Les processus et les pipes en C (II)

Cédric Meuter, Eythan Levy

25 novembre 2005

1 Rappel

Lors de la séance précédente, nous avons passé en revue une série de *system calls* permettant de créer des processus, d'exécuter des programmes et permettant également à deux processus d'une même machine de communiquer entre eux. Pour rappel, les signatures de ces *system calls* sont repris ci-après:

- Création de processus:

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

pid_t fork();
```

- Exécution de programmes:

```
#include <unistd.h>

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

- Attente d'un processus:

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

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- Création de *pipes* anonymes:

```
#include <unistd.h>

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

- Création de *pipes* nommés:

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

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

2 Exercices

- Téléchargez les sources des exercices de la séance 5 (disponibles sur la page du TP[1]). Compilez et testez ces programmes.
- Ecrivez un programme C qui prend en argument un nom de fichier et qui lance l'éditeur XEmacs avec ce fichier (il suffit de passer le nom du fichier en argument à XEmacs). Le programme devra ensuite attendre que l'éditeur se termine et afficher ensuite le code de retour du processus.

3 Les *file descriptors*

Chaque processus, dans sa zone mémoire, possède une *file descriptor table*. Chaque entrée dans cette table est indexée par un entier (un *file descriptor*) et contient un pointeur vers une entrée dans la *file table*. Cette *file table* contient une entrée par ouverture de fichier, et est commune à tous les processus. Notez que dans chaque processus, les *file descriptors* 0, 1 et 2 correspondent respectivement au *standard input*, au *standard output* et au *standard error*.

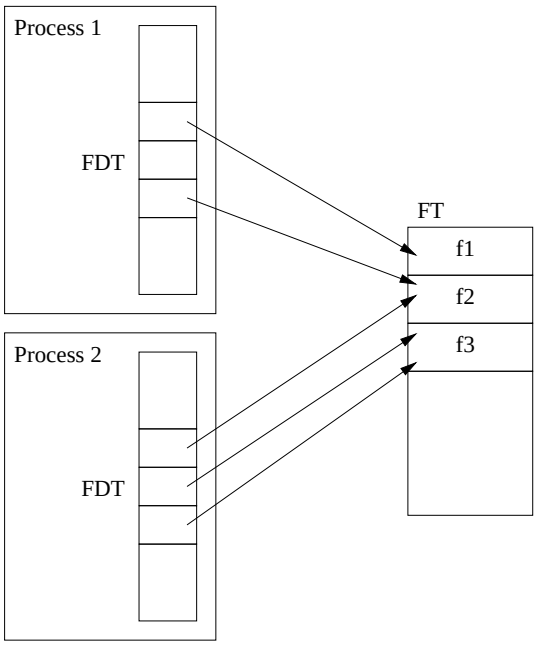

Figure 1: Les *file descriptors*

Comme on peut le constater sur la figure 1, dans certains cas, deux *file descriptors* de deux processus différents peuvent pointer vers le même fichier (c'est le cas de f2). Ceci peut se produire dans le cas de `fork`. En effet, le `fork` duplique les processus et donc également la *file descriptor table*. Il se peut également que deux *file descriptors* du même processus pointent vers le même fichier, et ce grâce au *system calls* `dup` et `dup2`.

```
#include <unistd.h>

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

[1] `http://www.ulb.ac.be/di/ssd/cmeuter/archi/`

Le *system call* `dup` renvoie une copie du *file descriptor* passé en paramètre, c'est-à-dire une autre entrée dans le *file descriptor table* pointant vers le même ouverture de fichier. Le *file descriptor* renvoyé est garanti d'être le plus petit correspondant à une entrée libre de la *file descriptor table*. Ceci permet, par exemple de rediriger le *standard output* d'un programme vers un fichier. Pour cela, il faut d'abord fermer le *standard output* avec `close`. On duplique ensuite le *file descriptor* correspondant au fichier voulu et, comme `dup` garanti de choisir le plus petit *file descriptor* disponible, c'est le *file descriptor* correspondant au *standard output* (1) qui sera choisi. Mais attention, si le *standard input* a été fermé également, c'est ce *file descriptor* (0), étant plus petit, qui sera choisi. Pour éviter ce genre surprises, on préférera utiliser le *system call* `dup2`, qui copie l'entrée `oldfd` dans l'entrée `newfd`. Le *file descriptor* `newfd` est fermé préalablement si nécessaire.

4 Exercices

- Ecrivez un programme qui affiche *hello world* sur le *standard output* (en utilisant `printf`, mais dont le *standard output* a été préalablement redirigé (à l'aide de `dup` ou `dup2`) vers un fichier dont le nom est passé en argument du programme.

- Ecrivez une fonction `spawn` qui prend les même paramètres que `execvp`, ainsi qu'un tableau de deux entiers `fd` (un peu comme pour `pipe`) et qui crée un processus. Le processus fils devra ensuite exécuter le programme spécifié en paramètre (à l'aide de `execvp`). De plus, a l'aide de *pipes*, le *standard input* du processus fils sera redirigé vers le *file descriptor* `fd[1]` et son *standard output* vers le *file descriptor* `fd[0]`, comme illustré à la figure 2.

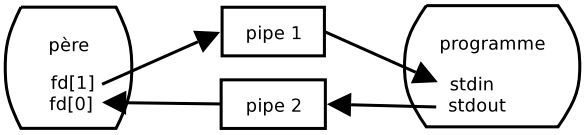

Figure 2: Redirection via des *pipes*

- Utilisez la fonction `spawn` pour exécuter un *shell* (`/bin/bash`). Envoyez-lui la commande `ls -la` (sans oublier le `\n` de fin) via `fd[1]`. Lisez le résultat sur `fd[0]` et affichez le à l'écran.