TP N°6
Les tubes

Sylvain Chevallier

Préambule

Notions : `pipe ()`, `fork ( )`, `execl ()` et `dup ()`.

Prochain TD : Les signaux et les IPC.

Rappel

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

La fonction `pipe()` crée un tube et retourne deux descripteurs de fichiers : `pipefd[0]` est l'entrée de lecture du tube et `pipefd[1]` est l'entrée d'écriture du tube.

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

Les fonctions `dup()` et `dup2()` créent une copie du descripteur de fichier `oldfd` et renvoie comme valeur de retour la copie du descripteur de fichier. La fonction `dup2()` permet de spécifier quel descripteur de fichier `newfd` doit être utilisé comme copie : si ce descripteur de fichier `newfd` est déjà utilisé, il est fermé puis `oldfd` est copié pour donner un nouveau `newfd`. Attention, après un appel à `dup()` ou `dup2`, `oldfd` n'est pas fermé, deux descripteurs de fichiers pointent donc vers le même fichier.

1 Utilisation des tubes pour la communication

Le but de cet exercice est de manipuler en C les tubes (pipe) UNIX.
Voici l'exemple d'utilisation indiqué dans la page `man pipe` :

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

int main(int argc, char *argv[])
{
  int pipefd[2];
  pid_t cpid;
  char buf;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s <string>\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
  }

  cpid = fork();
  if (cpid == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
  }
```

```
28
   if (cpid == 0) {    /* Child reads from pipe */
30   close(pipefd[1]);          /* Close unused write end */

32   while (read(pipefd[0], &buf, 1) > 0)
       write(STDOUT_FILENO, &buf, 1);
34
   write(STDOUT_FILENO, "\n", 1);
36   close(pipefd[0]);
   _exit(EXIT_SUCCESS);
38
   } else {            /* Parent writes argv[1] to pipe */
40   close(pipefd[0]);          /* Close unused read end */
   write(pipefd[1], argv[1], strlen(argv[1]));
42   close(pipefd[1]);          /* Reader will see EOF */
   wait(NULL);                /* Wait for child */
44   exit(EXIT_SUCCESS);
   }
46 }
```

Analysez le fonctionnement de cet exemple. Vous indiquerez un schéma de la table de descripteurs de fichiers pour chaque processus. Indiquez sur un axe vertical, les différentes commandes et les tubes ouvert.

2 Problème de communication

Le programme suivant est incorrect. Pourquoi ?

```
int main()
2 {
  pid_t idpr;
4 int descP[2] , nbOctLus, cptRendu;
  char bufLect [80];
6
  if (pipe(descP) == -1) {
8   perror ("echec creation pipe");
    exit(EXIT_FAILURE);
10 }
  idpr=fork();
12 if (idpr == -1) {
    perror ("echec creation processus");
14  exit(EXIT_FAILURE);
  }
16 if (idpr == 0) {    /* le fils */
    char mots[] = "bonjour Pere";
18  close (descP [0]);
    write (descP[1], mots, strlen (mots));
20  exit (0);
  }
22
  close (descP [0]); close (descP [1]);
24
  nbOctLus = read (descP [0], bufLect, 80);
26 if (nbOctLus >= 0) {
    write (1, "Fils a ecrit :",14);
28  write (1, bufLect, nbOctLus);
  }
30 wait (&cptRendu);
  if (WIFEXITED (cptRendu))
32  write (1, "Normal \n", 8);
  else
34  write (1, "Anormal \n", 9);
  exit (0);
```

```
36 }
```

3 Communication bilatérale

Pour mettre en oeuvre une communication bilatérale entre un processus père et un processus fils, il suffit d'étendre ce mécanisme à deux tubes, l'un étant utilisé pour écrire du père vers le fils, et l'autre du fils vers le père. Il est également possible d'initialiser depuis un processus père des communications entre plusieurs processus fils.

Soit un programme dans lequel il s'agit de faire communiquer entre eux, de façon symétrique, deux processus enfants A et B. Pour cela on enchaîne dans le processus père :

1. La création de deux tubes : `pipe(tube_a_b); pipe(tube_b_a);`
2. La création du processus A : `pid_a = fork ();`
3. La création du processus B : `pid_b = fork ();`
4. Puis
   — Dans le processus père : fermeture de tous les descripteurs de tube (le père ne communique pas avec ses fils) et attente de la terminaison des deux fils.
   — Dans chaque fils : fermeture des descripteurs inutiles.

Dessinez le schéma correspondant au programme décrit ci-dessus, sur lequel vous ferez apparaître sur un axe vertical les différents appels aux fonctions pipe et fork, ainsi que les fils créés et les tubes qui leur permettent de communiquer.

4 Duplication de descripteurs

Il est possible de rediriger l'entrée et la sortie standard d'un processus, vers un fichier ou vers un tube. Dans cet exercice, la configuration recherchée est la suivante :

1. Les entrées/sorties standard du processus père demeurent inchangées, afin qu'il puisse continuer à interagir avec le terminal.
2. La sortie standard du processus fils est ouverte en écriture sur un tube T.
3. Le processus père possède un descripteur ouvert en lecture sur le même tube T.
4. La sortie standard erreur du processus fils n'est pas modifiée.

Dessiner un schéma qui visualise cette configuration, où l'on voit les tables de descripteur des processus père et fils, l'écran et le tube.

Pour implémenter cette configuration, il faut utiliser un mécanisme de duplication de descripteur de fichier (fonction `dup2()`). Modifiez le programme fourni dans l'exercice 1 afin de faire correspondre l'entrée standard du processus fils avec la sortie standard du processus père en utilisant la commande `dup2()`.

Un fois ces modifications faîtes dans l'exercice 1, que se passe-t'il si on remplace la commande `write()` du père par la commande `ps -l` (via un appel à `exec`) et la commande `read()` du fils par la commande `wc -l` ? Dessiner un schéma explicatif.

Deuxième partie

Vous testerez vos programmes avec des jeux d'essai, c'est-à-dire différentes valeurs pour vérifier son bon fonctionnement. Vous indiquerez en commentaires dans le code source ou dans un fichier séparé, le jeu d'essai que vous avez testé et les résultats obtenus.

Les exercices possédant une **(*)** sont facultatifs. À la fin de la séance, vous ferez une archive compressée, avec `tar` du dossier du TP que vous enverrez par mail à votre enseignant.

5 Communication père-fils

Utilisez un tube pour réaliser une communication entre un processus père et son fils. Le père lit des caractères au clavier (`getchar ()`) et les envoie à son fils via le tube. Le fils affiche à l'écran uniquement les caractères alphabétiques.

6 Communication bilatérale

Écrire un programme, semblable à celui de l'exercice 1, dans lequel un père crée deux fils, ouvre deux tubes et attend la fin de ses fils. Le fils A ferme les extrémités des tubes non utilisées, lit des caractères au clavier et les envoie au fils B via le tube, puis il attend que le fils B lui renvoie via le deuxième tube le nombre de caractères alphabétiques qu'il (B) a reçus.

7 Duplication de descripteurs

Il est possible de rediriger l'entrée et la sortie standard d'un processus, vers un fichier ou vers un tube. Dans cet exercice, la configuration recherchée est la suivante :

1. Les entrées/sorties standard du processus père demeurent inchangées, afin qu'il puisse continuer à interagir avec le terminal.
2. La sortie standard du processus fils est ouverte en écriture sur un tube T.
3. Le processus père possède un descripteur ouvert en lecture sur le même tube T.
4. La sortie standard erreur du processus fils n'est pas modifiée.

Dessiner un schéma qui visualise cette configuration, où l'on voit les tables de descripteur des processus père et fils, l'écran et le tube.

Pour implémenter cette configuration, il faut utiliser un mécanisme de duplication de descripteur de fichier (fonction `dup2()`). Modifiez le programme fourni dans l'exercice 1 afin de faire correspondre l'entrée standard du processus fils avec la sortie standard du processus père en utilisant la commande `dup2()`.

8 Redirection de l'entrée et de la sortie standard

Écrivez un programme qui permette d'implémenter la configuration décrite dans l'exercice 4 du TD. Vous devrez donc créer un tube dans le père, créer un fils, rediriger l'entrée standard du fils vers la sortie du tube et rediriger la sortie standard du père vers l'entrée du tube. Le père execute ensuite la commande `ps -l` et le fils la commande `wc -l` (en utilisant `exec` tous les deux). N'oubliez pas de fermer les descripteurs non utilisés.

9 Communication améliorée (*)

Dans l'exercice précedent, un blocage peut survenir si le second `fork()` échoue (celui servant à créer le fils B). En effet, le fils A va attendre l'envoie de B, qui ne surviendra jamais puisque B n'aura pas été crée. Vous ajouterez donc une sécurité pour prévenir cette situation dans le père. Dans le cas où cette seconde création échoue, on provoque la terminaison du processus A au moyen du signal `SIGKILL` envoyé avec la primitive `kill()`. L'utilisation de cette primitive est indiqué dans sa page `man`.