

# M3101 – Principes des systèmes d'exploitation

## Tubes et Redirections

Damien Genthial, Clément Raïevsky



Département Informatique

2019

# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

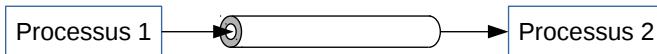
# Plan du chapitre

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Tube – Pipe

## Définition

- Canal de communication direct entre processus
- Unidirectionnel : deux extrémités **distinctes** :
  - Extrémité de **lecture** – `read end`
  - Extrémité d'**écriture** – `write end`
- Accessible de la même manière qu'un fichier
- Les données écrites dans un tube sont stockées dans un tampon noyau jusqu'à leur lecture





# Création

Deux types de tubes :

- Tube classique – *Pipe*
- Tube nommé ou FIFO

Une fois créés, strictement identiques

```
int mkfifo(char* pathname, mode_t mode)
```

- Crée un fichier spécial FIFO dans le système de fichier
- Ouverture en lecture ou écriture avec `open & Co.`
- L'ouverture en lecture bloque tant que le fichier n'est pas également ouvert en écriture
- Et inversement
- Retourne 0 en cas de succès, -1 sinon
- Aucun contenu correspondant au fichier sur le support

# Création

## pipe

```
int pipe(int pipefd[2])
```

- Crée un tube
- Crée **deux** descripteurs de fichier
- Les stocke dans **pipefd**
- Retourne 0 en cas de succès, -1 sinon
- pipefd[0] : extrémité de **lecture** → read
- pipefd[1] : extrémité d'**écriture** → write

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

# Écriture

```
int descripteurs[2];  
pipe(descripteurs);  
  
int nbEcrits;  
char buf[MAX] = "Bonjour, \u00a0monde."  
  
nbEcrits = write(descripteurs[1], buf, strlen(buf));
```

Bloquant si le tube est plein

# Lecture

```
int descripteurs[2];  
pipe(descripteurs);  
  
int nbLus;  
char buf[MAX]; // Pour stocker ce qui sera lu  
nbLus = read(descripteurs[0], buf, MAX);
```

Bloquant si le tube est vide

# Communication Interprocessus par Tubes

Exemple Minimal (ne pas utiliser tel quel)

(1/2)

```
1 | int descripteurs[2];
2 | pipe(descripteurs); // Les descripteurs de fichiers survivent a un fork
3 |
4 | pid_t childPid = fork();
5 |
6 | if (childPid > 0){
7 |     int nbEcrits;
8 |     char buf[MAX] = "Bonjour, \u0026monde.";
9 |
10 |     nbEcrits = write(descripteurs[1], buf, strlen(buf));
11 | } else if (childPid == 0){
12 |     int nbLus;
13 |     char buf[MAX];
14 |
15 |     nbLus = read(descripteurs[0], buf, MAX);
16 | }
```

# Communication Interprocessus par Tubes

Exemple Minimal (ne pas utiliser tel quel)

(2/2)

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

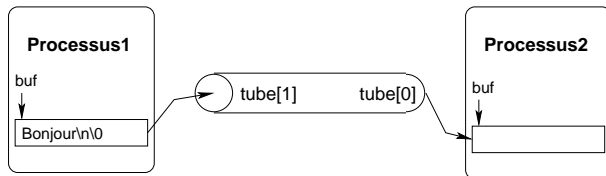
pid_t childPid = fork();

if (childPid > 0){
    int nbEcrits;
    char buf[MAX] = "Bonjour\n";

    nbEcrits = write(descripteurs[1], buf, strlen(buf));
}

else if (childPid == 0){
    int nbLus;
    char buf[MAX];

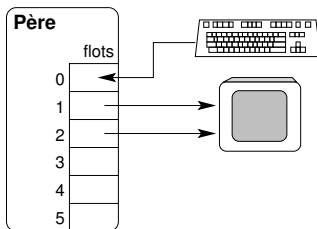
    nbLus =
        read(descripteurs[0], buf, MAX);
}
```



# Communication Interprocessus par Tubes

## Pas à pas

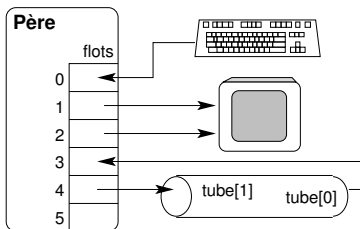
```
int main(void) {  
    pid_t childPid;  
    int tube[2];  
    char msg[8];  
    pipe(tube);  
    childPid = fork();  
    if (childPid == 0) {  
        close(tube[1]);  
        read(tube[0], msg, 8);  
    } else {  
        close(tube[0]);  
        write(tube[1],  
            "Bonjour", 8);  
    }  
    return EXIT_SUCCESS;  
}
```



# Communication Interprocessus par Tubes

## Pas à pas

```
int main(void) {
    pid_t childPid;
    int tube[2];
    char msg[8];
    pipe(tube);
    childPid = fork();
    if (childPid == 0) {
        close(tube[1]);
        read(tube[0], msg, 8);
    } else {
        close(tube[0]);
        write(tube[1],
            "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```

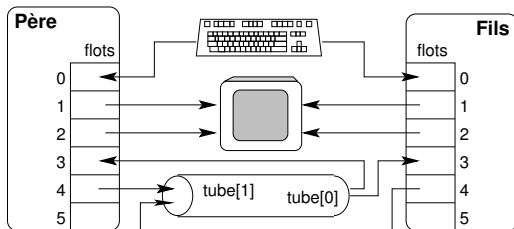




# Communication Interprocessus par Tubes

## Pas à pas

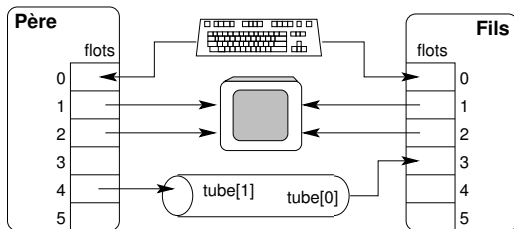
```
int main(void) {
    pid_t childPid;
    int tube[2];
    char msg[8];
    pipe(tube);
    childPid = fork();
    if (childPid == 0) {
        close(tube[1]);
        read(tube[0], msg, 8);
    } else {
        close(tube[0]);
        write(tube[1],
            "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```



# Communication Interprocessus par Tubes

## Pas à pas

```
int main(void) {
    pid_t childPid;
    int tube[2];
    char msg[8];
    pipe(tube);
    childPid = fork();
    if (childPid == 0) {
        close(tube[1]);
        read(tube[0], msg, 8);
    } else {
        close(tube[0]);
        write(tube[1],
            "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```



## Conditions limites

- write sur un tube plein → bloquant
- read sur un tube vide → bloquant
- write sur un tube dont l'extrémité de lecture est fermée → SIGPIPE
- read sur un tube dont l'extrémité d'écriture est fermée → EOF

# Redirection des entrées-sorties

- 1 Les tubes Unix
  - Définition
  - Création
- 2 Écriture
- 3 Lecture
- 4 Conditions d'arrêt
- 5 Redirection des entrées-sorties
  - Rappel
  - La fonction dup2
  - Redirection de l'entrée
  - Redirection de la sortie
- 6 Redirections dans des tubes

# Les entrées-sorties standard sont des fichiers

## Rappel

fd ↔ FILE\*

- 0 ↔ stdin : Entrée standard
- 1 ↔ stdout : Sortie standard
- 2 ↔ stderr : Sortie des erreurs standard

# La fonction dup2

Dupliquer un descripteur de fichier

```
int dup2(int oldfd, int newfd)
```

- Duplique, copie le descripteur de fichier `oldfd`
- Attribue à cette copie le descripteur de fichier `newfd`
- Dans le cas où `newfd` était utilisé, il est fermé

Après un appel réussi à `dup2`, `oldfd` et `newfd` référence la même entrée dans la table des fichiers ouverts

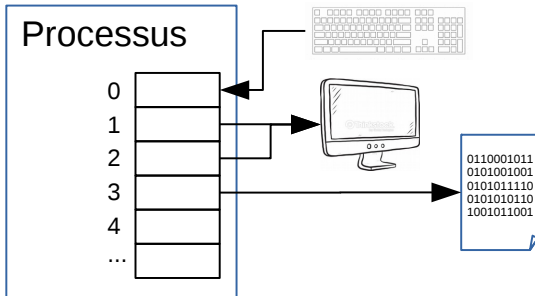
Les opérations modifiant la position courante s'appliquent aux deux descripteurs de fichier

# La fonction dup2

Dupliquer un descripteur de fichier

Exemple : redirection de la sortie standard des erreurs vers un fichier :

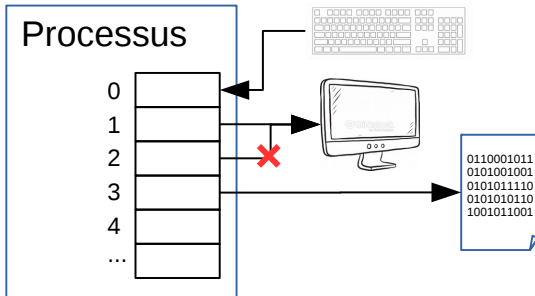
`dup2(3, 2);`



# La fonction dup2

Dupliquer un descripteur de fichier

Exemple : redirection de la sortie standard des erreurs vers un fichier :  
`dup2(3, 2);`

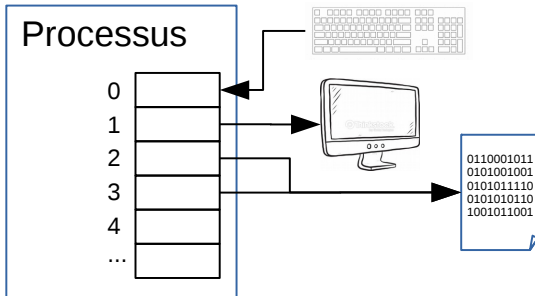




# La fonction dup2

Dupliquer un descripteur de fichier

Exemple : redirection de la sortie standard des erreurs vers un fichier :  
`dup2(3, 2);`



# Redirection de l'entrée

(1/2)

Entrée d'un programme : descripteur de fichier 0

À partir d'un **fichier**

```
int fileFd = open("unFichier.txt", O_RDONLY);
int tab[3];

dup2(fileFd, 0);

/* Lectures dans le fichier */
scanf("%d_%d_%d\n", &tab[0], &tab[1], &tab[2]);
fscanf(stdin, "%d_%d_%d\n", &tab[0], &tab[1], &tab[2]);
```

Le descripteur de fichier 0 référence la même **description de fichier ouvert** que fileFd

# Redirection de l'entrée

(2/2)

Entrée d'un programme : descripteur de fichier 0

À partir d'un **tube**

```
int descripteurs[2];  
  
pipe(descripteurs);  
  
dup2(descripteurs[0], 0);  
  
/* Lecture a l'extremite de lecture du tube */  
char buf[MAX];  
int nbMatched = scanf("%d_%d_%d", &tab[0], &tab[1], &tab[2]);
```

Après le `dup2`, le descripteur de fichier 0 référence la même **description de fichier ouvert** que l'extrémité de lecture du tube

# Redirection de la sortie

Sortie d'un programme : descripteur de fichier 1

Vers d'un tube

```
int descripteurs[2];  
  
pipe(descripteurs);  
  
dup2(descripteurs[1], 1);  
  
/* Écriture dans le tube */  
printf("Envoi dans le tube\n");
```

Après le `dup2`, le descripteur de fichier 1 référence la même **description de fichier ouvert** que l'extrémité d'écriture du tube

# Combinaison de la redirection et des tubes

## Utilité :

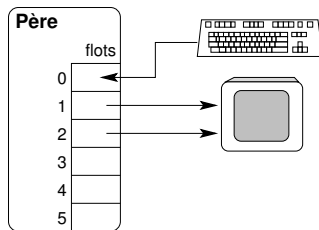
- Sa création
- Ses entrées
- Ses sorties

Contrôler un processus

sans avoir accès à son code

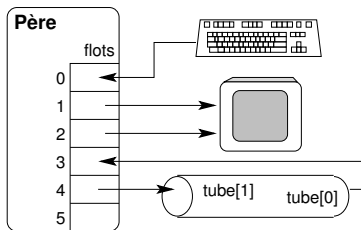
# Redirection des entrées / sorties standard vers des tubes

```
int main(void) {  
    pid_t id;  
    int tube[2];  
    char msg[8];  
    pipe(tube);  
    id = fork();  
    if (id == 0) {  
        close(tube[1]);  
        dup2(tube[0], 0);  
        read(0, msg, 8);  
    } else {  
        close(tube[0]);  
        dup2(tube[1], 1);  
        write(1, "Bonjour", 8);  
    }  
    return EXIT_SUCCESS;  
}
```



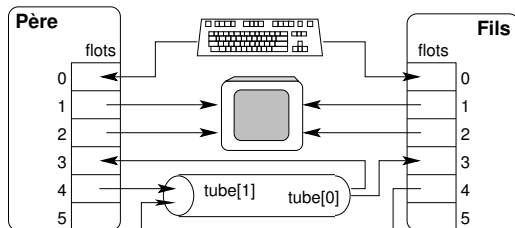
# Redirection des entrées / sorties standard vers des tubes

```
int main(void) {
    pid_t id;
    int tube[2];
    char msg[8];
    pipe(tube);
    id = fork();
    if (id == 0) {
        close(tube[1]);
        dup2(tube[0], 0);
        read(0, msg, 8);
    } else {
        close(tube[0]);
        dup2(tube[1], 1);
        write(1, "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```



# Redirection des entrées / sorties standard vers des tubes

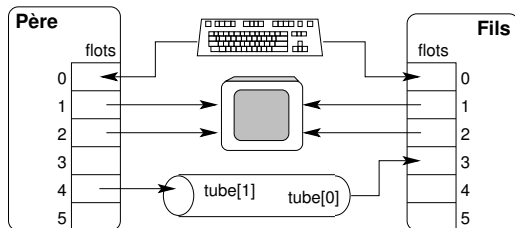
```
int main(void) {
    pid_t id;
    int tube[2];
    char msg[8];
    pipe(tube);
    id = fork();
    if (id == 0) {
        close(tube[1]);
        dup2(tube[0], 0);
        read(0, msg, 8);
    } else {
        close(tube[0]);
        dup2(tube[1], 1);
        write(1, "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```





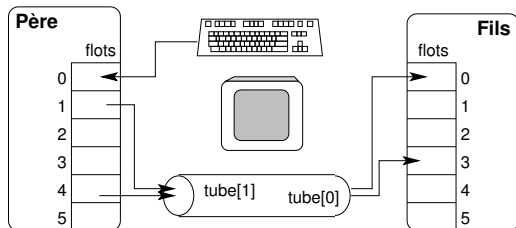
# Redirection des entrées / sorties standard vers des tubes

```
int main(void) {  
    pid_t id;  
    int tube[2];  
    char msg[8];  
    pipe(tube);  
    id = fork();  
    if (id == 0) {  
        close(tube[1]);  
        dup2(tube[0], 0);  
        read(0, msg, 8);  
    } else {  
        close(tube[0]);  
        dup2(tube[1], 1);  
        write(1, "Bonjour", 8);  
    }  
    return EXIT_SUCCESS;  
}
```



# Redirection des entrées / sorties standard vers des tubes

```
int main(void) {
    pid_t id;
    int tube[2];
    char msg[8];
    pipe(tube);
    id = fork();
    if (id == 0) {
        close(tube[1]);
        dup2(tube[0], 0);
        read(0, msg, 8);
    } else {
        close(tube[0]);
        dup2(tube[1], 1);
        write(1, "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```



# Redirection des entrées / sorties standard vers des tubes

```
int main(void) {
    pid_t id;
    int tube[2];
    char msg[8];
    pipe(tube);
    id = fork();
    if (id == 0) {
        close(tube[1]);
        dup2(tube[0], 0);
        read(0, msg, 8);
    } else {
        close(tube[0]);
        dup2(tube[1], 1);
        write(1, "Bonjour", 8);
    }
    return EXIT_SUCCESS;
}
```

