TP3

Exercice 1 : Primitive *wait()*

Le programme suivant illustre le cas ou le processus père attend son fils.

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

main(){
int PID, valeur_fork, status;

valeur_fork=fork();
switch(valeur_fork){
case -1 : exit(2);
case 0:
        printf("Fils : Processus avec PID %d \n", getpid());
        sleep (10);
        printf("Fils : Processus dont le PID %d est terminé\n", getpid());
        exit(10);
default :
        printf("Père : Processus dont le PID %d\n",getpid());
        PID=wait(&status);
        printf("Père : fin du wait \n");
        printf("Père : Processus fils dont le PID = %d est terminé \n", PID);
        printf("Père : Processus père terminé \n\n");
        exit(0);
} }
```

Solution :
La primitive *wait()* permet à un processus appelant de suspendre son exécution en attente de recevoir un signal de fin de l'un de ses fils.
Lorsque le signal est reçu, la cause du décès du fils est stockée dans la variable "status".
3 cas à distinguer :
-Processus stoppé,
-Processus terminé volontairement par exit,
-Processus terminé à la suite d'un signal.
wait renvoie le PID du processus fils terminé ou -1 en cas d'erreur.

Exercice 2 : Pipes sans nom

```
#include<sys/types.h>
#include <fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define R 0
#define W 1

int main(){
int d[2], nboctets, valeur_fork;
char message[100] ;
char phrase[50] = "message envoyé au père par le fils";
pipe(d); // création d'un tube sans nom
valeur_fork=fork();
if(valeur_fork== 0) // création d'un processus fils
{

close(d[R]); // Le fils ferme le descripteur non utilisé de lecture
write(d[W], phrase, strlen(phrase)) ;
close(d[W]); // fermeture du tube en mode écriture

}
else {
close(d[W]); // Le père ferme le descripteur non utilisé d'écriture
nboctets = read(d[R],message,100); //extraction du message envoyé via le tube
printf ("Père : Lecture de %d octets : %s\n", nboctets, message) ;
close (d[R]); // fermeture du tube en mode lecture
}
return 0 ;}
```

Solution :
L'objectif c'est de pouvoir faire communiquer un processus père avec un des processus fils en utilisant les tubes sans nom.

Exercice 3 : Pipes nommées

1) Le processus écrivain (writer.c)

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

```
int main ()
{
int fd;
char message[50] = "Bonjour du processus writer \n";
printf ("processus writer dont le PID = %d\n",getpid());

fd = open("mypipe",O_WRONLY); //Ouverture du tube mypipe en mode écriture
if(fd != 1){
write(fd,message,strlen(message)) ;
}
else {
printf ("Le tube n'est pas disponible \n");
}
// Fermeture du tube
close (fd);
return 0 ;}
```

2) Le processus lecteur (reader.c) :

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

int main() {
int fd, n;
char in;
// ouverture du tube mypipe en mode lecture
printf("processus reader dont le PID = %d \n",getpid());
fd = open("mypipe", O_RDONLY); //Ouverture du tube mypipe en mode lecture

if(fd != 1) {
printf ("Reader : debut de lecture\n");

while ((n=read(fd,&in,1)) > 0){
        printf ("%c", in) ;
}
printf ("Lecture terminée !\n") ;
}
else
printf ("Le tube n'est pas disponible \n ") ;

close (fd) ;
return 0 ;}
```

Solution :

La communication entre deux processus distincts est possible grâce aux tubes nommés.