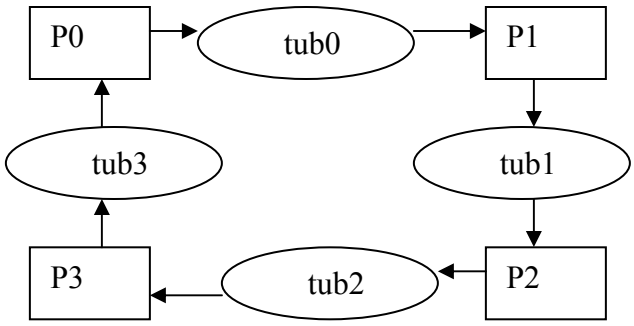
Partie 2 : Communication interprocessus

Exercice 1 :

Considérez N processus qui communiquent au moyen de tubes de communication non nommés (unnamed pipe). Chaque processus partage deux tubes (un avec le processus de droite et un autre avec le processus de gauche). Par exemple pour N=4, les processus communiquent selon le schéma suivant :



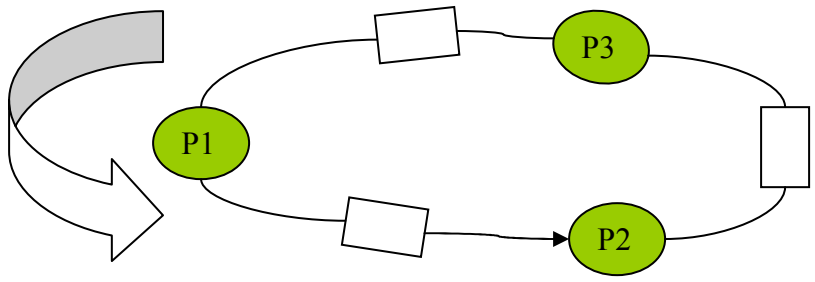
1) Complétez le code ci-après de manière à implémenter cette architecture de communication des N processus créés. L'entrée standard et la sortie standard de chaque processus Pi sont redirigées vers les tubes appropriés. Par exemple, pour le processus P0, l'entrée et la sortie standards deviennent respectivement les tubes tub3 et tub0.

```
#include ″EntetesNecessaires.h″
#define N 4
void proc( int ) ;
int main ( )
{          int i ;
          for( i=0 ; i <N; i++)
                    if ( fork() ==0)
                   {          proc(i);   exit(0);          }
          exit(0) ;
 }
#include  ″codedeproc″
```

Attention : Vous ne devez pas écrire le code de la fonction proc.

Exercice 2 :

On veut établir, en utilisant les tubes anonymes (pipes), une communication de type anneau unidirectionnel entre trois processus fils. Pour ce faire, la sortie standard de l'un doit être redirigée vers l'entrée standard d'un autre, selon le schéma suivant :



Complétez le programme suivant en y ajoutant le code permettant de réaliser les redirections nécessaires à la création d'un tel anneau.

```
/*0*/
int main ()
{
        /*1*/
        if (fork()) // création du premier processus
        {
                if(fork())
                {
                        /*2*/
                        if(fork())
                        { /*3*/
                                while (wait(NULL)>0);
                                /*4*/
                        } else
                        { // processus P3
                                /*5*/
                                execlp("program3", "program3",NULL);
                                /*6*/
```

```
                }
        } else
        { // processus P2
                /*7*/
                execlp("program2", "program2",NULL);
                /*8*/
        }
} else
{ //processus P1
        /*9*/
        execlp("program1","program1", NULL);
        /*10*/
}
/*11*/
}
```

Exercice 3 :

Écrivez le « main » d'un processus qui permet de simuler un pipe ('|'). Ce code doit utiliser un tube nommé et doit avoir exactement le même comportement qu'un pipe ('|') dans un shell.

Exemples :

bash$> my_pipe who wc –l (equivaut à bash$> who | wc –l)

bash$> my_pipe ls grep "my_pipe" (équivaut à bash$> ls | grep "my_pipe")

Exercice 4 :

Considérez le programme suivant :

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>

void sigintP()
{/*1*/}
void sigalrm()
{/*2*/}
```

```
void sigintF()
{/*3*/}
void sigchld()
{
        int status;
        wait(&status);
        exit(0);
}

int main(void)
{
        signal(SIGCHLD, sigchld);
        if (fork() == 0)
        {
                signal(SIGINT, sigintF);
                while(1)
                {
                        printf ("ici fils \n");
                        sleep(1);
                }
        }
        while(1)
        {
                signal(SIGINT, sigintP);
                printf("ici pere \n");
                sleep(1);
        }
        return 0;
}
```

Complétez le code précédent de manière à réaliser ces traitements :

1. Si l'utilisateur presse les touches Ctrl-C lorsque le programme s'exécute, les processus père et fils ne se terminent pas immédiatement, mais après un délai de 5 secondes. Lorsque l'utilisateur presse les touches Ctrl-C, le père affiche son identificateur (sans se terminer).

Indication: Ctrl-C doit déclencher un appel système alarm(5), qui envoie automatiquement le signal SIGALRM après 5 secondes.

2. Dans quel ordre les processus père et fils se terminent? Expliquez.

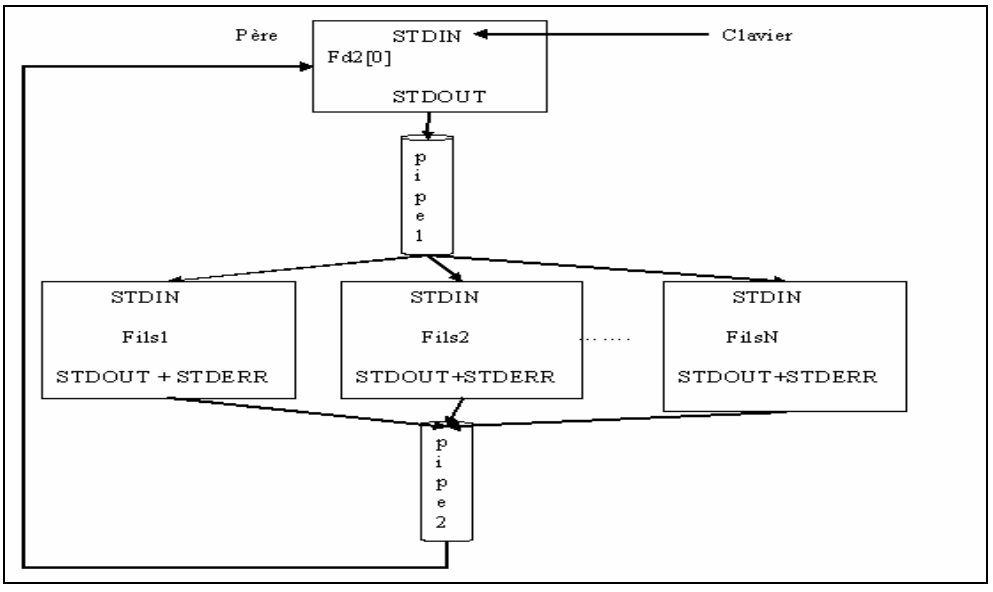
Exercice 5 :

Considérez le programme suivant :

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define N 5
void gestion(){ /* traitement */ };

/*0*/
int main( )
{       pid_t pid[N];
        int i;
        /*1*/
        for ( i=0; i<N;i++)
            if ((pid[i]=fork())==0)
            {    /* 2*/
                 execlp("traitement", "traitement", NULL);
                 exit(1);
            }
          /*3*/
        gestion(  );
         return 0;
}
```

1- On veut faire communiquer le processus père avec ses fils au moyen de deux tubes anonymes (pipes sans nom) selon le schéma suivant :



La sortie standard du père est redirigée vers le pipe1 qui devient l'entrée standard des fils. Les sorties standards et erreurs des fils sont redirigées vers le pipe2.

Complétez le code de manière à établir ces canaux de communication (supposez que le fork n'échoue jamais).

Exercice 6 :

On dispose d'une fonction F() d'une bibliothèque qui écrit une certaine quantité de données sur la sortie standard (descripteur de fichier 1). On aimerait récupérer, en utilisant un pipe anonyme, ces données pour les traiter.

```
#define Taille  1024
int main
{
        char Data[Taille];
        /*1*/
        F ( );
        /*2*/
        utiliser_resultat (Data);
}
```

1. Insérez du code en amont et en aval de F( ) afin que tous les caractères émis par F( ) sur la sortie standard soient récupérés dans Data. Vous pouvez utiliser des variables et appels système supplémentaires, mais vous ne pouvez pas utiliser de fichiers ni de processus supplémentaires.
2. Que se passe-t-il, si la taille des données émises par F dépasse celle de Data ?
3. Que se passe-t-il, si la taille des données émises par f dépasse celle du pipe ?
4. Proposez une deuxième version corrigeant ces problèmes. Pour ce faire, vous pouvez utiliser un processus supplémentaire.