Cours 6
Tubes anonymes et nommés

Cours 6 : Tubes anonymes et nommés

- **Mécanisme de communications du système de fichiers**
  - I-node associé.
  - Type de fichier: S_IFIFO.
  - Accès au travers des primitives ***read*** et ***write***.
- **Les tubes sont unidirectionnels**
  - Une extrémité est accessible en ***lecture*** et l'autre l'est en ***écriture***.
  - Dans le cas des tubes anonymes, si l'une ou l'autre extrémité devient inaccessible, cela est irréversible.

Tubes anonymes et nommés

- **Mode FIFO**
  - Première information écrite sera la première à être consommée en lecture.
- **Communication d'un flot continu de caractères (stream)**
  - Possibilité de réaliser des opérations de lecture dans un tube sans relation avec les opérations d'écriture.
- **Opération de lecture est destructive :**
  - Une information lue est extraite du tube.

Tubes anonymes et nommés

- **Capacité limitée**
  - Notion de tube plein (taille : PIPE_BUF).
  - Écriture éventuellement bloquante.
- **Possibilité de plusieurs lecteurs et écrivains**
  - Nombre de lecteurs :
    - L'absence de lecteur interdit toute écriture sur le tube.
    - Signal SIGPIPE.
  - Nombre d'écrivains :
    - L'absence d'écrivain détermine le comportement du *read:* lorsque le tube est vide, la notion de fin de fichier est considérée.

Les tubes anonymes

- **Primitive pipe**
- **Pas de nom**
  - Impossible pour un processus d'ouvrir un pipe anonyme en utilisant *open*.
- **Acquisition d'un tube:**
  - Création : primitive *pipe*.
  - Héritage : *fork*, *dup*
    - Communication entre père et fils.
    - Un processus qui a perdu un accès à un tube n'a plus aucun moyen d'acquérir de nouveau un tel accès.

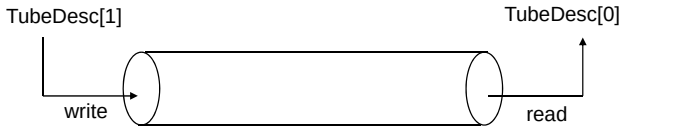
Les tubes anonymes

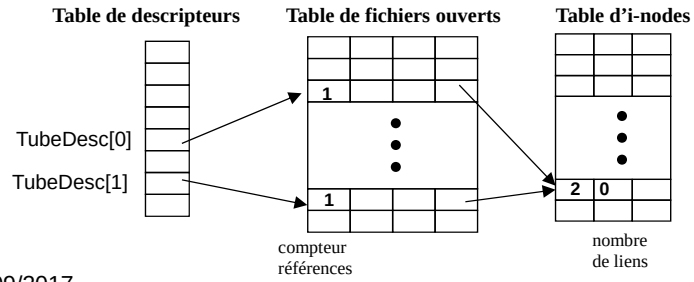
#include <unistd.h>

int pipe (int *TubeDesc);

- en cas de succès: appel renvoie 0
  - TubeDesc[0] : descripteur de lecture
  - TubeDesc[1] : descripteur d'écriture
- en cas d'échec : appel renvoie -1
  - errno = EMFILE (table de descripteurs de processus pleine).
  - errno = ENFILE (table de fichiers ouverts du système pleine).

Les tubes anonymes





Les tubes anonymes

- **Opérations autorisées**
  - ***read***, ***write*** : lecture et écriture
    - Opérations bloquantes par défaut
  - ***close***: fermer des descripteurs qui ne sont pas utilisés
  - ***dup***, ***dup2*** : duplication de description; redirection
  - ***fstat***, ***fcntl***: accès/modification des caractéristiques
- **Opérations non autorisées**
  - *open, stat, access, link, chmod, chown, rename*

Les tubes anonymes (fstat)

- **Accès aux caractéristiques d'un tube**

```
struct stat stat;

int main (int argc, char ** argv) {
  int tubeDesc[2];

 if (pipe (tubeDesc) == -1) {
    perror ("pipe"); exit (1);
}

 if ( fstat (tubeDesc[0],&stat) == -1) {
   perror ("fstat"); exit (2);
 }
```

```
if (S_ISFIFO (stat.st_mode)) {
   printf ("il s'agit d'un tube \n");
   printf ("num. inode %d \n", (int)stat.st_ino);
   printf ("nbr. de liens %d \n", (int)stat.st_nlink);
   printf ("Taille : %d \n", (int) stat.st_size);
}

return EXIT_SUCCESS;
}
```


Les tubes anonymes (lecture)

- **Lecture dans un tube d'au plus TAILLE_BUF caractères**
  - **read (tube[0], buff, TAILLE_BUF);**

- **si le tube n'est pas vide** et contient *taille* caractères, lire dans *buff* min (taille, TAILLE_BUF). Ces caractères sont extraits du tube.
- **si le tube est vide**
  - **si le nombre d'écrivains est nul**
    - fin de fichier; *read* renvoie 0
  - **sinon**
    - **si la lecture est bloquante** (par défaut), le processus est mis en sommeil jusqu'à ce que le tube ne soit plus vide ou qu'il n'y ait plus d'écrivains;
    - **sinon** retour immédiat; renvoie -1 et errno = EAGAIN.

Les tubes anonymes (écriture)

- **Ecriture de TAILLE_BUFF caractères dans un tube:**
  - **write (tube[1], buff, TAILLE_BUF);**
    - Ecriture sera **atomique** si **TAILLE_BUF** < **PIPE_BUF.**

- **si le nombre de lecteurs dans le tube est nul**
  - signal **SIGPIPE** est délivré à l'écrivain (terminaison du processus par défaut); si SIGPIPE capté, fonction *write* renvoie -1 et **errno = EPIPE**.
- **sinon**
  - **si l'écriture est bloquante**
    - le retour du *write* n'a lieu que lorsque **TAILLE_BUF** caractères ont été écrits**.**
  - **sinon**
    - **si (TAILLE_BUF <= PIPE_BUF)**
      s'il y a au moins TAILLE_BUFF emplacements libres dans le tube écriture atomique est réalisée;
      sinon renvoie -1, **errno = EAGAIN.**
    - **sinon** le retour est un nombre inférieur à **TAILLE_BUF**.

Les tubes anonymes (exemple fork)

- **Communication entre processus père et fils**

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#define S_BUF 100

int main (int argc, char ** argv) {
     int tubeDesc[2];
     char buffer[S_BUF];
     int n; pid_t pid_fils;

      if (pipe (tubeDesc) == -1) {
        perror ("pipe"); exit (1);
      }
if ( (pid_fils = fork ( )) == -1 ) {
       perror ("fork"); exit (2);
}
```

```
 if (pid_fils == 0) { /*fils */
    if (( n = read (tubeDesc[0],buffer, S_BUF)) == -1) {
       perror ("read"); exit (3);
  }
  else {
       buffer[n] = '\0'; printf ("%s\n", buffer);
  }
     exit (0);
}
else {/*père */
       if ( write (tubeDesc[1],"Bonjour", 7)== -1) {
            perror ("write"); exit (4);
       }
  wait (NULL);
 }
  return (EXIT_SUCCESS); }
```

Affichage fils:
Bonjour

Les tubes anonymes (exemple 2 fork)

- **Communication entre père et fils : blocage**

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#define S_BUF 100

int main (int argc, char ** argv) {
      int tubeDesc[2];
      char buffer[S_BUF];
      int n; pid_t pid_fils;

       if (pipe (tubeDesc) == -1) {
         perror ("pipe"); exit (1);
       }
if ( (pid_fils = fork ( )) == -1 ) {
        perror ("fork"); exit (2);
}
```

```
if (pid_fils == 0) { /* fils*/
  for (i=0; i<2; i++)
    if (( n= read (tubeDesc[0], buffer, S_BUF)) == -1){
              perror ("read"); exit (3);
     }
    else { buffer[n] = '\0'; printf ("%s\n",buffer); }
   exit (0);
  }
  else { /* père */
    for (i=0; i<2; i++)
      if ( write (tubeDesc[1],"Bonjour",7) == -1 {
              perror ("write"); exit (4);
       }
    wait (NULL);
  }
return (EXIT_SUCCESS); }
```

Affichage fils:
BonjourBonjour

• Processus père et fils bloqués

Les tubes anonymes (exemple)

- **Ecriture dans un tube sans lecteur** **test-sigpipe.c**

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

void sig_handler (int sig) {
  if (sig == SIGPIPE)
    printf ("eciture dans un tube sans lecteurs \n");
}

int main (int argc, char ** argv) {
   int tubeDesc[2]; struct sigaction action;
   action.sa_handler= sig_handler;
   sigaction (SIGPIPE, &action, NULL);
```

```
if (pipe (tubeDesc) == - 1) {
   perror ("pipe");
   exit (1);
}
 close (tubeDesc[0]); /* sans lecteur */

 if ( write (tubeDesc[1],"x", 1) == -1)
   perror ("write");

 return EXIT_SUCCESS;
}
```

```
> test-sigpipe
   eciture dans un pipe sans lecteurs
    write: Broken pipe
```


Les tubes anonymes

- ***read* et *write* sont des opérations bloquantes par défaut**
- **fonction *fcntl*:**
  - permet de les rendre non bloquantes

```
int tube[2], attributs;
….. pipe (tube); ….
/* rendre l'écriture non bloquante */
attributs = fcntl (tube[1], F_GETFL);
attributs |=O_NONBLOCK;
fcntl (tube[1], F_SETFL,attributs);
…..
```


Les tubes anonymes (fdopen)

- **Fonctions de haut niveau : *fdopen***
  - Obtenir un pointeur sur un objet du type *FILE*.

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

int main (int argc, char ** argv) {
int tube[2]; char car;

 FILE *ptLire;
  if (pipe (tube) == -1) {
   perror ("pipe"); exit (1);
  }

  if ((ptLire = fdopen (tube[0],"r")) == NULL) {
    perror ("fdopen"); exit (2);
  }
```

```
  if (write (tube[1],"x",1)!=1) {
     perror ("write"); exit (3);
  }

  if (fscanf (ptLire, "%c", &car) == -1) {
      perror ("fscanf"); exit (4);
  }
  else
    printf ("caractere lu: %c\n", car);
  return (EXIT_SUCCESS);
  }
```

```
> test-fdopen
    caractere lu : x
```

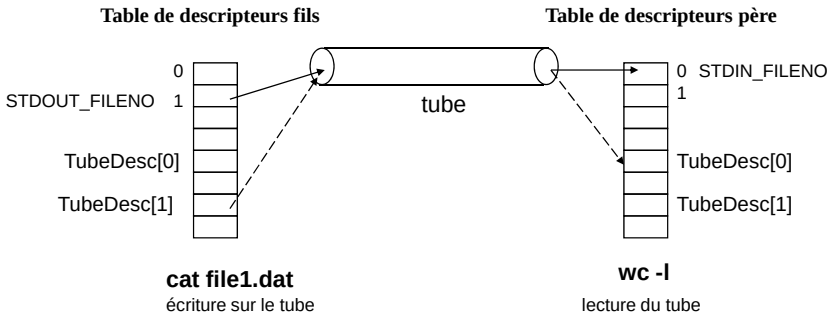
Les tubes anonymes (dup et close)

- ***close***
  - Fermeture des descripteurs qui ne sont pas utilisés.
- ***dup, dup2***
  - Duplication des descripteurs.
  - Rediriger les entrées-sorties standard d'un processus sur un tube.

```
int tube[2], attributs;
…..
  pipe (tube); ….
  dup2(tube[0], STDIN_FILENO);
  close (tube[0]);
…
```


Les tubes anonymes (exemple de redirection)

- **/* nombre de lignes d'un fichier */**
  - cat file1.dat | wc -l



Les tubes anonymes (exemple de redirection)

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

int main (int argc, char ** argv) {
int tubeDesc[2]; pid_t pid_fils;

 if (pipe (tubeDesc) == -1) {
   perror ("pipe");
   exit (1);
}

 if ( (pid_fils = fork ( )) == -1 ){
   perror ("fork");
   exit (2);
}
```

```
 if (pid_fils == 0) { /* fils */
     dup2(tubeDesc[1],STDOUT_FILENO);
     close (tubeDesc[1]); close (tubeDesc[0]);
     if (execl ("/bin/cat", "cat", "file1.dat",NULL) == -1) {
         perror ("execl"); exit (3);
     }
  }
  else { /* père */
          dup2(tubeDesc[0],STDIN_FILENO);
          close (tubeDesc[0]);
          close (tubeDesc[1]);
         if (execl ("/bin/wc", "wc", "-l", NULL) == -1) {
            perror ("execl"); exit (3);
        }
   }
   return (EXIT_SUCCESS);
}
```


Tubes nommés

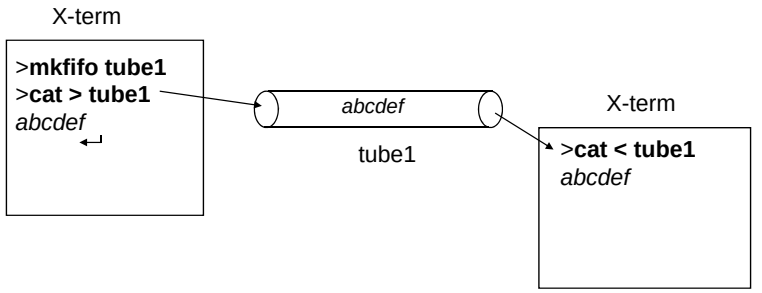
- **Permettent à des processus sans lien de parenté de communiquer en mode flot (stream).**
  - Toutes les caractéristiques des tubes anonymes.
  - Sont référencés dans le système de gestion de fichiers.
  - Utilisation de la fonction *open* pour obtenir un descripteur en lecture ou écriture.
  - ls -l

```
prw-rw-r--   1 arantes  src          0 Nov  9  2004   tube1
```

Tubes nommés (mkfifo)

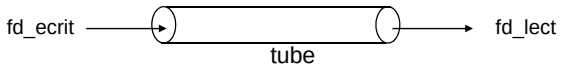
- **Création d'un tube nommé**
  - **mkfifo [-p] [-m mode]** ***référence***
    - -m mode : droits d'accès (les mêmes qu'avec chmod).
    - -p : création automatique de tous les répertoires intermédiaires dans le chemin *référence*.
- **Fonction**
  - **int mkfifo (const char *ref, mode_t droits);**
    - *ref* définit le chemin d'accès au tube nommé et *droits* spécifie les droits d'accès.
    - Renvoie 0 en cas de succès; -1 en cas d'erreur.
      - errno = EEXIST, si fichier déjà créé.

Tubes nommés (mkfifo)



Tubes nommés (open)

- **Par défaut bloquante (rendez-vous):**
  - Une demande d'ouverture en lecture est bloquante s'il n'y a aucun écrivain sur le tube.
  - Une demande d'ouverture en écriture est bloquante s'il n'y a aucun lecteur sur le tube.




```
int fd_lect, fd_ecrit;
fd_lect = open ("tube", O_RDONLY);
fd_ecrit = open ("tube", O_WRONLY);
```


Tubes nommés (open)

- **Ouverture non bloquante**
  - Option O_NONBLOCK lors de l'appel à la fonction *open*
    - **Ouverture en lecture** :
      - Réussit même s'il n'y a aucun écrivain dans le tube.
      - Opérations de lectures qui se suivent sont non bloquantes.
    - **Ouverture en écriture** :
      - Sur un tube sans lecteur, l'ouverture échoue: valeur -1 renvoyée.
      - Si le tube possède des lecteurs, l'ouverture réussit et les écritures dans les tubes sont non bloquantes.

Tube nommé (suppression du nœud)

- **Un nœud est supprimé quand:**
  - Le nombre de liens physiques est nul.
    - Fonction *unlik* ou commande *rm.*
  - Le nombre de liens internes est nul.
    - Nombres de lecteurs et écrivains sont nuls.
- **Si nombre de liens physiques est nul, mais le nombre de lecteurs et/ou écrivains est non nul**
  - Tube nommé devient un tube anonyme.

Tubes nommés (écrivain)

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>

#define S_BUF 100
int n ;
char buffer[S_BUF];
int main (int argc, char ** argv) {
  int fd_write;

   if ( mkfifo(argv[1],
      S_IRUSR|S_IWUSR) == -1) {
        perror ("mkfifo");
        exit (1);
   }
```

```
if (( fd_write = open (argv[1],
     O_WRONLY)) == -1) {
       perror ("open");
       exit (2);
 }

 if (( n= write(fd_write,"Bonjour", 7)) == -1) {
    perror ("write");
    exit (3);
 }

 close (fd_write);
 return EXIT_SUCCESS;

}
```


Tubes nommés (lecteur)

```
#define _POSIX_SOURCE 1
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#define S_BUF 100
int n ; char buffer[S_BUF];

int main (int argc, char ** argv) {
int fd_read;
    if (( fd_read = open (argv[1],
        O_RDONLY)) == -1) {
        perror ("open"); exit (1)
}
```

```
if (( n= read (fd_read, buffer, S_BUF)) == -1){
            perror ("read");
            exit (2);
 }
 else {
        buffer[n] = '\0';
        printf ("%s\n",buffer);
 }
 close (fd_read);
 return EXIT_SUCCESS;
}
```


Tubes nommés: interblocage (ouverture bloquante)

PROCESSUS 1 :

```
int main (int argc, char ** argv) {
  int fd_write, fd_read;

 if ( (mkfifo("tube1",S_IRUSR|S_IWUSR) == -1) ||
 (mkfifo("tube2",S_IRUSR|S_IWUSR) == -1)) {
       perror ("mkfifo"); exit (1);
 }

 if (( fd_write = open ("tube1", O_WRONLY)) == -1) {
       perror ("open"); exit (2);
 }

 if (( fd_read = open ("tube2", O_RDONLY)) == -1) {
       perror ("open"); exit (3);
 }
…..
 return EXIT_SUCCESS;
 }
```

PROCESSUS 2 :

```
int main (int argc, char ** argv) {
   int fd_write, fd_read;

   if (( fd_write = open ("tube2", O_WRONLY))
       == -1) {
            perror ("open"); exit (2);
   }

   if (( fd_read = open ("tube1", O_RDONLY))
      == -1) {
            perror ("open"); exit (3);
   }
   …..
   return EXIT_SUCCESS;
}
```