Programmation Système TD N°3

Département Informatique
Licence S6

Notions Abordées :

- **Les tubes** : *pipe(), read(), write(), close(), fork(), dup()/dup2().*

Exercice 1

Ecrire le programme C sous Unix qui permet à un processus père de créer un processus Fils. Le fils récupère une chaine de caractère l'écrit dans un tube. Le père lit du tube caractère par caractère et l'affiche en majuscule.

Appels Système : fork(), pipe(), read(), write(), close();

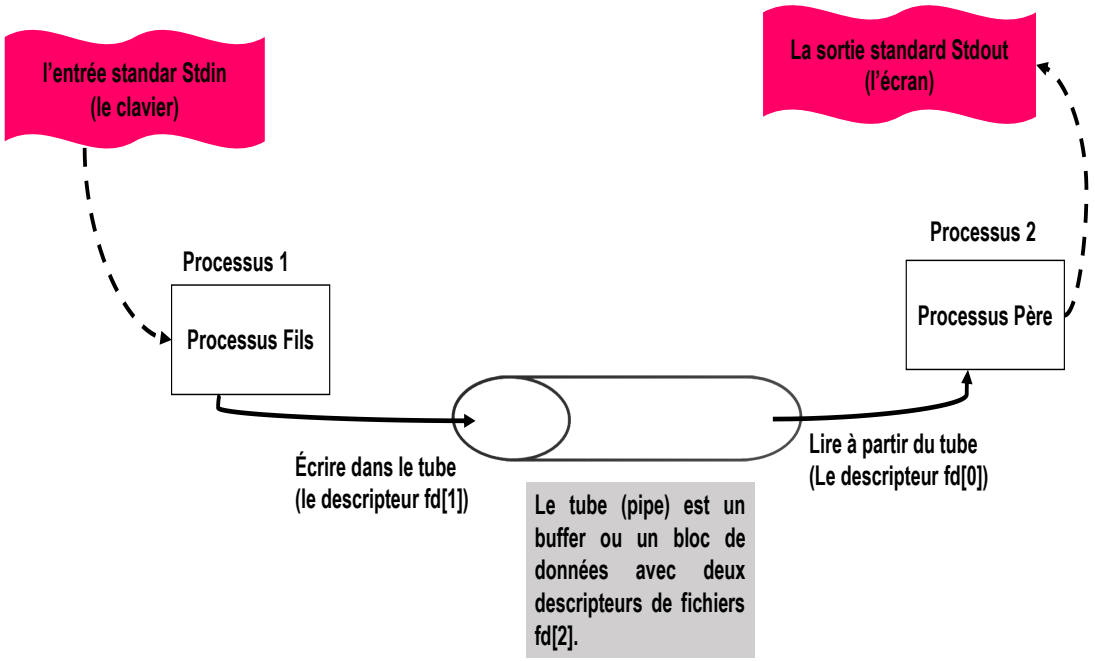
Correction :
Rappel sur les tubes anonymes :

- Les tubes anonymes sont un moyen de communication unidirectionnels entre processus
- Les processus communicant avec un tube anonyme (pipe()) **doivent avoir un lien de parenté**.
- Deux descripteurs pour manipuler un tube
  - Un descripteur pour la lecture
  - Un autre pour l'écriture
- Un tube est une file de type FIFO (premier écrit, premier lu)
- Ce qui est lu qui est lu quitte définitivement le tube et ne peut être relu
- De même ce qui est écrit est définitivement écrit
- Ils sont implémentés comme des fichiers mais n'ayant aucun nom physique (il n'existe que pendant l'exécution du programme).
- Un tube est une forme de redirection utilisé pour envoyer la sortie d'un programme à un autre programme.

Correction Exercice 1 :

Un programme C qui réalise ce qui suit:

- Création d'un processus fils.
- Le fils récupère une chaîne de caractère (soit à partir d'un argument du main, soit entrée par un utilisateur), et l'écrit dans le tube
- Le père lit du tube caractère par caractère et l'affiche en majuscule.



La création d'un tube se fait à l'aide de l'appel système pipe.

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

Par définition
fd[0] est le descripteur (prédéfini) par lequel on peut lire dans le tube.
fd[1] est le descripteur par lequel on peut écrire dans le tube.

NB : Si le tube est créé par un processus avant la création de ses fils, alors ses fils héritent alors du descripteur du père. Par conséquent , ***il faut utiliser pipe avant d'utiliser fork***.

La pipe (tube) est alors automatiquement partagée entre le père et le fils.

Après sa création, un tube est directement utilisable. Par contre, sa fermeture se fait en appelant la fonction close().

Pour être certain de qui va écrire et qui va lire dans la pipe, il faut fermer les deux extrémités non-utilisées. En effet, en écrivant dans la pipe sans fermer l'extrémité inutilisée, on ne sait pas lequel des deux processus va recevoir l'information. Cela, peut donner des résultats inattendus.

```
if(pid == 0){ //Dans le processus fils
    close(fd[0]); // On ferme le descripteur de lecture pour le fils

     Instructions pour écrire la chaine dans le tube
     ……..
    close(fd[1]); // Après que le fils aie terminé l'écriture dans le tube, on ferme le descripteur de l'écriture fd[1]

}
else
{   //Le Père
    close(fd[1]); // On ferme le descripteur d'écriture pour le père

     Instructions pour lire la chaine caractère par caractère à partir du tube
     ……
    close(fd[0]); // Après avoir terminé la lecture à partir du tube, on ferme le descripteur de lecture fd[0]

}
```

Dans le cas de cet exercice, en fermant le descripteur de lecture fd[0] pour le processus fils et le descripteur d'écriture fd[1] pour le père, on peut être certain que le fils écrit dans la pipe (fd[1]), et que le père va recevoir l'information en lecture (fd[0]).

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

#define MAX 25

int main(){

 char buffer[MAX];
 char c;
 int nbc;
 int pid, fd[2];


//Création d'un tube
/**** pipe(fd) retourne une valeur négative si elle échoue *******/
/**** pipe(fd) retourne une valeur 0  si elle réussit *******/

if(pipe(fd) < 0)
      exit(1);

 pid = fork();

if (pid == -1){
   perror("fork");
   exit(1);
}

   if(pid == 0){
     close(fd[0]); // On ferme le descripteur de lecture pour le fils
 printf(" Entrez une chaine de caractère : ");
 scanf("%s", buffer);
      write(fd[1], buffer, strlen(buffer) * sizeof(char));

 close(fd[1]); // Après avoir terminé l'écriture dans le tube, on ferme le descripteur d'écriture
  }
 else
 {
    //Le père
   //wait(NULL);
   close(fd[1]); // On ferme le descripteur d'écriture pour le père

     while( read(fd[0], &c , 1) != 0){
       printf("%c", toupper(c));
     }

    printf("\n");
   close(fd[0]); // Après avoir terminé la lecture à partir du tube, on ferme le descripteur de lecture

 }
return 0;
}
```

ssize_t write(fd[1], buffer, strlen(buffer)*sizeof(char)) :
écrit jusqu'à la longeur: (strlen(buffer) * sizeof(char)) (nombre de caractère accessible) dans le fichier associé au descripteur *fd[1]* depuis le tampon pointé par *buffer*.

ssize_t read(fd[0], c, 1 *sizeof(char)) :
lit un caractère à la fois depuis le descripteur de fichier fd[0] dans le buffer pointé par *buffer*.

Remarque : On utilise la fonction sizeof pour assurer la portabilité.

Exercice 2 : Synchronisation Père/Fils

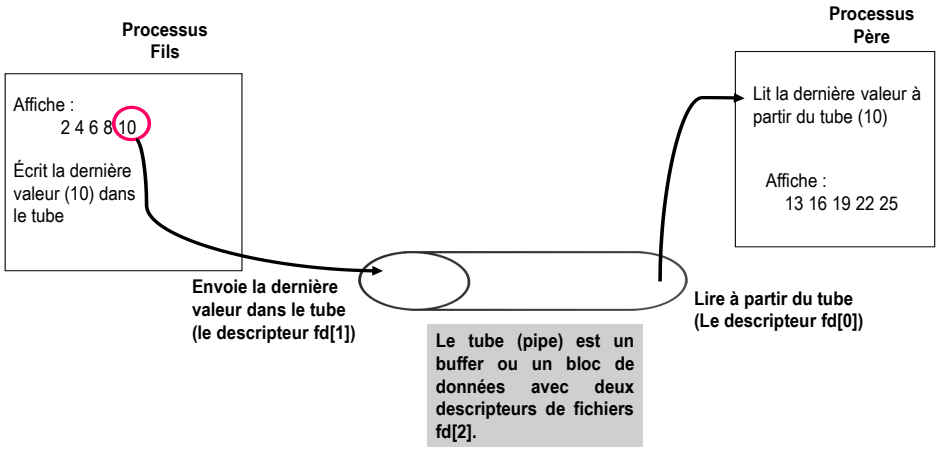
Ecrire le programme C sous Unix qui permet à un processus père de créer un processus Fils. Le père et le Fils doivent s'exécuter en parallèle et permettent l'affichage suivant :
Fils: 2 4 6 8 10
Père: 13 16 19 22 25

Appels Systèmes : *fork(), pipe()*

Indication :

Le Fils doit informer le père de la dernière valeur affichée.
On utilise un tube pour la communication Père/Fils.

Correction de l'Exercice 2 :



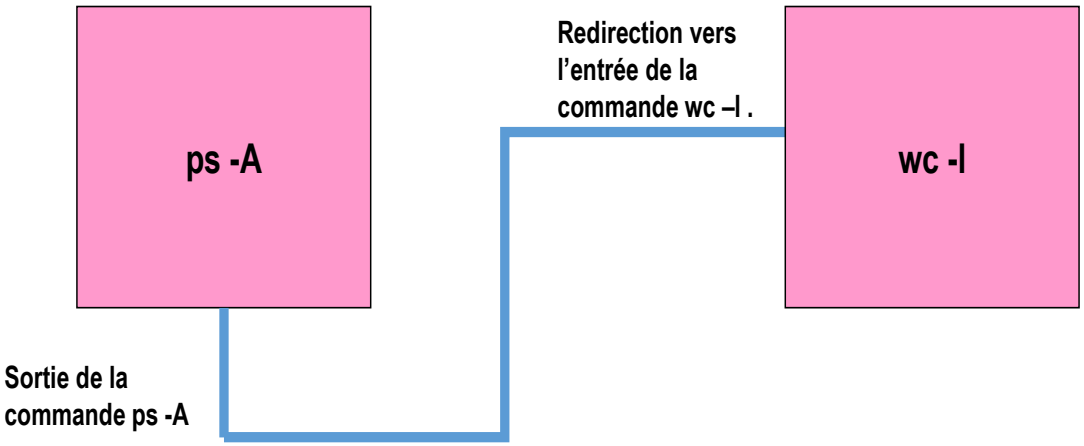

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<stdlib.h>
#include<wait.h>
#include<ctype.h>

#define MAX 25

int main(){

    int p[2];
    int pid, i = 1;
    int n1;

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

pid = fork();

if(pid == 0){

     do{
           n1 = i * 2;
       printf(" %d ", n1);
       i ++;
     }while (n1 % 5 !=0);
   printf("\n");

 //printf("  ----- %d ",n);

 close(p[0]);
 write(p[1], &n1, 1 * sizeof(int));
 close(p[1]);

} else{
  wait(NULL);
 close(p[1]);

 read(p[0], &n1, 1 * sizeof(int));

        do{
         n1 = n1 + 3;
         printf(" %d ", n1);
        }while(n1 % 5 !=0);

  printf("\n");
close(p[0]);

 }
}
```

Exercice 3 : Redirection des Entrée/Sortie

Ecrire le programme C sous Unix qui permet d'afficher le nombre de processus en cours d'exécution.

Appels Systèmes : *fork(), dup(), close()*

Correction de l'Exercice 3 :



Cela revient à exécuter la commande sous Unix ps -A | wc -l Deux processus Fils et père : le père exécute la commande : ps -A et redirige le résultat dans un tube. Le fils exécute la commande wc -l avec comme entrée le contenu du tube(redirection de l'entrée standard).

On peut utiliser l'un de ces deux appels suivants :

- int dup (int descripteur);
- int dup2 (int descripteur, int nouveau);

On utilise également l'appel *execlp()* qui permet de remplacer le processus en cours par l'exécution d'une commande.

Le code :

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

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

   int pid, fd[2];
   /* créer une pipe*/
   if (pipe(fd) == -1)
     {
      perror("pipe");
      exit(1);
     }

   /* création d'un processus*/
   pid = fork();

   if ( pid == -1)
     {
      perror("fork");
      exit(1);
     }

   if (pid == 0)
     {
      /* le fils */
      close(fd[1]); /* ferme le descripteur d'écriture */
      dup2(fd[0], 0); /* connecte le descripteur de lecture avec le stdin */
      close(fd[0]); /* ferme le descripteur de lecture*/

/* Exécute le processus (la commande wc -l) */
      execlp("wc", "wc", "-l",(char *) NULL);
     }
   else
     {
      /* Le père */
      close(fd[0]); /* ferme le descripteur de lecture */
      dup2(fd[1], 1); /* connecte le descripteur d'écriture avec le stdout */
     close(fd[1]); /* ferme le descripteur d'écriture */
     /* Exécute le processus (la commande ps -A) */
     execlp("ps", "ps", "-A", (char *)NULL);
     }
   return 0;
  }
```