Chapitre 3

Les tubes

Les tubes (ordinaires et nommés)

- Les tubes sont :
  - un mécanisme de communication
  - orienté flot de caractères
  - entre deux processus locaux
- Les tubes sont des FIFO :
  - l'ordre des caractères en entrée est conservé en sortie
  - la lecture est « destructrice »

- Par défaut, *la lecture* dans un tube vide est bloquante
- Par défaut, *l'écriture* dans un tube plein est bloquante
- Les tubes ont une taille finie de l'ordre de quelques Koctets (4 KiB sous Linux)

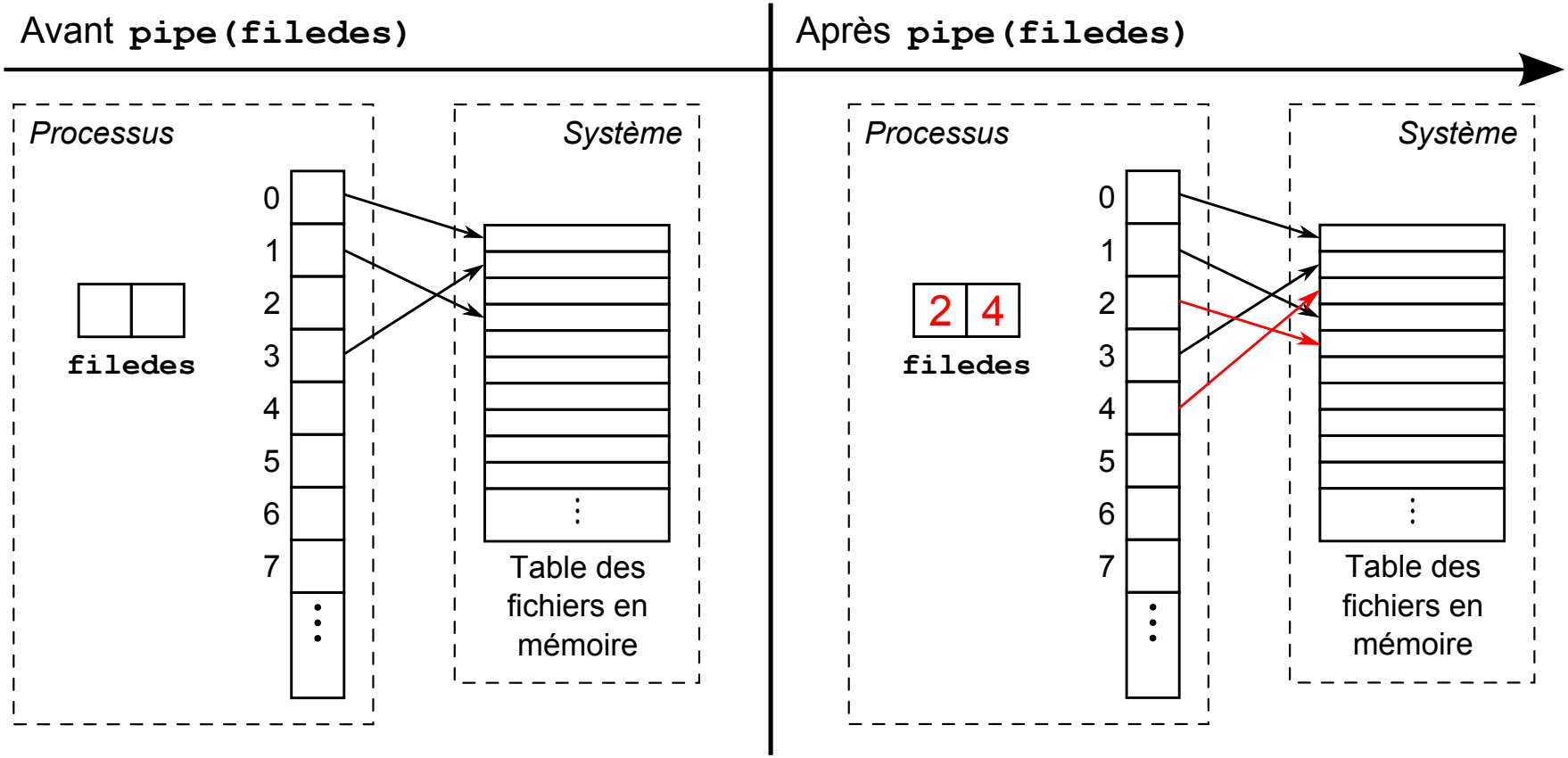
Les tubes ordinaires

- Un processus ne peut utiliser que les tubes :
  - qu'il a créés lui-même (avec `pipe()`)
  - qu'il a hérités de son père (héritage des descripteurs à travers `fork()` et `exec()`)
- En général deux processus (créés par `fork()`) se partagent le tube, et utilisent les appels `read()` et `write()` pour se transmettre des données.
- Aussi appelés *tubes volatiles*.

L'appel système `pipe`

- `#include <unistd.h>`
- `#int pipe(int filedes[2]);`
- `pipe()` crée une paire de descripteurs de fichiers
- Chaque descripteur pointe sur un inode de tube
- Les descripteurs sont placés par `pipe()` dans un tableau :
  - `filedes[0]` : descripteur pour la lecture
  - `filedes[1]` : descripteur pour l'écriture

L'appel système `pipe`



Lecture dans un tube

- `filedes[0]` est le descripteur réservé à la lecture
- La lecture est réalisée via l'appel système `read()` :

```
#define TAILLE_BUF 1024
char buffer[TAILLE_BUF];
int nbLus;
nbLus = read(filedes[0], buffer, TAILLE_BUF);
```

Lecture dans un tube

Comportement de l'appel `read()`

si *le tube n'est pas vide et contient* `taille` *caractères* **alors**
- Lecture de `nbLus = min(taille, TAILLE_BUF)` caractères

sinon si *le tube est vide* **alors**
- **si** *le nombre d'écrivains est nul* **alors**
  - c'est la fin de fichier et `nbLus == 0`
- **sinon si** *le nombre d'écrivains est non nul* **alors**
  - **si** *lecture bloquante* **alors**
    - sommeil
  - **sinon si** *lecture non bloquante* **alors**
    - **suivant** *l'indicateur* **faire**
      - **cas où** `O_NONBLOCK` *:* `nbLus==-1` et `errno==EAGAIN`
      - **cas où** `O_NDELAY` *:* `nbLus==0`

Écriture dans un tube

- `filedes[1]` est le descripteur réservé à l'écriture
- L'écriture est réalisée via l'appel système `write()` :

```
#define TAILLE_BUF 1024
montype_t buffer[TAILLE_BUF];
int nbEcrits;
int n; /* n < (TAILLE_BUF * sizeof(montype_t)) */
nbEcrits = write(filedes[1], buffer, n);
```

- L'écriture est *atomique* si le nombre de caractères à écrire est inférieur à `PIPE_BUF`, la taille du tube dans le système (voir `<limits.h>`)

Écriture dans un tube

Comportement de l'appel `write()`

si *le nombre de lecteurs est nul* **alors**
- Envoi du signal `SIGPIPE` à l'écrivain

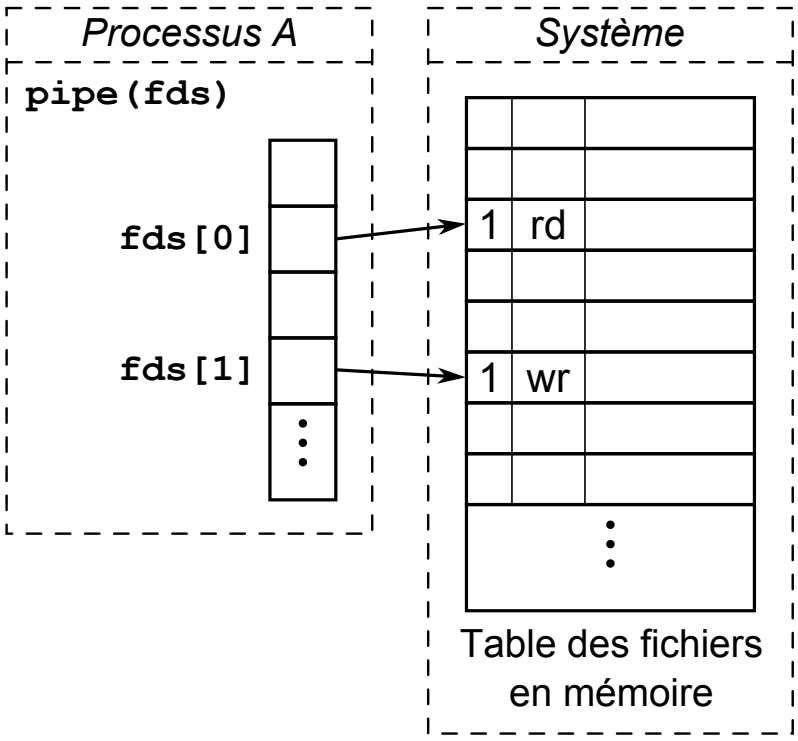
sinon si *le nombre de lecteurs est non nul* **alors**
- **si** *l'écriture est bloquante* **alors**
  - `write()` ne retourne que quand les `n` caractères ont été écrits dans le tube
- **sinon si** *l'écriture est non bloquante* **alors**
  - **si** `n > PIPE_BUF` **alors**
    - retour avec un nombre < `n`, éventuellement -1
  - **sinon si** `n <= PIPE_BUF` **alors**
    - **si** `n` *emplacements libres* **alors**
      - écriture de `n` caractères (et `nbEcrits=n`)
    - **sinon**
      - retour -1 ou 0

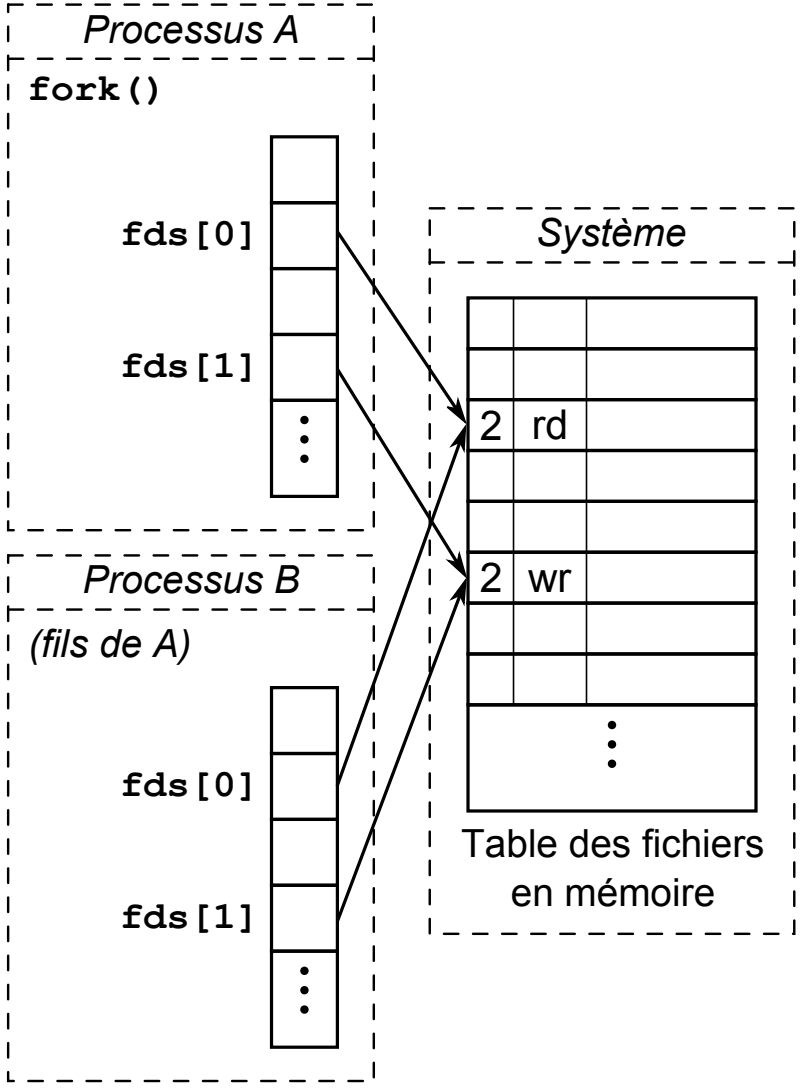
Lecture/Écriture dans un tube

Remarques importantes :

- L'algorithme de lecture (resp. d'écriture) précédent dépend du nombre d'écrivains (resp. lecteurs). Il est impératif que ce nombre soit à jour.
- Tout processus disposant d'un descripteur sur une extrémité d'un tube qu'il n'utilise pas doit le fermer (avec `close()`).
- Puisque les descripteurs sur un tube ne peuvent être obtenus que par l'appel système `pipe()` ou par héritage :
  - il existe obligatoirement un lien de parenté entre deux processus communicant à l'aide d'un tube classique.

Tubes ordinaires et `fork()`





Exemple

Cas le plus simple : un processus père émet un entier vers son fils.

```
int main(void) {
  int tube[2];

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

  switch(fork()) {
    case -1 : perror("fork"); exit(errno);
    case 0 :  // le fils
      close(tube[1]);
      codeDuFils(tube);
      exit(0);
    default : // le pere
      close(tube[0]);
      codeDuPere(tube);
  }
}
```

Exemple

```
void codeDuFils(int tube[2]) {
   int d;
   read(tube[0], &d, sizeof(int));
   printf("fils : lecture de %d\n", d);
   close(tube[0]);
   exit(0);
}

void codeDuPere(int tube[2]) {
   int e = 10;
   write(tube[1], &e, sizeof(int));
   close(tube[1])
   wait(NULL);
   exit(0);
}
```

Tubes nommés

- Les tubes nommés sont des tubes ayant une référence dans le système de fichier.
- Création :

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

- Le tube nommé existera tant que :
  - son entrée dans le système de fichiers ne sera pas supprimée
  - et que le nombre de processus l'ayant ouvert n'est pas nul.
- **Remarques :**
  - Puisqu'un tube nommé est ouvert à l'aide de son nom dans le système de fichier, il peut n'exister aucun lien de parenté entre les processus l'utilisant.
  - Il faut bien sûr ouvrir le fichier par `open()` avant de l'utiliser.
  - L'ouverture d'un tube nommé se fait **exclusivement** en mode `O_RDONLY` ou en mode `O_WRONLY` afin de pouvoir comptabiliser le nombre de lecteurs et d'écrivains.

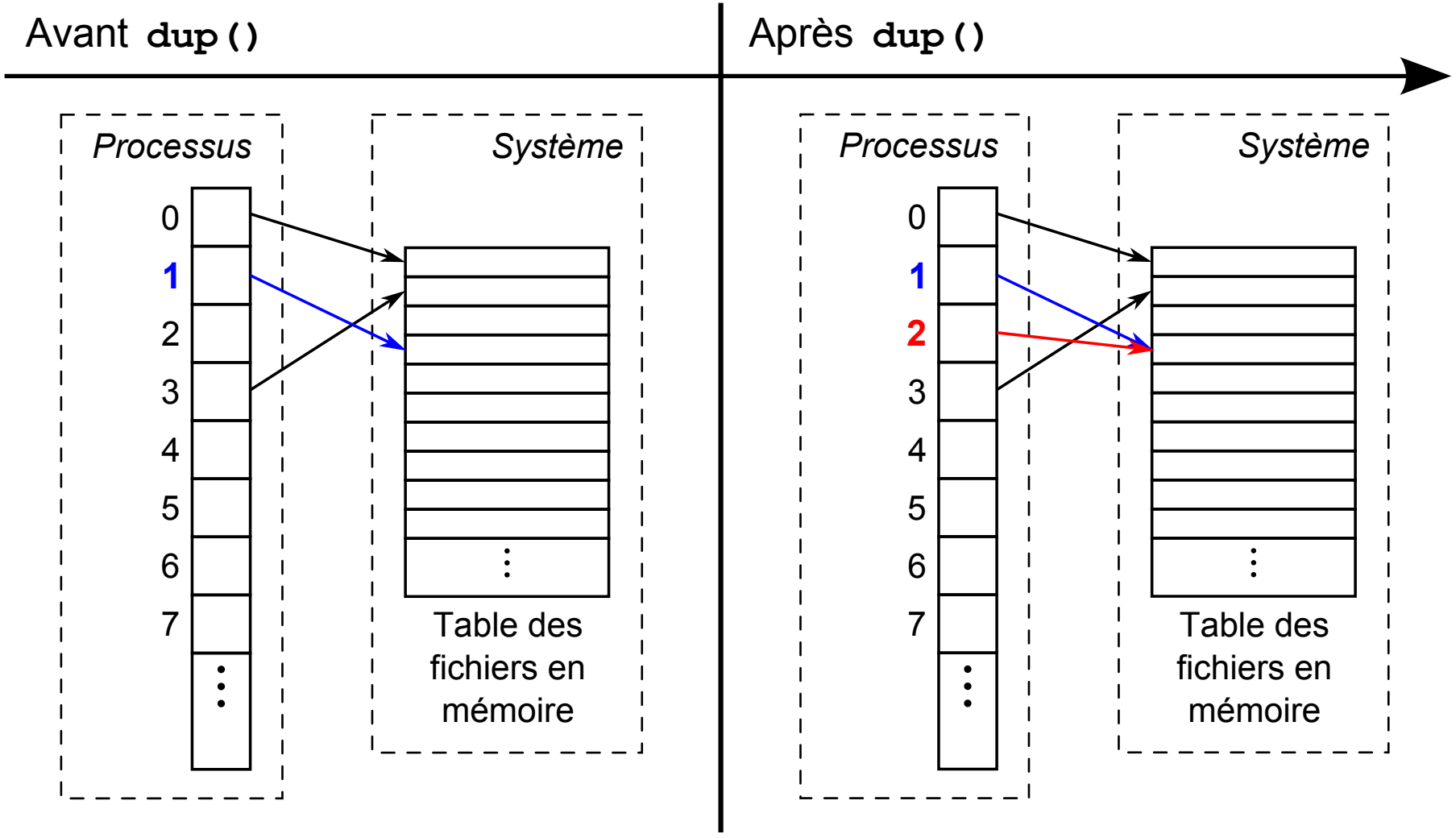
Tubes nommés

- Ouverture en **mode bloquant** d'un tube nommé :
  - L'appel à `open()` est bloquant en lecture (resp. écriture) : le processus attend qu'un autre processus ouvre le tube en écriture (resp. lecture).
  - L'ouverture bloquante se termine de façon synchrone pour les deux processus.
  - Il y a automatiquement synchronisation des processus qui ouvrent en mode bloquant un tube nommé.
- En **mode non bloquant** (`O_NONBLOCK` ou `O_NDELAY`) :
  - L'ouverture en *lecture* (`O_RDONLY`) réussit dans tous les cas.
  - L'ouverture en *écriture* (`O_WRONLY`) ne fonctionne que si un processus a déjà ouvert le tube en lecture.
    - Écrire dans un tube sans lecteur engendrerait un signal `SIGPIPE` (tube détruit).
  - Toutes les opérations de lecture/écriture qui suivent sont alors non bloquantes.

Duplication de descripteurs : dup()

- `#include <unistd.h>`
- `int dup(int oldfd);`
- Crée une copie du descripteur de fichier `oldfd`
  - Utilise le plus petit descripteur de fichier libre
  - Renvoie le descripteur de fichier désignant la copie
- `oldfd` et le descripteur de fichier renvoyé partagent :
  - Les pointeurs de position dans le fichier (lseek)
  - Les verrous, les drapeaux (read/write, eof...)
    - Sauf le flag « close-on-exec »

Fonctionnement de dup()

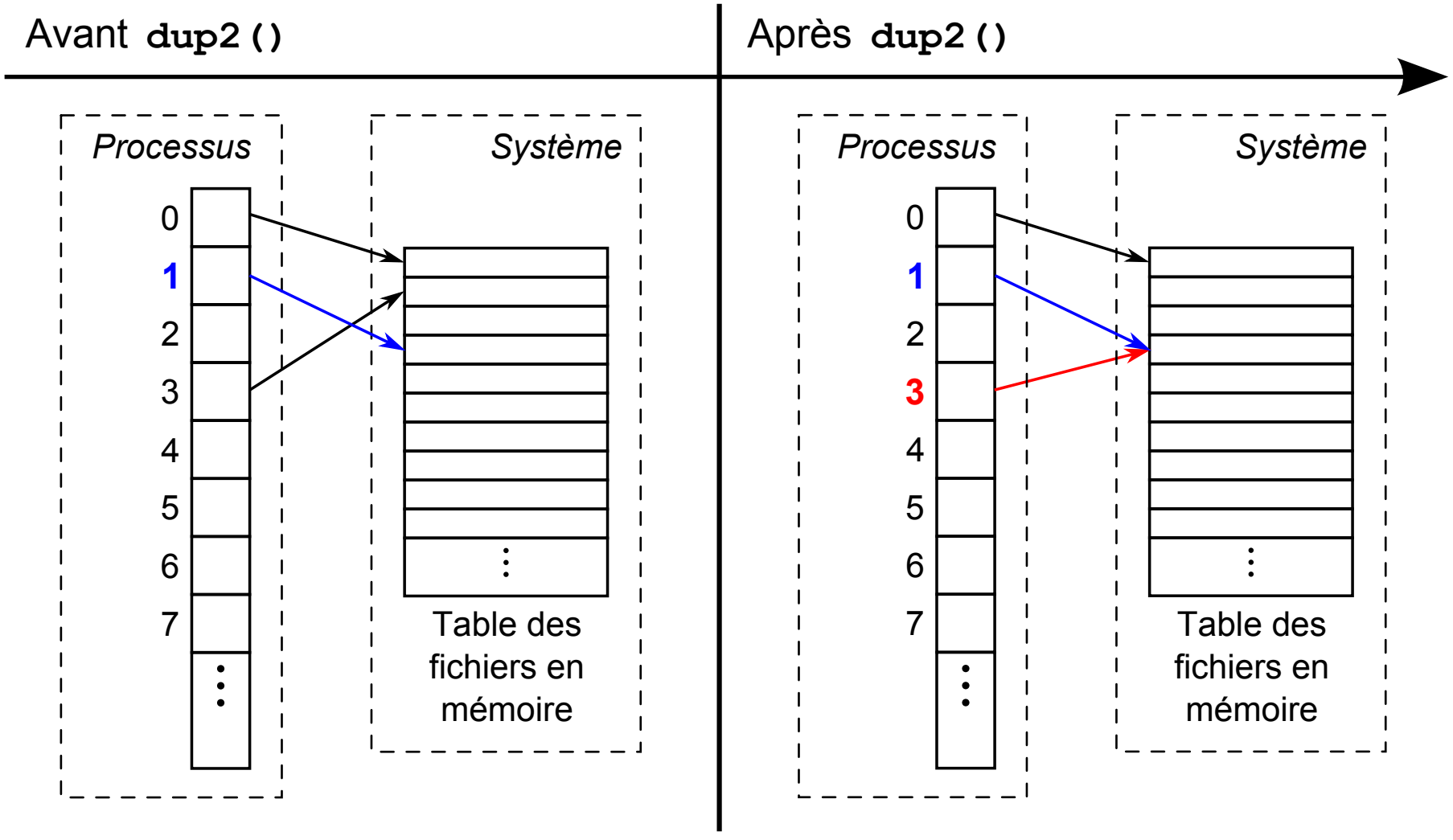


dup(1) => 2

Duplication *sur* un descripteur : `dup2()`

- `#include <unistd.h>`
- `int dup2(int oldfd, int newfd);`
- Similaire à `dup()` mais :
  - Crée une copie de `oldfd`
    - S'arrête là si `oldfd` n'est pas un descripteur valide
  - Ferme `newfd` si il était ouvert
  - Écrase le descripteur `newfd` par une copie de `oldfd`

Fonctionnement de dup2()



dup2(1,3) => 3

Duplication de descripteur en pratique

- A l'aide de `dup()` et `dup2()`, un processus peut agencer comme bon lui semble ses descripteurs de fichiers.
- Il peut par exemple associer l'extrémité d'un tube à une de ses entrée/sortie standard :

```
dup2(tube[1], 1); // pour rediriger sa sortie standard
close(tube[1]);   // vers le tube

close(1);       // idem avec dup (le descripteur 0
dup(tube[1]); // ne doit pas etre libre)


saveStdout = dup(1); // pour rediriger temporairement sa
                     // sortie standard vers le tube
dup2(tube[1], 1);
close(tube[1]);
...
dup2(saveStdout, 1); // restauration de la sortie standa
```

Petit exercice

- Soit un programme du type "amarok" ou "rythmbox" (programmes du type jukebox). Nous ne nous attacherons pas à écrire ce jukebox, mais à le commander : ces programmes créent un "tube nommé" qui permet de commander le processus depuis un programme extérieur. On peut ainsi :
  - Écrire dans le tube "/tmp/commande", pour donner une commande au jukebox (augmenter volume, morceau suivant, . . .)
  - Lire dans le tube "/tmp/info", pour trouver le nom du morceau en cours.

- Écrivez les programmes suivants :
  - "Afficheur" : écrit sur la sortie standard le nom du morceau joué
  - "Télécommande" qui lit une commande (sous forme de deux entiers) sur la sortie standard et la transmet au jukebox.