Programmation Système

Communication par Tubes

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Licence Sciences et Technologies

Mention : Informatique

2ème Année

Mohamed TAGHELIT
taghelit@univ-tours.fr

Communication par Tubes

- Caractéristiques générales et communes
- Les tubes ordinaires
- Algorithmes de lecture/écriture dans un tube
- Les tubes nommés
- Principes d'ouverture d'un tube nommé

Caractéristiques Générales des Tubes

- Un tube est un mécanisme de communication entre processus, appartenant au système de fichiers
  - association d'un nœud du système de fichiers (type : **`S_IFIFO`**)
  - association de descripteurs (application des appels **`read()`**, **`write()`**, …)
- Un tube est un canal unidirectionnel (une entrée, une sortie)
  - association de deux entrées dans la table des descripteurs du processus
  - association de deux entrées dans la table des fichiers ouverts
- La lecture depuis un tube est destructrice
- La communication via un tube correspond à un flot continu de caractères
- Les entrées/sorties via un tube sont gérées en mode **`FIFO`**
- Un tube a une capacité finie (notion de tube plein)
- À un tube est associée la notion de "nombres de lecteurs/écrivains" qui influe sur les lectures/écritures (et l'ouverture des tubes nommés)

Tubes Ordinaires

- Un tube ordinaire a un compteur de liens nul (aucune référence à ce nœud)
- Un tube ordinaire est supprimé lorsque aucun processus ne l'utilise
- Impossibilité d'ouvrir un tube ordinaire, avec `open()` par exemple
- La connaissance de l'existence d'un tube ordinaire se traduit par la possession d'au moins un de ses descripteurs suite :
  - à une création du tube ordinaire, ou
  - à l'héritage d'un ou plus de ses descripteurs
- Un tube ordinaire permet une communication uniquement entre processus ayant un ancêtre commun
- La perte d'accès à un tube ordinaire est irréversible

Création d'un Tube Ordinaire

- Un processus peut créer à tout moment un tube ordinaire

tableau qui stocke les deux descripteurs du tube créé

```
#include <unistd.h>
      int pipe(int pipefd[2]);
```

- Alloue un nœud, deux entrées dans la table des fichiers ouverts et deux descripteurs dans la table du processus appelant

- Valeur de retour :
  - 0 en cas de succès,
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

- Un tube ordinaire peut être manipulé par la majorité des primitives systèmes applicables à un nœud du système de fichiers :
  - **read()**, **write()**,
  - **fstat()**, **fcntl()**, **close()**, …

- Du fait de la gestion **FIFO** des lectures/écritures via un tube ordinaire, l'utilisation de la primitive **lseek()** est interdite.

Héritage des Descripteurs d'un Tube

Programme `tube_heritage.c`

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

int main(int argc) {

int fd1[2], fd2[2];
char buf[128];

    if (pipe(fd1) == -1)
        perror("Erreur pipe 1");
    if (pipe(fd2) == -1)
        perror("Erreur pipe 2");

    if (fork() != 0) {
        write(fd1[1], "Bonjour du père !\0", strlen("Bonjour du père !") + 1);
        read(fd2[0], buf, sizeof(buf));
        printf("Message reçu du fils : %s\n", buf);
        exit(0);
    } else {
        read(fd1[0], buf, sizeof(buf));
        printf("\tMessage reçu du père : %s\n", buf);
        write(fd2[1], "Bonjour du fils !\0", strlen("Bonjour du fils !") + 1);
        exit(0);
    }
}
```

création de deux tubes par le père

exécuté par le père

exécuté par le fils

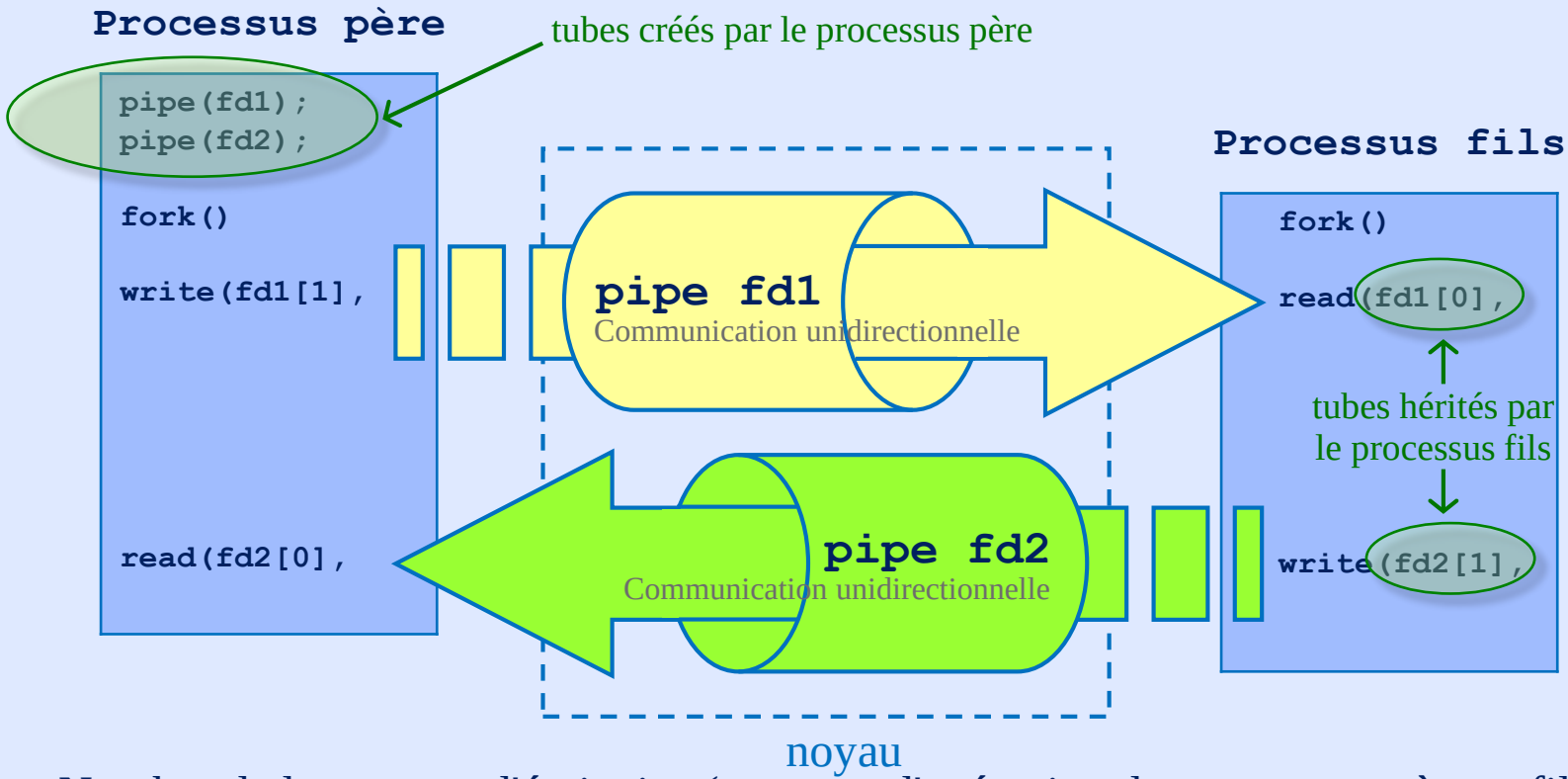
Exécution de `tube_heritage.c`

```
[student]$ ./tube_heritage
        Message reçu du père : Bonjour du père !
Message reçu du fils : Bonjour du fils !
[student]$
```

affiché par le processus fils

affiché par le processus père

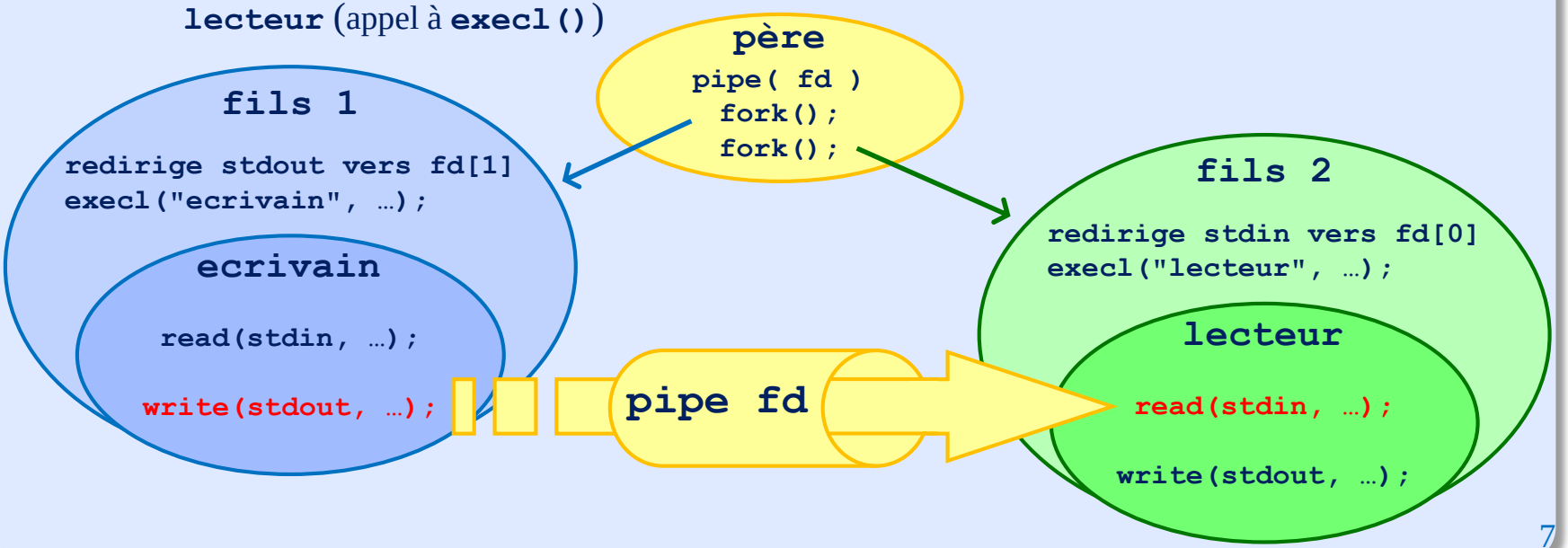
Héritage et Nombre de Lecteurs/Écrivains



- Nombre de lecteurs et d'écrivains (en cours d'exécution des processus père et fils) ?
  - tube **fd1** : descripteur **fd1[0]** ouvert chez le père et chez le fils = 2 lecteurs
    descripteur **fd1[1]** ouvert chez le père et chez le fils = 2 écrivains
  - tube **fd2** : descripteur **fd2[0]** ouvert chez le père et chez le fils = 2 lecteurs
    descripteur **fd2[1]** ouvert chez le père et chez le fils = 2 écrivains
- Il est recommandé de toujours fermer les descripteurs de tube non utilisés (impact sur le nombre de lecteurs/écrivains)

Exemple d'Utilisation d'un Tube Ordinaire

- Problème
  - Données : **ecrivain** et **lecteur** sont deux exécutables qui chacun lit depuis l'entrée standard des données qu'il écrit sur la sortie standard
  - But : faire communiquer les deux exécutables de telle sorte que **lecteur** puisse lire les données écrites par **ecrivain**
  - Indices : écrire un programme qui :
    - crée un tube ordinaire
    - crée un processus fils qui redirige sa sortie standard vers l'entrée du tube et lance **ecrivain** (appel à **execl()**)
    - crée un processus fils qui redirige son entrée standard vers la sortie du tube et lance **lecteur** (appel à **execl()**)



Exemple d'Utilisation d'un Tube Ordinaire

- Programme **ecrivain_lecteur.c**

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

```
void main() {

int fd[2];
int retour;

          if ( pipe(fd) == -1 )
                    perror("Erreur création tube");

    if (fork() == 0) {
        close(STDOUT_FILENO);
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]); close(fd[1]);
        execl("ecrivain", "ecrivain", NULL);
        exit(-1);
    }

    if (fork() == 0) {
        close(STDIN_FILENO);
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]); close(fd[1]);
        execl("lecteur", "lecteur", NULL);
        exit(-11);
    }
    close(fd[0]); close(fd[1]);
    wait(&retour);
    wait(&retour);
}
```

tableau de descripteurs du tube

création du tube

création du 1er fils

fermeture de la sortie standard

redirection de la sortie standard vers l'entrée du tube

lance l'exécutable **ecrivain**

exécuté par le 1er fils

création du 2ème fils

exécuté par le 2ème fils

Exemple d'Utilisation d'un Tube Ordinaire

Programme `ecrivain.c` (identique à `lecteur.c`)

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

void main() {

char buf[128];
int nl;

    while(1){
        nl = read(STDIN_FILENO, buf, sizeof(buf));
        buf[nl] = '\0';
        write(STDOUT_FILENO, buf, strlen(buf));
    }
}
```

← lecture depuis l'entrée standard

← écriture vers la sortie standard

Exécution de `ecrivain_lecteur.c`

```
[student]$ gcc ecrivain.c -o ecrivain
[student]$ gcc ecrivain.c -o lecteur
[student]$ gcc ecrivain_lecteur.c -o ecrivain_lecteur
[student]$ ./ecrivain_lecteur
Bonjour !
Bonjour !
Bonsoir !
Bonsoir !
Bye bye !
Bye bye !
^C
[student]$
```

← texte introduit par l'utilisateur, lu par ecrivain qui l'a ensuite écrit dans l'entrée du tube

← texte affiché par lecteur après l'avoir lu de la sortie du tube

Algorithme de Lecture dans un Tube

- Cas d'un processus qui tente de lire **N** octets à partir de la sortie d'un tube

1. Le tube n'est pas vide et contient **n** octets
   a. min(**n**, **N**) octets sont lus,
   b. la primitive renvoie le nombre réel d'octets lus.
2. Le tube est vide
   a. Le nombre d'écrivains est nul (la fin de fichier est atteinte)
      - aucun octet n'est lu,
      - la primitive renvoie 0.

   b. Le nombre d'écrivains 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 primitive renvoie -1 et **`errno = EAGAIN.`**

Algorithme d'Écriture dans un Tube

- Cas d'un processus qui tente d'écrire **N** octets à partir de l'entrée d'un tube

1. Le nombre de lecteurs est nul
   a. le signal **SIGPIPE** est délivré au processus,
2. Le nombre de lecteurs est non nul
   a. si l'écriture est bloquante
      - retour de la primitive une fois les **N** octets écrits,
      - le processus peut, éventuellement, être mis en sommeil dans l'attente que le tube se vide .

   b. si l'écriture est non bloquante
      - si **N > PIPE_BUF**, la primitive retourne un nombre inférieur à **N**,
      - si **N ≤ PIPE_BUF** et s'il y a au moins **N** octets libres dans le tube, une écriture atomique est réalisée,
      - si **N ≤ PIPE_BUF** et s'il y a moins de **N** octets libres dans le tube, aucune écriture n'est réalisée et la primitive retourne -1.

Les Tubes Nommés

- Un tube nommé est référencé dans le système de fichiers
- Un tube nommé est supprimé lorsqu'il ne lui est associé aucun lien physique et aucun lien interne
- Un tube nommé nécessite une étape d'ouverture avant qu'il puisse être accessible (en lecture/écriture). Étape bloquante par défaut.
  - Synchronisation des ouvertures entre processus
- Un tube nommé permet une communication entre processus même sans lien de parenté (n'ayant pas un ancêtre commun)

Création d'un Tube Nommé

- Un processus peut créer à tout moment un tube ordinaire

```
#include <sys/types.h>
#include <sys/stat.h>
      int mkfifo(const *pathname, mode_t mode);
```

permissions d'accès demandées (modifiées par umask)

nom du tube

- Crée un fichier spécial **FIFO** (tube nommé) à l'emplacement **pathname**.
- Valeur de retour :
  - 0 en cas de succès,
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).
- Un tube nommé peut être manipulé par la majorité des primitives systèmes applicables à un nœud du système de fichiers :
  - **open()**, **close()**,
  - **read()**, **write()**,
  - **fstat()**, **fcntl()**, **unlink()**, …
- Du fait de la gestion **FIFO** des lectures/écritures via un tube nommé, l'utilisation de la primitive **lseek()** est interdite.

Ouverture d'un Tube Nommé

- Cas d'une demande d'ouverture d'un tube nommé par un processus ayant les droits correspondants.

1. Si l'ouverture est bloquante → synchronisation (prise de rendez-vous)
   a. Une demande d'ouverture en lecture est bloquante s'il n'y a aucun écrivain.
   b. Une demande d'ouverture en écriture est bloquante s'il n'y a aucun lecteur.

2. Si l'ouverture est non bloquante
   a. Une demande d'ouverture en lecture réussit toujours.
      Les opérations de lecture ultérieures sont non bloquantes jusqu'à demande explicite du contraire.
   b. Une demande d'ouverture en écriture échoue s'il n'y a aucun lecteur.
   c. Une demande d'ouverture en écriture réussit s'il y a au moins un lecteur.
      Les opérations d'écriture ultérieures sont non bloquantes jusqu'à demande explicite du contraire.

Exemple d'Utilisation d'un Tube Nommé

Programme `ecrivain_nomme.c`

```
void main() {
int fd;

mkfifo("tube_echange.txt", 0666);

fd = open("tube_echange.txt", O_WRONLY);
write(fd, "Bonjour !", sizeof("Bonjour !"));
close(fd);
printf("Fin Ecrivain.");
}
```

Programme `lecteur_nomme.c`

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

void main() {
int fd;
char car;

fd = open("tube_echange.txt", O_RDONLY);
while ( read(fd, &car, 1) != 0 )
    printf("Caractère lu : %c\n", car);
close(fd);
printf("Fin Lecteur.");
}
```

Exécution de `ecrivain_nomme.c`

```
[student]$ ./ecrivain_nomme
```

bloqué sur **open** tant que **lecteur** n'a pas fait **open** à son tour

```
Fin Ecrivain.
[student]$
```

Exécution de `lecteur_nomme.c`

```
[student]$
[student]$
[student]$ ./lecteur_nomme
Caractère lu : B
Caractère lu : o
Caractère lu : n
Caractère lu : j
Caractère lu : o
Caractère lu : u
Caractère lu : r
Caractère lu :
Caractère lu : !
Caractère lu :
Fin Lecteur.
[student]$
```