TD Programmation système "Pipe"

1. **Héritage d'un tube ordinaire**
   Ecrire un programme qui crée un tube ordinaire puis un fils.
   Successivement, le père et le fils s'échangent des messages (Salut PAPA et Salut Fils)

2. **Tube sans écrivains**
   Ecrire un programme qui crée un tube ordinaire p et lit dans p[0]. Que se passe-t-il si le programme ne ferme pas p[1] avant de lire dans p[0]? Même question si le programme ferme p[1] avant de lire dans p[0]?

3. **Tube sans lecteurs**
   Ecrire un programme qui crée un pipe ordinaire p et écrit dans p[1]. Que se passe-t-il si le programme ne ferme pas p[0] avant d'écrire dans p[1]? Même question si le programme ferme p[0] avant d'écrire dans p[1]?

4. Tubes nommés
   Ecrire deux programmes qui communiquent par tube nommé:
   L'ordre de lancement des deux programmes est indifférent. Expliquer.
   1) Le premier effectue les actions suivantes:
      - crée un tube nommé myTube en lecture et en écriture pour le user et le groupe
      - demande un descripteur en écriture
      - écrit un message
      - libère le descripteur
      - demande un descripteur en lecture
      - lit un message et l'affiche
      - libère le descripteur
      - supprime la référence myTube

   2) Le deuxième effectue les actions suivantes:
      - demande un descripteur en lecture sur le tube nommé myTube
      - lit un message et l'affiche
      - libère le descripteur
      - demande un descripteur en écriture sur le tube nommé myTube
      - écrit un message
      - libère le descripteur .

1. Héritage d'un tube ordinaire

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include <errno.h>
4. void erreur(const char * message);
5. void main()  {
6.   int p[2];
7.   char message[15];
8.   pid_t pid;
9.   int retour,statut;
10.
11.  retour = pipe(p);
12.  if (retour==-1) erreur("Erreur pipe");
13.  pid = fork();
14.  switch(pid)    {
15.         case -1:
16.                erreur("Erreur fork:");
17.         case 0:
18.                retour = write(p[1],"Salut Papa",11);
19.                if (retour == -1) perror ("Erreur write p[1] du fils:");
20.                sleep(4);
21.                kill(getppid(), SIGCONT);
22.                kill(getpid(), SIGSTOP);
23.                retour = read(p[0],message,11);
24.                if (retour == -1) perror ("Erreur read p[0] du fils:");
25.                else printf("Message lu par le fils %s\n", message);
26.                break;
27.         default :
28.                kill(getpid(), SIGSTOP);
29.                retour = read(p[0],message,11);
30.                if (retour == -1) perror ("Erreur read p[0] du pere:");
31.                else printf("Message lu par le fils %s\n", message);
32.                retour=write(p[1],"Salut Fils",11);
33.                if (retour == -1) perror ("Erreur write p[1] du pere:");
34.                sleep(3);
35.                kill(pid, SIGCONT);
36.                wait(&statut);
37.
38.         }
39.         printf("Libere les descripteurs\n");
40.         close (p[1]);
41.         close (p[0]);
42.  }
43.  void erreur(const char * message) {
44.         perror(message);
45.         exit(-1);
46.  }
```

2. Tube sans écrivains

```
1. #include<stdio.h>
2. #include<unistd.h>
3. #include<signal.h>
4. #include<errno.h>
5.
6. void main(){
7.          char ch;
8.          int nb_lu, p[2], retour;
9.          retour=pipe(p);
10.         if (retour==-1) perror("erreur pipe:");
11.         //close(p[1]); /*ligne une fois en commentaire une fois non */
12.         nb_lu=read(p[0], &ch,1);
13.         if (nb_lu==-1) perror("erreur write:");
14.         else printf("Retour du read %d\n", nb_lu);
15.         close(p[0]);
16.  }
```

3. Tube sans lecteurs

```
1. #include<unistd.h>
2. #include<stdio.h>
3. #include<errno.h>
4. #include<signal.h>
5. void handler_SigPipe(int sig);
6.
7. void main() {
8.          struct sigaction action;
9.          int nb_ecrit, p[2], retour;
10.         action.sa_handler=handler_SigPipe;
11.         retour=sigaction(SIGPIPE, &action, NULL);
12.         if (retour==-1) perror("erreur sigaction:");
13.         retour=pipe(p);
14.         if (retour==-1) perror("erreur pipe:");
15.         // close(p[0]); /*ligne une fois en commentaire une fois non*/
16.         nb_ecrit=write(p[1], "A",1);
17.         if (nb_ecrit==-1) perror("erreur write:");
18.         else printf("Retour du write %d\n", nb_ecrit);
19.         close(p[1]);
20.  }
21.  void handler_SigPipe(int sig)  {
22.         printf("Signal intercepte %d",sig);
23.  }
```

4. Tubes nommés

Programme1.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

#define TAILLE 20

void Erreur(const char *);

void main() {
   int fd, retour_mkfifo;
   char Message[TAILLE+1];

   printf("\t COMMUNICATION PAR UN TUBE NOMME\n");

   if ((retour_mkfifo = mkfifo("mytube", S_IRUSR|S_IWUSR)) == -1)
   Erreur("Mkfifo");

   printf("Tube cree \n");

   if((fd = open("mytube", O_WRONLY)) == -1)
   Erreur("Open ecriture");
   strcpy(Message, "test tube nomme");
   write(fd,Message,strlen(Message)+1);
   printf("Message envoye par le createur: %s\n",Message);
   close(fd);

   if((fd = open("mytube", O_RDONLY)) == -1)
      Erreur("Open lecture");
   read(fd,Message,TAILLE+1);
   printf("Message recu par le createur: %s\n",Message);
   close(fd);
   unlink("mytube");
}

void Erreur(const char * Message)
{
  perror(Message);
  exit(-1);
}
```

programme2.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

#define TAILLE 20

void Erreur(const char *);

void main() {
  int fd_access, fd;
  char Message[TAILLE + 1];

  do
    fd_access = access("mytube",R_OK);
  while(fd_access == -1);

  if((fd = open("mytube",O_RDONLY))== -1)
  Erreur("Open lecture:");

  read(fd,Message,TAILLE+1);
  printf("Message recu par le consultant: %s\n",Message);
  close(fd);

  if ((fd = open("mytube",O_WRONLY)) == -1)
  Erreur("open ecriture");

  strcpy(Message, "retour test");
  write(fd,Message,strlen(Message)+1);
  printf("Message envoye par le consultant : %s\n",Message);
  close(fd);
}

void Erreur (const char * Message){
perror(Message);
exit(-1);
}
```