USTHB
Departement Informatique
Master 1 SII

TP 4 Systeme d'exploitation:
IPC suite – Les tubes, les FIFO et les sockets

1. **Partie I: Les pipes**

Les pipes, ou tubes, sont des canaux de communication entre processus manipulés à l'aide de descripteurs de fichiers. Deux descripteurs de fichiers sont assicies a un tube, un pour la lecture et un pour l'écriture. Le tube en lui meme est un chemin unidirectionel, le processus qui ecris ne peut pas lire le pipe aussi, pour realiser un vrai dialogue entre deux processus il faut deux pipes. Les tubes sont utilisable uniquement entre processus lies (pere-fils ou fils d'un meme processus).

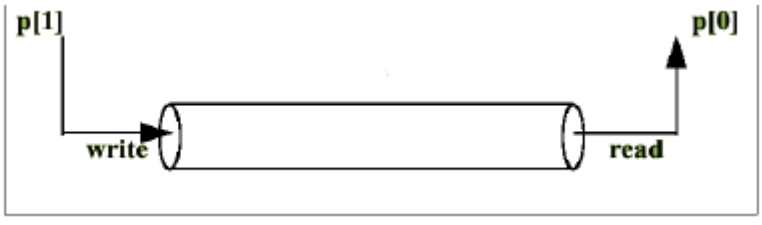
Quatre principales fonctions correspondent à la gestion des pipes:

```
#include <unistd.h>
int pipe (int descripteur[2]);
int write(int descripteur, char * buffer, int longueur);
int read (int descripteur, char * buffer, int longueur);
int close(int descripteur);
```

1. Pipe : pour la création du pipe
    a. descripteur[0]: désigne la sortie (Lecture) du tube ;
    b. descripteur[1] : désigne l'entrée (Écriture) du tube.
2. Write : utilisée pour écrire dans un pipe. Elle renvoie le nombre d'octets effectivement écrits.
3. Read : lire à partir d'un pipe.
4. Close : Fermeture d'un pipe.

Les fonctions read et write sur l'extrémité d'un pipe sont **bloquantes** jusqu'à ce que l'autre extrémité soit fermée à l'aide de close.

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



Une fin de fichier « caractère end of file : EOF » est envoyée dans un tube lorsque tous les processus ayant accès en écriture à ce tube (descripteur fd[1]) ont fermé ce descripteur (fichier).

La fonction read () retourne la valeur 0 à la lecture du caractère « fin de fichier ».

2. **Partie II: Les tubes nommes ou les FIFOs**

Les tubes nommes sont des extensions des tubes ordinaires permettant a des processus

indépendants de communiquer via un chemin crée dans l'arborescence linux du système.
La fonction **mkfifo(const char *path, mode_t mode)** permet de créer un nouveau tube dont le nom est spécifié dans *path* et les droits d'accès sont spécifiés dans *mode*. Un nouveau fichier de type pipe est crée dans le répertoire /var.

Une fois le tube crée dans l'arborescence du système, il faudra l'ouvrir pour pouvoir écrire et lire dedans comme si c'était un fichier ordinaire avec les appels système **open**, **write** et **read, close**. Il faudra toujours ouvrir deux extrémités du fichier une en mode lecture et une en mode écriture (dans les deux processus communicants). Et comme pour les tubes anonymes, un processus ne peut pas lire et ecrire dans une même FIFO. Pour faire de la communication bidirectionelle il faut utiliser deux FIFO.

Les Exercices:

Exercice 1: Les pipes
Compiler et executer le programme suivant

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(void)
{  int p[2];
   pid_t pid;
   char  buf[50];

   if (pipe(p))
   {  perror("pipe"); exit(1);
   }
   if ((pid = fork()) == -1)
   {  perror("fork"); exit(1);
   }
   if (pid>0)
   { close(p[0]);//le pere ferme descripteur lecture

      write(p[1],"Bonjour fils!\n", strlen("Bonjour fils!\n"));
      printf("Processus pere- message ecris.\n");
      fflush(NULL);
      close(p[1]);
      wait(NULL); //attendre fin fils

   }
   else //le fils
   {  close(p[1]);//ferme descripteur ecriture
      while(read(p[0],buf, 20)>0){
      printf("Processus fils-message lu est:%s\n",buf);
      }
      printf("Processus fils-FIN \n");

      close(p[0]);
   }
   return 0;
}
```

- Que se passera-t-il si le processus pere ne ferme pas le descripteur d'ecriture apres avoir fini d'ecrire?
- Pourquoi il faut toujours fermer les descripteurs qu'un processus n'utilise pas?

Exercice 2 – Les pipes :

Ecrire un programme qui donnera le même résultat que la commande shell en utilisant les pipes et 3 processus fils (un fils pour chaque partie de la commande puis passer les résultats dans un pipe au fils suivant pour faire la suite).

ps aux | grep root | wc -l
Utiliser les fonctions fork, pipe, execlp et dup (manuel!).

Exercice 3 – Les FIFOs:

Compiler les deux codes suivants de deux programmes ecrivain.c et lecteur.c, exécuter chacun des programmes dans un terminal différent.
-Exécuter d'abord écrivain et attendre quelque temps avant de lancer le programme lecteur. Le programme écrivain termine t-il son exécution ? A quel niveau il se bloque, pourquoi ?
-Vérifier la bonne création de la FIFO dans le chemin /var du système.
-Quel est le principe de fonctionnement des deux programmes (utilisez le manuel pour avoir plus de détails sur chacune des fonctions).

Programme ecrivain.c

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define TAILLE_MESSAGE      256

int main(void)
{
        int entreeTube;
        char nomTube[] = "essai.fifo";//le nom du FIFO dans le systeme

        char chaineAEcrire[TAILLE_MESSAGE] = "Bonjour";

        if(mkfifo(nomTube, 0644) != 0)
        {
                printf("Impossible de créer le tube nommé.\n");
                exit(EXIT_FAILURE);
        }

        if((entreeTube = open(nomTube, O_WRONLY)) == -1)
        {
                printf("Impossible d'ouvrir l'entrée du tube nommé.\n");
                exit(EXIT_FAILURE);
        }
       printf("Ecrivain: Fifo ouvert en mode ecriture.\n");
        write(entreeTube, chaineAEcrire, TAILLE_MESSAGE);
        printf("Ecrivain: message ecrit dans le FIFO.\n");
        return EXIT_SUCCESS;
}
```

Programme lecteur.c

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define TAILLE_MESSAGE      256

int main(void)
{
```

```
        int sortieTube;
        char nomTube[] = "essai.fifo";

        char chaineALire[TAILLE_MESSAGE];

        if((sortieTube = open ("essai.fifo", O_RDONLY)) == -1)
        {
                printf("Impossible d'ouvrir la sortie du tube nommé.\n");
                exit(EXIT_FAILURE);
        }
        printf("Le tube nommé essaie.fifo est ouvert en mode lecture.\n");

        read(sortieTube, chaineALire, TAILLE_MESSAGE);
        printf("%s", chaineALire);

        return EXIT_SUCCESS;
}
```

Exercice 4 – Les FIFOs + semaphores+threads:

Implémenter une application client-serveur basée sur les FIFO permettant (cote serveur) de réaliser un test de primalité sur un chiffre envoye par un client et de lui retourner le résultat (premier ou pas premier).

- Le serveur maintient une seule FIFO connue de tout les clients pour recevoir toutes les commandes des clients (exp. FIFO-SERVER)
- Une commande d'un client contient le couple suivant: chiffre, FIFO-CLIENT-%d qui est le nom standard utilise par chaque client pour recevoir les réponses du serveur. Tel que %d est le PID du client. La commande pourrait être écrite sous forme d'une structure.
- Le serveur crée un thread pour chaque nouvelle demande d'un client et lui passe le chiffre a vérifier et la FIFO du client ou il doit envoyer la réponse.
- Le serveur ne peux traiter que n requêtes clients en même temps, donc il faut utiliser les sémaphores pour bloquer les clients superflus.