جامعة ابن طفيل Université Ibn Tofail
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵅⵓ ⵟⵓⴼⴰⵢⵍ Ibn Tofail University

Système d'Exploitation
Tuyaux (Tubes)

Med. AMNAI
Filière SMI-S4
Département d'Informatique

2023-2024

Plan

Plan

Plan

Plan

Plan

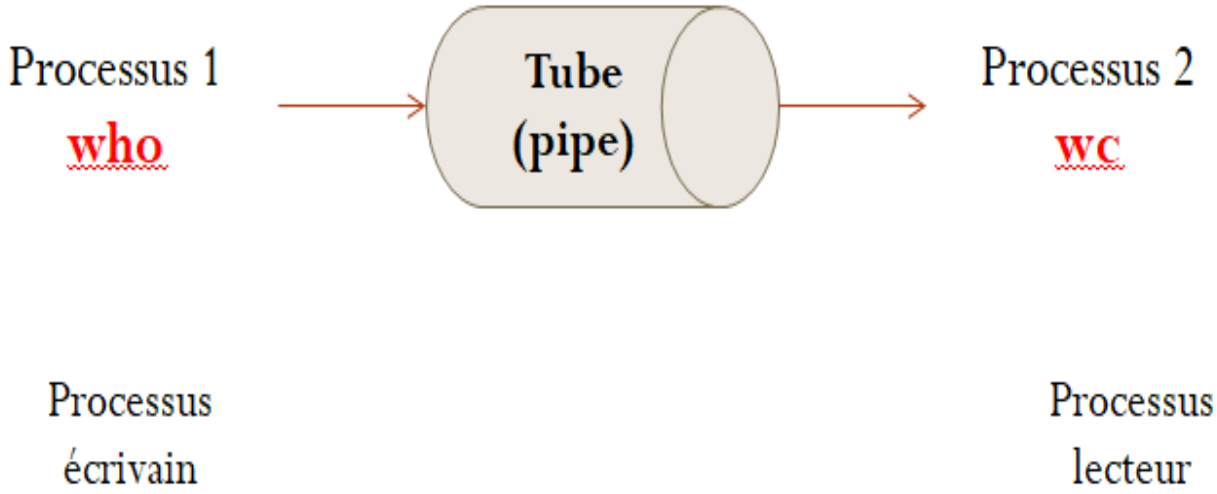
Principe

Un processus peut envoyer des **données** à un autre processus se trouvant sur la même machine ou sur des machines différentes via :

- Tubes de communication **anonymes** ;
- Tubes de communication **nommés** ;
- **Sockets**.

Principe (suite)

- **Exemple** : Déterminer le nombre d'utilisateurs connectés au système en appelant **who**, puis en comptant les lignes avec **wc**.



- **Principe** : Sortie standard redirigée vers entrée d'un tube :

  commande 1 | commande 2 | ... | commande n

- **Ex : ps -aux | wc -l**

Généralités

- **Tubes** (ou **"pipe"**) permettent à un groupe de processus d'envoyer des **données** à un autre groupe de processus ;
- Données envoyées directement **en mémoire flot continu d'octets** et ne sont pas stockées temporairement sur le disque dur (**rapidité**) ;
- Un tube appartient aux mécanismes lies au SGF.
- Un tube est désigné par un **descripteur**(s) ;
- Un tube peut donc être manipulé par les primitives classiques : **read**, **write**, ...

Généralités (1/2)

- Un tube (tuyau ou pipe) de données a deux côtés :
  - Un côté permettant d'**écrire** les données dedans ;
  - Un côté permettant de **lire** les données ;
  - Chaque côté est un **descripteur** de fichier ouvert soit en **lecture** ou en **écriture** ;
- L'opération de lecture est **destructive** !
- L'**ordre** des caractères en entrée est **conservé** en sortie (**FIFO**).
- Un tube a une **capacité** finie de **5 à 80 Ko**.

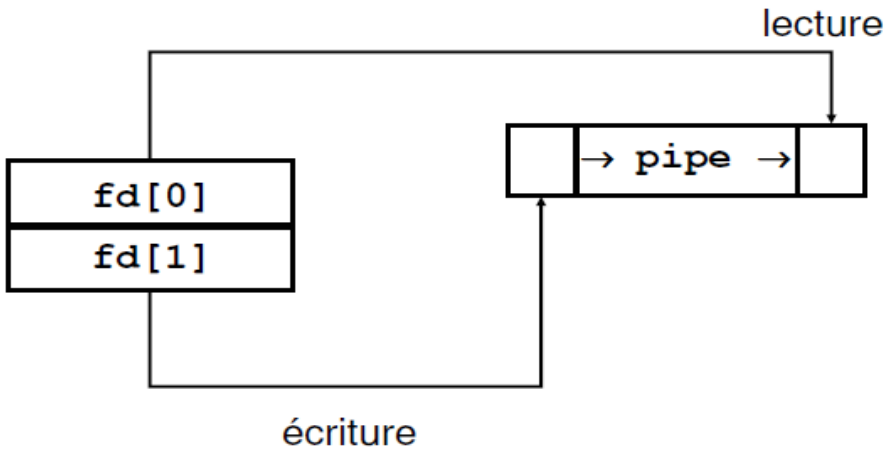
Généralités (2/2)

- Si un tube est **vide**, le processus essayant de lire le tuyau sera **suspendu** jusqu'à ce que des données soient disponibles ;
- Un processus essayant **d'écrire dans un tuyau plein sera suspendu** en attendant qu'un espace suffisant se libère ;
- Si plusieurs processus **lisent le même tuyau**, **toute donnée lue par l'un disparaît** pour les autres ;
- Si l'on veut **envoyer** des informations identiques à **plusieurs processus**, il est nécessaire de **créer un tuyau vers chacun** d'eux.

Contraintes d'Utilisation

- Les tuyaux ne permettent qu'une communication **unidirectionnelle** ;
- Les processus pouvant **communiquer** au moyen d'un tuyau doivent être issus d'un **ancêtre commun** qui devra avoir créé le tuyau ;
- Un processus ne peut **utiliser** que les tubes qu'il a **créés** lui-même (pipe) ou qu'il a **hérités** de son père via (**fork** ou **exec**).

Création de Tubes



Création de Tubes (suite)

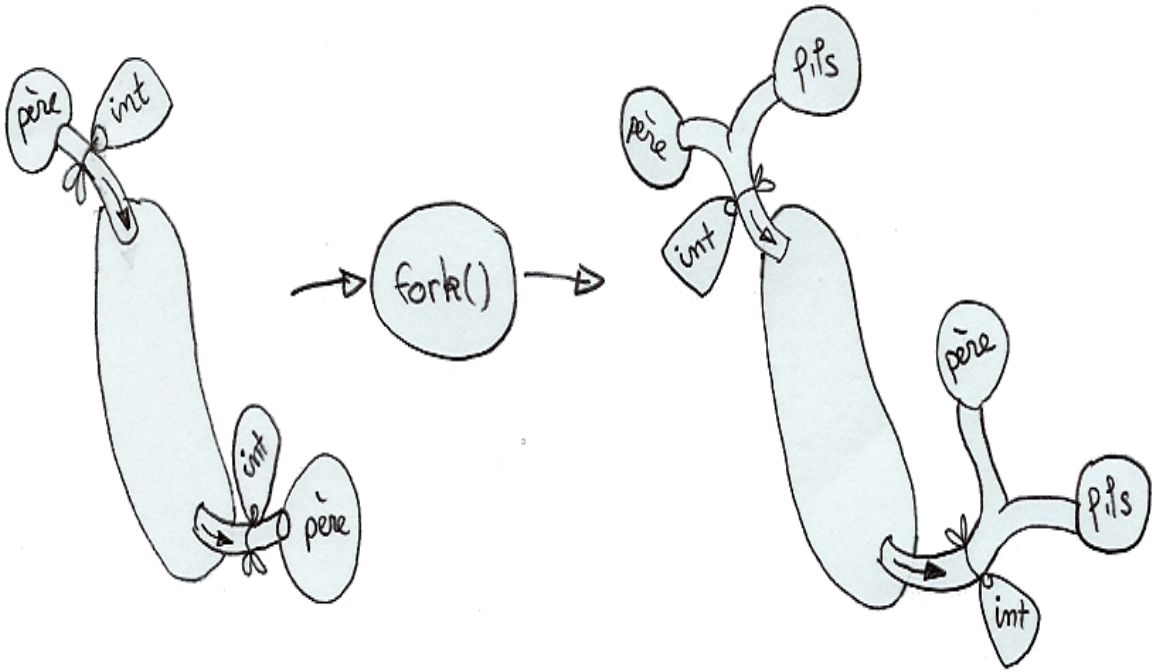
- **#include <unistd.h> int pipe(int fd[2]) ;**
- Appel à la primitive (**pipe**) avec **int fd[2]** en paramètre ;
  - **fd[0]** descripteur en **lecture** ;
  - **fd[1]** descripteur en **écriture**.
- Valeur de retour **0** (ok) ou **-1** (plantage).

Etapes de Communication de Données

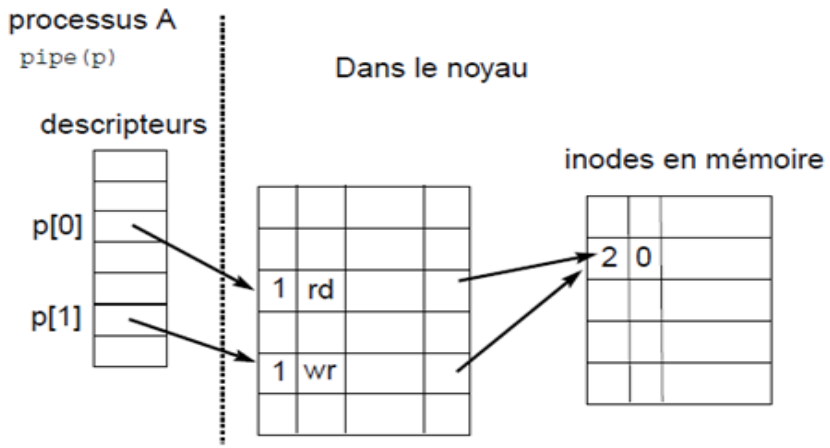
Exemple : Un processus qui crée un fils auquel il va envoyer des données :

1. Le processus père crée le tuyau au moyen de **pipe()**.
2. Le père crée un processus fils grâce à **fork()**. ==> les deux processus **partagent** le tuyau.
3. Puisque le **père** va écrire dans le tuyau, il n'a pas besoin du **côté lecture**, donc il **le ferme**.
4. De même, le **fils ferme** le **côté écriture**.
5. Le processus père peut envoyer des données au fils à travers le tuyau.

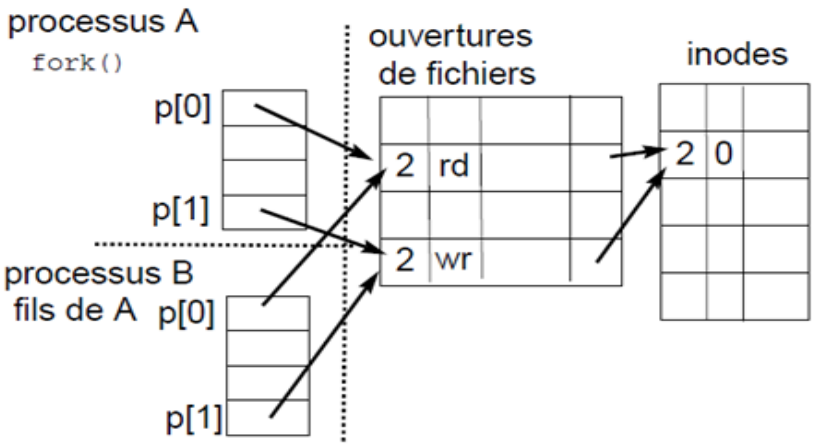
Etapes de Communication de Données



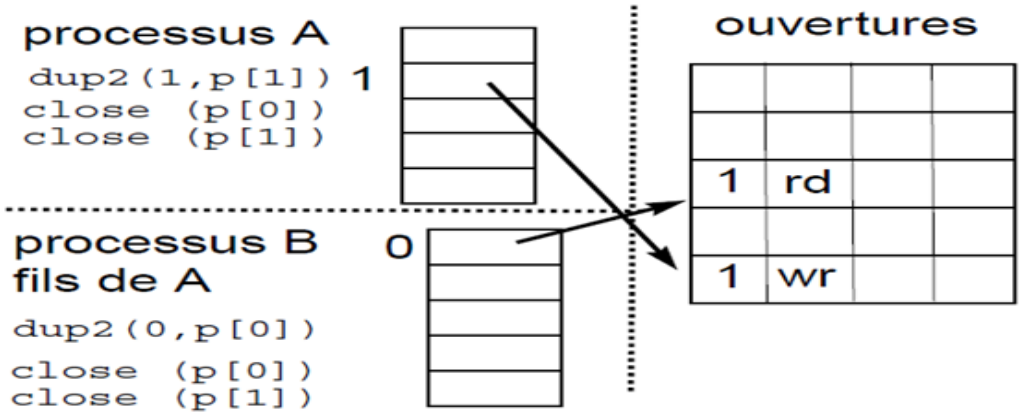
Etapes de Communication de Données



1



2



3

Fermeture d'un Tube

Considéré fermé lorsque :

- **Tous** les descripteurs en **lecture** sont **fermés** ;
- **Tous** les descripteurs en **écriture** sont **fermés** ;

RQ : Primitive int **close(int fd) ;** permet de fermer un tube.

Lecture et Ecriture d'un tube anonyme

- Primitive Lecture : **int read(int desc[0], char *buf, int nb)**
  - Lecture de **nb** caractères depuis le tube **desc** dans le tampon **buf** ;
  - **Retourne** en résultat le **nombre** de **caractères** réellement **lus**.
- Primitive Ecriture : **int write(int desc[1], char *buf, int nb) ;**
  - Écriture de **nb** caractères placés dans le tampon **buf** dans le tube **desc** ;
  - **Retourne** en résultat le **nombre** de **caractères** réellement **écrits**.

Appel système pipe()

```
#include <unistd.h>

int tuyau[2], retour;
retour = pipe(tuyau);
if ( retour == -1 ) {

    /* erreur : le tuyau n'a pas pu etre cree */

}
```

Exemple (pipe1.c)

```c
#define LECTURE 0
#define ECRITURE 1
int main(int argc, char *argv[]) {
int tuyau[2], nb, i;
char donnees[10];

if (pipe(tuyau) == -1) { /* creation du pipe */
     perror("Erreur dans pipe()");
     exit(EXIT_FAILURE);
}

switch (fork()){ /* les deux processus partagent le pipe */

case -1 : /* erreur */
     perror("Erreur dans fork()");
     exit(EXIT_FAILURE);

case 0 : /* processus fils, lecteur */
     close(tuyau[ECRITURE]); /* on ferme le cote ecriture */

     /* on peut alors lire dans le pipe */
     nb = read(tuyau[LECTURE], donnees, sizeof(donnees));

     for (i = 0; i < nb; i++) {
          putchar(donnees[i]);
     }

     putchar('\n');
     close(tuyau[LECTURE]);
     exit(EXIT_SUCCESS);
     default : /* processus pere, ecrivain */
     close(tuyau[LECTURE]); /* on ferme le cote lecture */
     strncpy(donnees, "bonjour", sizeof(donnees));

     /* on peut ecrire dans le pipe */
     write(tuyau[ECRITURE], donnees, strlen(donnees));
     close(tuyau[ECRITURE]);
     exit(EXIT_SUCCESS);
}
}
```

Fonctions E/S standard

- Afin de pouvoir utiliser les fonctions d'entrées/sorties standard (**fprintf(), fscanf()...**) au lieu de **read**() et **write**(),
- Il faut transformer les descripteurs de fichiers en pointeurs de type **FILE ***.

```
FILE * mon_tuyau = fdopen(tuyau[LECTURE], "r");
```

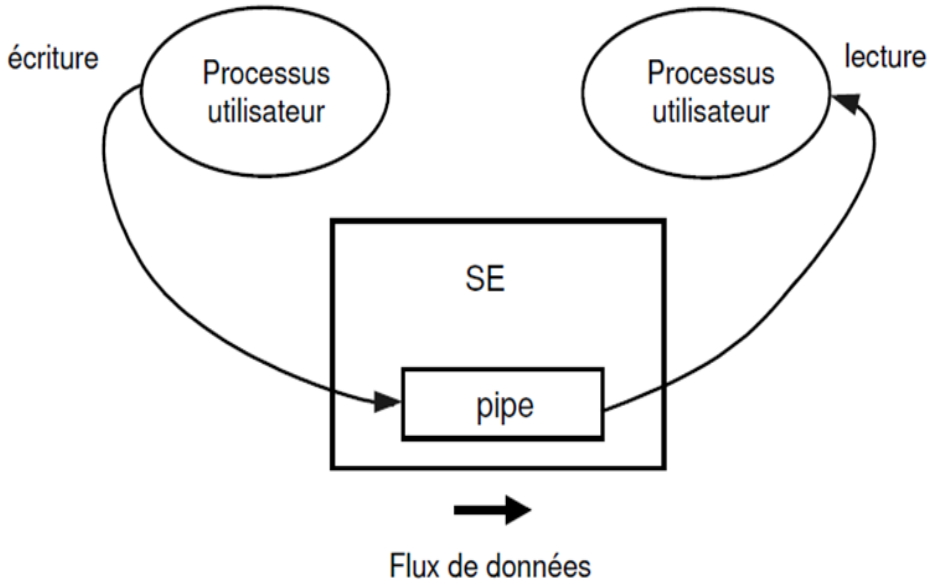
Exemple (pipe2.c)

```c
#define LECTURE 0
#define ECRITURE 1

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

int tuyau[2];
char str[100];
FILE *mon_tuyau ;

if ( pipe(tuyau) == -1 ) {
    perror("Erreur dans pipe()");
    exit(EXIT_FAILURE);
}

switch (fork()) {

case -1 : /* erreur */
        perror("Erreur dans fork()");
        exit(EXIT_FAILURE);
case 0 : /* processus fils, lecteur */
        close(tuyau[ECRITURE]);

        /* ouvre un descripteur de flot FILE * a par
        /* du descripteur de fichier UNIX */

        mon_tuyau = fdopen(tuyau[LECTURE], "r");
        if (mon_tuyau == NULL) {
            perror("Erreur dans fdopen()");
            exit(EXIT_FAILURE);
        }

        /* mon_tuyau est un FILE * accessible en lecture */
        fgets(str, sizeof(str), mon_tuyau);
        printf("Mon pere a ecrit : %s\n", str);

        /* il faut faire fclose(mon_tuyau) ou a la rigueur */
        /* close(tuyau[LECTURE]) mais surtout pas les deux */
        fclose(mon_tuyau);
        exit(EXIT_SUCCESS);
default : /* processus pere, ecrivain */
close(tuyau[LECTURE]);
mon_tuyau = fdopen(tuyau[ECRITURE], "w");

if (mon_tuyau == NULL) {
    perror("Erreur dans fdopen()");
    exit(EXIT_FAILURE);

}
/* mon_tuyau est un FILE * accessible en ecriture */
fprintf(mon_tuyau, "petit message\n");
fclose(mon_tuyau);
exit(EXIT_SUCCESS);

}
}
```

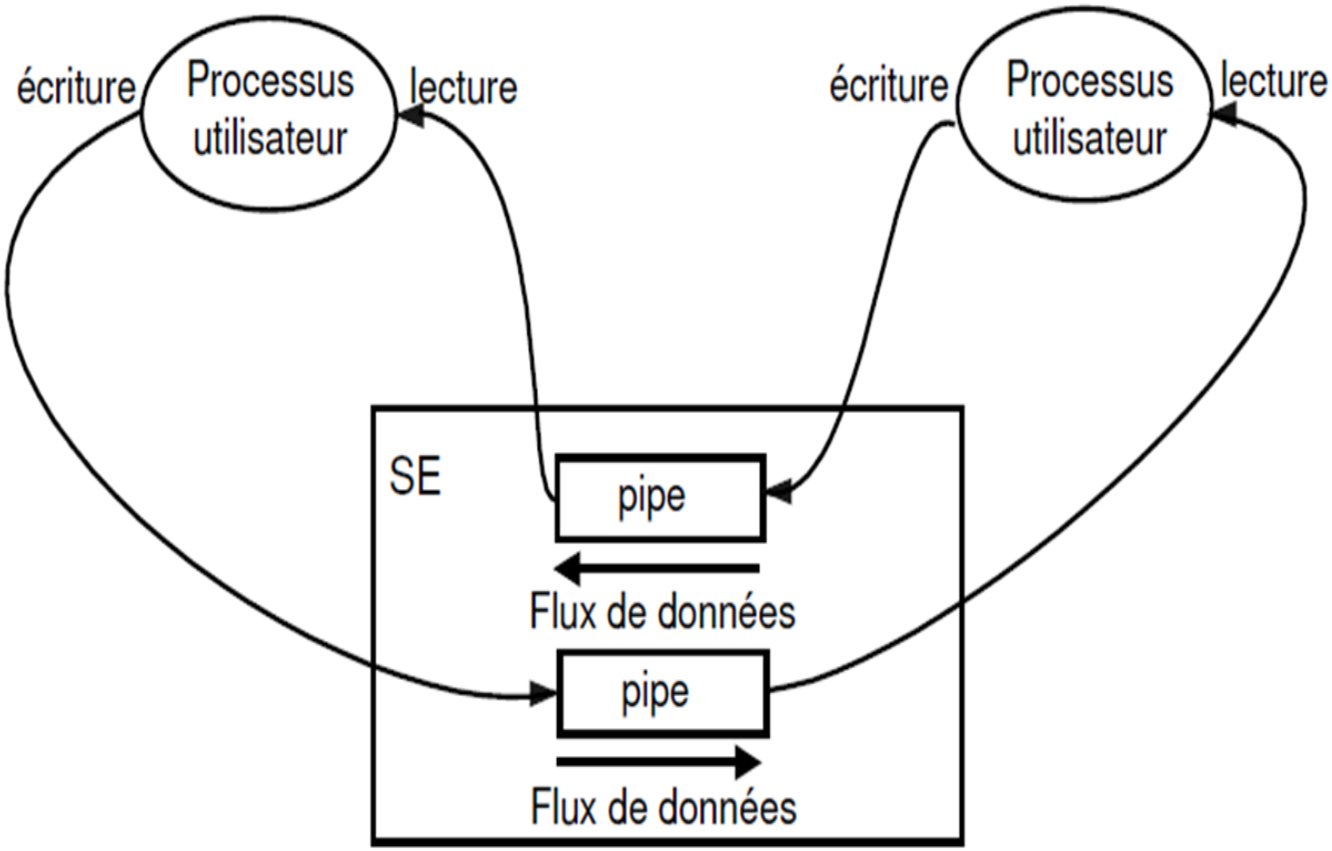
Exemple (pipe3.c)

```c
#include<sys/types.h> // pipe3.c
#include<unistd.h>
#include<stdio.h>
#define R 0
#define W 1
int main ( )
{
int fd[ 2 ] ;
char message [100 ] ;    // pour recuperer un message
int nboctets ;
char * phrase = " message envoye au pere par le fils " ;
pipe ( fd ) ;          // creation d ' un tube sans nom

if ( fork ( ) == 0 )     // creation d' un processus fils
{
    // Le fils ferme le descripteur non utilise de lecture
    close ( fd[R ] ) ;

    // depot dans le tube du message
    write( fd[W] , phrase , strlen ( phrase ) + 1 ) ;

    // fermeture descripteur d' ecriture
    close( fd[W] ) ;
}else{
    // Le pere ferme le descripteur non utilise d' ecriture
    close ( fd[W] ) ;

    // extraction du message du tube
    nboctets = read ( fd[R ] , message , 100 ) ;
    printf( " Lecture %d octets : % s\n " , nboctets , message ) ;

    // fermeture du descripteur de lecture
    close ( fd[R ] ) ;
}
```

Communication Unidirectionnelle



Communication Bidirectionnelle



Redirection avec dup2()

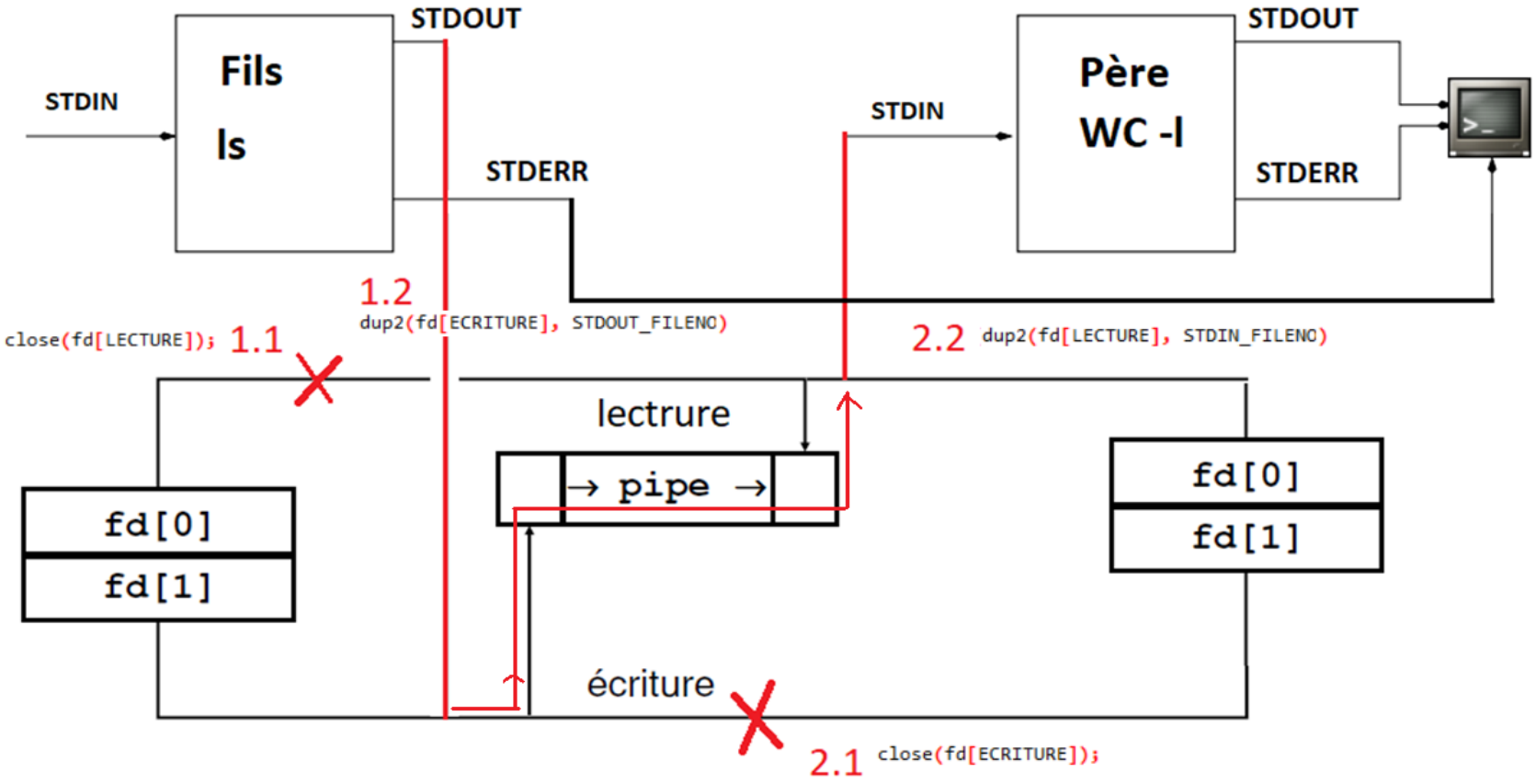
dup, dup2 : Dupliquer un descripteur de fichier.

- **int dup2 (int descripteur, int copie) ;**
  - **descripteur** : descripteur de fichier à dupliquer ;
  - **copie** : numéro du descripteur souhaité pour la copie. Le descripteur copie est éventuellement fermé avant d'être réalloué.
- **int descripteur2 = dup(int descripteur1) ;**
  - utilise le plus petit numéro inutilisé pour le nouveau descripteur.

Remarques :

- **dup**() et **dup2**() renvoient le nouveau **descripteur**, ou **-1** s'ils échouent, auquel cas **errno** contient le code d'erreur.
- Il est recommandé d'**utiliser** plutôt **dup2**() que **dup**() pour des raisons de **simplicité**.

Exemple dup2()



Exemple dup2() (suite)

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

#define LECTURE 0
#define ECRITURE 1

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

if (pipe(fd) == -1) {
        perror("Erreur dans pipe()");
        exit(EXIT_FAILURE);
}

switch (fork()) {
case -1 : /* erreur */
        perror("Erreur dans fork()");
        exit(EXIT_FAILURE);
case 0 : /* processus fils, ls , ecrivain */
        //Etape 1.1
        close(fd[LECTURE]);
        /* dup2 va brancher le cote ecriture du tuyau */
        /* comme sortie standard du processus courant */
        //Etape 1.2
        if (dup2(fd[ECRITURE], STDOUT_FILENO) == -1) {
            perror("Erreur dans dup2()");
        }
        /* on ferme le descripteur qui reste pour */
        /* << eviter les fuites >> ! */
        close(fd[ECRITURE]);
        /* ls en ecrivant sur stdout envoie en fait dans le
        /* tuyau sans le savoir */
        if (execlp("ls", "ls", NULL) == -1) {
            perror("Erreur dans execlp()");
            exit(EXIT_FAILURE);
        }
default : /* processus pere, wc , lecteur */
        //Etape 2.1
        close(fd[ECRITURE]);
        /* dup2 va brancher le cote lecture du tuyau */
        /* comme entree standard du processus courant */
        //Etape 2.2
        if (dup2(fd[LECTURE], STDIN_FILENO) == -1) {
            perror("Erreur dans dup2()");
        }
        /* on ferme le descripteur qui reste */
        close(fd[LECTURE]);
        /* wc lit lâ€™entree standard, et les donnees */
        /* quâ€™il recoit proviennent du tuyau */
        if (execlp("wc", "wc", "-l", NULL) == -1) {
            perror("Erreur dans execlp()");
            exit(EXIT_FAILURE);
        }
}
exit(EXIT_SUCCESS);
}
```

Synchronisation de deux processus au moyen d'un tuyau

- **Rappel** : Un processus tentant de lire un **tuyau vide** est **suspendu** jusqu'à ce que des données soient disponibles.
- ==> On peut synchroniser le processus **lecteur** sur le **rythme** du processus **écrivain**.

Exemple (synch.c)

```c
#define LECTURE 0
#define ECRITURE 1

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

if (pipe(tuyau) == -1) {
    perror("Erreur dans pipe()");
    exit(EXIT_FAILURE);
}

switch (fork()) {
case -1 : /* erreur */
        perror("Erreur dans fork()");
        exit(EXIT_FAILURE);
case 0 : /* processus fils, lecteur */
        close(tuyau[ECRITURE]);
        /* on lit les caracteres un a un */
        while (read(tuyau[LECTURE], &car, 1) != 0 ) {
            putchar(car);
            /* affichage immediat du caracter lu */
            fflush(stdout);
        }
        close(tuyau[LECTURE]);
        putchar('\n');
        exit(EXIT_SUCCESS);
default : /* processus pere, ecrivain */
    close(tuyau[LECTURE]);
    for (i = 0; i < 10; i++) {
            /* on obtient le caractere qui represente le chiffre i */
            /* en prenant le i-eme caractere a partir de '0' */
            car = '0'+i;
            /* on ecrit ce seul caractere */
            write(tuyau[ECRITURE], &car, 1);
            sleep(1); /* et on attend 1 sec */
    }
    close(tuyau[ECRITURE]);
    exit(EXIT_SUCCESS);
}
}
```

Contexte des Tubes Nommés

- **Inconvéniont** :
  - Les tuyaux ne permettent qu'une communication **unidirectionnelle**
  - Seulement les processus partageant un **même ancêtre** peuvent communiquer via un tuyau.

- **Solution** :

  1. **Tubes nommées** (**FIFOs**) : les tubes nommées sont des fichiers **spéciaux** qui se comportent comme des tuyaux une fois ouverts. Les données sont envoyées directement sans être stockées sur disque.
  2. **Sockets** : peuvent être utilisés dans le but de permettre une communication **bidirectionnelle** entre divers processus sur la même ou sur des machines reliées par réseaux.

Avantatges des Tubes Nommés

Les tubes de communication nommés offrent les avantages suivants :

- Ils ont chacun un **nom** qui existe dans le **système de fichiers** (une entrée dans la Table des fichiers) ;
- Ils sont considérés comme des **fichiers spéciaux** ;
- Ils peuvent être **utilisés** par des processus **indépendants**, à condition qu'ils s'exécutent **sur une même machine**.
- Ils existeront jusqu'à ce qu'ils soient **supprimés explicitement** ;
- Ils sont de **capacité** plus **grande**.

Création des Tubes Nommés

- **mkfifo**() : Permet de créer un tube nommé ;
- Ouverture avec **open**() ou **fopen**() et s'utilise au moyen des fonctions d'**entrées/sorties** classiques.

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

- **nom** correspond au nom du fichier ;
- **mode** correspond aux droits d'accès associés au tube.

Exemple

```
menthe22> mkfifo fifo
menthe22> ls -l fifo
prw-r--r--  1 in201    in201    0 Jan 10 17:22 fifo
```

```
menthe22> echo coucou > fifo &
[1] 25312
menthe22> cat fifo
[1]  + done       echo coucou > fifo
coucou
```

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

int retour;
retour = mkfifo("fifo", 0644);
if ( retour == -1 ) {
   /* erreur : le tuyau nomme n'a pas pu etre cree */
}
```