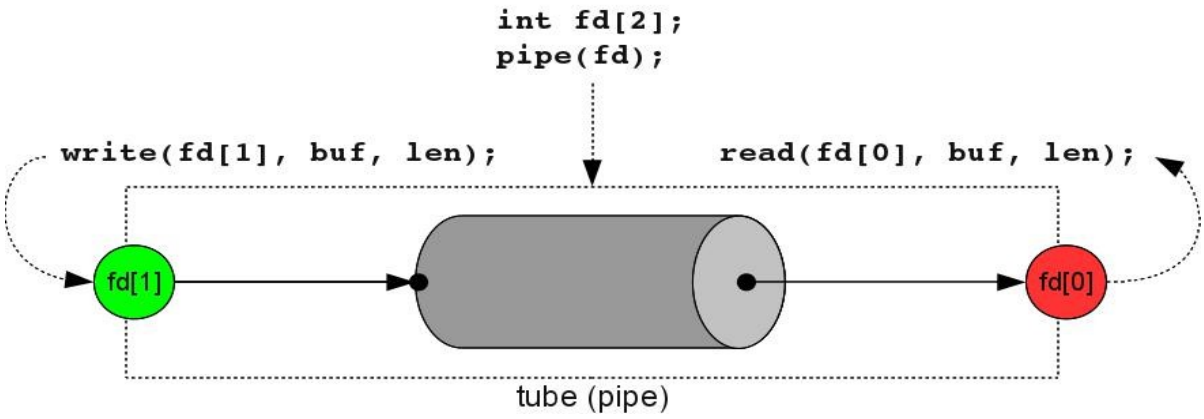
Cette connaissance des descripteurs impose deux conséquences :

- ✓ La communication par l'intermédiaire de ces tubes est réservée à des processus ayant été créés après le tube pour pouvoir en hériter.
- ✓ Un processus qui a perdu un accès dans un des deux modes ne peut plus obtenir de nouvel accès.

II-2-1/ Création d'un tube anonyme

La création d'un tube anonyme est réalisée par la fonction ***int pipe(int filedes[2])***.

La création correspond à l'allocation d'un inode et à la création d'une paire de descripteurs de fichiers pointant sur le tube créé : ***filedes[0]*** est utilisé pour la lecture et ***filedes[1]*** est utilisé pour l'écriture. En général deux processus, créés par ***fork()*** vont se partager le tube et utiliser les fonctions ***read*** et ***write*** pour s'échanger des données.



II-2-2/Accès 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 de dernier accès,
- …

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

Exemple :

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

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 proprietaire %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;
}
```

II-2-3/ Lecture dans un tube anonyme

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.

Remarque : lors du développement d'applications utilisant les tubes, il faut faire attention aux situations de blocage :

- 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));
 }
```

Il faut donc ne conserver que les descripteurs utiles et fermer systématiquement les autres.

II-2-4/ Ecriture dans un tube anonyme

La fonction ***write*** permet de réaliser l'écriture.

Si le nombre de lecteurs dans le tube est nul, le processus rédacteur reçoit un signal ***SIGPIPE*** qui par défaut provoque sa terminaison. En effet s'il n'y a plus de lecteurs il n'y en aura plus d'autres et il n'y a plus de raison d'y avoir de rédacteurs.

S'il existe au moins un lecteur :

- Si l'écriture est bloquante, le retour de la fonction ***write*** n'a lieu que lorsque tous les caractères ont été écrits. Le processus rédacteur est donc susceptible de passer à l'état endormi dans l'attente que le tube se vide.
- Si l'écriture n'est pas bloquante :
  - S'il reste de la place pour écrire, une écriture atomique est réalisée.
  - S'il ne reste pas assez de place pour écrire, le retour est immédiat et sans écriture, il y a une erreur.

Les processus qui n'écrivent pas dans un tube ne doivent pas garder de descripteur d'écriture afin de ne pas provoquer les situations de blocage.

II-2-5/ Duplication des descripteurs

Deux fonctions permettent de dupliquer les descripteurs et donc permettent d'augmenter le nombre de lecteurs ou de rédacteurs.

La fonction ***int dup(int oldfd)*** créé une copie du descripteur passé en paramètre. Elle utilise le plus petit numéro inutilisé pour le descripteur.

La fonction ***int dup2(int oldfd, int newfd)*** transforme ***newfd*** en une copie de ***oldfd*** fermant auparavant ***newfd si*** besoin est.

Après l'utilisation de ces fonctions, l'ancien et le nouveau descripteur peuvent être utilisés de manière interchangeable. Il est donc possible de rediriger un descripteur de tube vers l'entrée standard (***stdin***), la sortie standard (***stdout***) ou la sortie d'erreur (***stderr***).

Remarque : On peut également utiliser la fonction ***fcntl*** avec le paramètre ***F_DUPFD*** pour dupliquer les descripteurs.

Exemple :

```
#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,"Nombre de fichiers du repertoire /etc: \n");
            execlp("ls","ls","-l",NULL);
            perror("execlp");
            return EXIT_FAILURE;
  }
}
```

Dans cet exemple, le père crée un tube puis fork un fils. Le père ferme la sortie standard, duplique le descripteur de fichier, ferme le descripteur d'écriture (il est dupliqué) puis recouvre son code avec la commande ls -l. la sortie standard écrira donc ses données dans le tube créé.

Le fils ferme l'entrée standard, duplique le descripteur de lecture, ferme le descripteur de lecture (dupliqué) puis recouvre son code sur la commande wc -l (qui compte le nombre de lignes). Il lira donc ses informations à partir du tube de communication et comptera le nombre de lignes de la commande ls -l.

Remarque :

Lorsque la sortie standard est fermée puis ré ouverte par duplication, les fonctions d'écriture sur la sortie standard écrivent maintenant dans un tube. Pour informer l'utilisateur, il est encore possible (dans le cas présent) d'écrire sur la sortie d'erreur standard (fprintf(stderr,"…")).

II-3/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.

Contrairement aux tubes ordinaires, ils possèdent donc une référence dans le système de fichiers et sont vus 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 connaitre le nom du tube.

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

II-3-1/Gestion des tubes nommés sous le shell

Un tube nommé peut être créé sous la shell par la commande ***mknod*** ou la commande ***mkfifo;*** sous réserve d'avoir les droits.

Exemple1 :

```
pi@raspberrypi:~ $ mkfifo canal
pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Bookshelf
prw-r--r-- 1 pi pi    0 oct.   2 14:59 canal
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Desktop
drwxr-xr-x 2 pi pi 4096 sept. 26 21:42 Documents
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Downloads
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Music
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Pictures
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Public
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Templates
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Videos
```

On peut maintenant ouvrir le tube par la commande ***cat*** et lire dedans.

```
pi@raspberrypi:~ $ cat canal &
[1] 1407
```

Comme le tube est encore vide, l'ouverture sera suspendue d'où le "&" pour garder la main. Il faut un écrivain qui ouvre le tube et écrive dedans. Par exemple la commande ls -l aura pour effet de relancer la commande cat suspendue.

```
pi@raspberrypi:~ $ ls -l > canal
total 36
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Bookshelf
prw-r--r-- 1 pi pi    0 oct.   2 14:59 canal
prw-r--r-- 1 pi pi    0 oct.   2 15:06 canal2
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Desktop
drwxr-xr-x 2 pi pi 4096 sept. 26 21:42 Documents
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Downloads
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Music
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Pictures
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Public
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Templates
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Videos
[1]+  Fini                    cat canal
pi@raspberrypi:~ $
```

On peut noter que le tube *canal* est vidé à la suite de la commande cat.

Exemple2 :

```
pi@raspberrypi:~ $ mknod canal2 p
pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Bookshelf
prw-r--r-- 1 pi pi    0 oct.   2 14:59 canal
prw-r--r-- 1 pi pi    0 oct.   2 15:06 canal2
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Desktop
drwxr-xr-x 2 pi pi 4096 sept. 26 21:42 Documents
```

Exemple3 :

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

```
pi@raspberrypi:~ $ ls -l >canal&
[1] 1708
pi@raspberrypi:~ $
```

Canal toujours vide

```
pi@raspberrypi:~ $ ls -l canal
prw-r--r-- 1 pi pi 0 oct.   2 15:17 canal
pi@raspberrypi:~ $
```

Lecture dans le canal

```
pi@raspberrypi:~ $ cat canal
total 36
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Bookshelf
prw-r--r-- 1 pi pi    0 oct.   2 15:17 canal
prw-r--r-- 1 pi pi    0 oct.   2 15:06 canal2
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Desktop
drwxr-xr-x 2 pi pi 4096 sept. 26 21:42 Documents
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Downloads
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Music
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Pictures
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Public
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Templates
drwxr-xr-x 2 pi pi 4096 janv. 11  2021 Videos
[1]+  Fini                    ls --color=auto -l > canal
pi@raspberrypi:~ $
```

II-3-2/Création d'un tube nommé en C

La création d'un tube nommé en langage C se fait à l'aide des fonctions :

- ***mknod***

int mknod(const char *pathname, mode_t mode, dev_t dev);

- ✓ **pathname :** nom du tube
- ✓ **mode :** droits et type
- ✓ **dev :** sans importance pour un tube

- ***mkfifo***

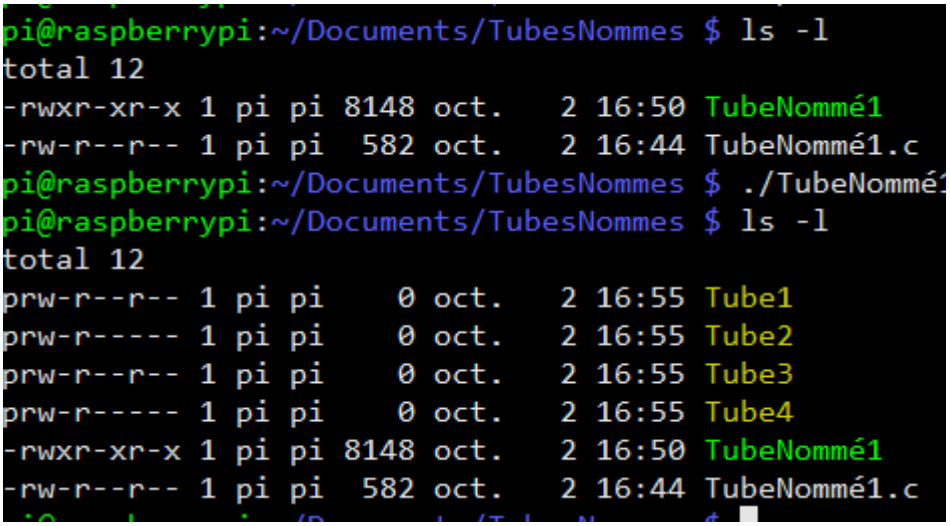
int mkfifo(const char *pathname, mode_t mode);

- ✓ **pathname :** nom du tube
- ✓ **mode :** droits

Exemple :

```
#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;
        mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWOTH;
        if (mkfifo("./Tube1",0666) == -1)
        {
                perror("mkfifo");
                return EXIT_FAILURE;
        }
        if (mkfifo("./Tube2",mode) == -1)
        {
                perror("mkfifo");
                return EXIT_FAILURE;
        }
        if (mknod("./Tube3",0666 | S_IFIFO,0) == -1)
        {
                perror("mkfifo");
                return EXIT_FAILURE;
        }
        if (mknod("./Tube4",mode | S_IFIFO,0) == -1)
        {
        perror("mkfifo");
        return EXIT_FAILURE;
        }
        return EXIT_SUCCESS;
}
```

L'exécution de cet exemple donne :



On peut voir que les 4 tubes nommés ont été créés et que les droits sont limités par la valeur de ***umask***.

II-3-3/ Ouverture d'un tube nommé en C

Pour qu'un processus puisse ouvrir un tube, il doit avoir les droits correspondants pour la lecture ou l'écriture des données dans ce tube. Par défaut la fonction open est bloquante.

- ✓ Une demande d'ouverture en lecture bloque le processus s'il n'y a pas de rédacteur.
- ✓ Une demande d'ouverture en écriture bloque le processus s'il n'y a pas de lecteur.

int open(char* pathname, int flags)

Une synchronisation est donc réalisée par système et permet ainsi à deux processus de se donner rendez-vous en un point particulier de leur exécution.

Si deux processus dialoguent à travers deux tubes, un pour chaque sens de communication, si l'un des processus ouvre un tube en lecture et l'autre ouvre le deuxième tube lui aussi en lecture, ils seront tous les deux bloqués!!! Pour éviter ce genre d'inter blocage, un processus doit ouvrir un tube en lecture et l'autre processus doit ouvrir le même tube en écriture.

On peut faire une ouverture en mode non bloquant grâce à l'option O_NONBLOCK pour le paramètre *flags* de la fonction *open.* Dans ce cas :

- ✓ Une ouverture en lecture réussit même s'il n'y a pas de rédacteur.
- ✓ Une ouverture en écriture réussit s'il y a au moins un lecteur et échoue s'il n'y a pas de lecteur. L'absence de lecteur provoquera l'apparition d'un signal SIGPIPE lors de l'écriture.

Une ouverture non bloquante le restera jusqu'à la demande du contraire.

Si le tube est ouvert en mode "non bloquant", ni la fonction open ni une autre opération ultérieure sur ce tube ne laissera le processus en attente.

II-3-4/ Lecture-écriture-fermeture et suppression d'un tube nommé en C

Toutes les opérations déjà vues pour un tube anonyme sont réalisables sur un tube nommé. Les opérations de lecture, écriture et fermeture sont permises grâce aux fonctions ***read***, ***write*** et ***close***.

La fonction ***int unlink(const char* pathname)*** permet de détruire un tube nommé dans le système de fichiers même s'il n'est pas vide, les données sont évidemment perdues.

Exemple complet :

ProcTube1.c

```
#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("./Tube1",mode) == -1)
        {
                perror("mkfifo");
                return EXIT_FAILURE;
        }
        if ((fd = open("./Tube1",O_RDWR)) == -1)
        {
                perror("open");
                unlink("./Tube1");
                return EXIT_FAILURE;
        }
        nbre = read(fd,Lecture,100);
        if (nbre > 0)
        {
                Lecture[nbre] = 0;
                printf("Message reçu : %s\n",Lecture);
                sleep(10);
```

```
                nbre = read(fd,Lecture,100);
                Lecture[nbre] = 0;
                printf("Nouveau message (%d caractères):%s\n",nbre,Lecture);
                sleep(10);
                close(fd);
                sleep(10);
                unlink("./Tube1");
                printf("Fini\n");
                return EXIT_SUCCESS;
        }
        perror("read");
        close(fd);
        unlink("./Tube1");
        return EXIT_FAILURE;
}
```

ProcTube2.c

```
#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(void)
{
        nt fd;
        char Texte[60] = "Ecriture d'un message dans le tube pour un autre processus \n";
        if ((fd = open("./Tube1",O_RDWR)) == -1)
        {
                perror("open");
                return EXIT_FAILURE;
        }
        write(fd,Texte,strlen(Texte));
        sleep(1);
        strcpy(Texte,"Un autre message de quelques caractères\n");
        write(fd,Texte,strlen(Texte));
        close(fd);
        return EXIT_ SUCCESS;
}
```

L'exécution de l'exemple complet donne :

```
pi@raspberrypi:~/Documents/TubesNommes $ ./ProcTube1 &
[1] 1540
pi@raspberrypi:~/Documents/TubesNommes $ ./ProcTube2
Message reçu : Ecriture d'un message dans le tube pour un autre processus

pi@raspberrypi:~/Documents/TubesNommes $ Nouveau message (41 caractères):Un autr
e message de quelques caractères

Fini

[1]+  Fini                    ./ProcTube1
pi@raspberrypi:~/Documents/TubesNommes $
```

Le processus *ProcTube1* est lancé en premier car c'est lui qui crée le tube. Il est lancé en tâche de fond car il se bloque sur la lecture dans le tube. Le processus *ProcTube2* est ensuite lancé, il écrit un premier message dans le tube. Ce message est lu et affiché par *ProcTube1* qui après une courte pose écrit un second message et se termine.

- ✓ Avec la commande ps l on peut voir que le processus *ProcTube1* est endormi.
- ✓ Avec la commande ls -l on peut voir que le tube contient 41 octets.

Lorsque le processus *ProcTube1* se réveille, il lit à nouveau dans le tube et affiche le second message puis s'endort à nouveau.

- ✓ Une nouvelle commande ls -l permet de voir que le tube est vide.

Le processus *ProcTube1* se réveille à nouveau et détruit le tube.

- ✓ Une nouvelle commande ls -l permet de voir que le tube est détruit.