Partie 2 : Communication interprocessus

Le corrigé

Solution 1

```
#include ″EntetesNecessaires.h″
#define N 4
void proc( int ) ;
int main ( )
{          int i ;
          int fd[N][2];
          for( i=0 ; i <N; i++)  pipe(fd[i]);
          for( i=0 ; i <N; i++)
                    if ( fork() ==0)
                    {
                              dup2(fd[i][1],1);
                              dup2(fd[(N+i-1)%N][0],0);
                              for(int j=0 ; j <N; j++) {close (fd[j][0]); close(fd[j][1]);}
                         proc(i);   exit(0);
                    }
          exit(0) ;
 }
#include  ″codedeproc″
```

Solution 2

```
/*0*/
int main ()
{
          /*1*/
          int fd12[2], fd23[2], fd31[2];
          pipe(fd12); pipe(fd23); pipe(fd31);
```

```
        if (fork()) // création du premier processus
        {
                if(fork())
                {
                        /*2*/
                        if(fork())
                        { /*3*/
                                while (wait(NULL)>0);
                                /*4*/
                        } else
                        { // processus P3
                                /*5*/
                                dup2(fd23[0],0); dup2(fd31[1],1);
                                close(fd12[1]); close(fd12[0]);
                                close(fd23[1]); close(fd23[0]);
                                close(fd31[1]); close(fd31[0]);
                                execlp("program3", "program3",NULL);
                                /*6*/
                        }
                } else
                { // processus P2
                        /*7*/
                        dup2(fd12[0],0); dup2(fd23[1],1);
                        close(fd12[1]); close(fd12[0]);
                        close(fd23[1]); close(fd23[0]);
                        close(fd31[1]); close(fd31[0]);
                        execlp("program2", "program2",NULL);
                        /*8*/
                }
        } else
        { //processus P1
                /*9*/
                dup2(fd12[1],1); dup2(fd31[0],0);
                close(fd12[1]); close(fd12[0]);
                close(fd23[1]); close(fd23[0]);
                close(fd31[1]); close(fd31[0]);
                execlp("program1","program1", NULL);
                /*10*/
        }
        /*11*/
}
```

Solution 3

```
/*  Code testé sur Linux */
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/stat.h>
#include <sys/errno.h>
extern  int errno;

#define FIFOA "fifoa"

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

	if (mknod(FIFOA, S_IFIFO | 0666, 0) <  0) /* create fifo */
		perror("FIFOA open failed");

	if (!fork())
	{
		if ((writefd = open(FIFOA, O_WRONLY)) < 0)
		{
			perror("Open FIFOA write");
			exit(1);
		}

		dup2(writefd, 1);
		close(writefd);
		argv[2] = NULL;
		execvp(argv[1], &argv[1]);
		perror("Error EXECVP");
		exit(0);
	}

	if (!fork())
	{
		if ((readfd = open(FIFOA, O_RDONLY)) < 0)
		{
			perror("Open FIFOA read");
			exit(1);
		}
```

```
            dup2(readfd, 0);
            close(readfd);
            execvp(argv[2], &argv[2]);
            perror("Error EXECVP");
            exit(0);
        }

 wait(NULL);
 wait (NULL);
 unlink(FIFOA);
 exit(0);
}
```

Solution 4

1.

```
/*1*/
sigintP()
{
        printf("pid=%d\n",getpid());
        signal(SIGINT, sigintP);
}
/*2*/
sigalrm()
{
        exit(1);
}
/*3*/
sigintF()
{
        signal(SIGINT, SIG_IGN);
        signal(SIGALARM,sigalrm);
        alarm(5);
}
```

2. Si l'utilisateur presse les touches Ctrl-C, le fils se termine après 5 secondes, lorsqu'il reçoit le signal SIGALRM. Le fils se termine et le père également.

Solution 5

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

/*0*/  int fd1[2], fd2[2];
int main( )
{       pid_t pid[N];
        int i;
        /*1*/  pipe(fd1); pipe(fd2);
        for ( i=0; i<N;i++)
            if ((pid[i]=fork())==0)
            {    /* 2*/  dup2(fd1[0],0) ; dup2(fd2[1],1) ; dup2(fd2[1],2) ;
                          close(fd1[0]); close(fd1[1]); close(fd2[1]); close(fd2[0]);
                 execlp("traitement", "traitement", NULL);
                 exit(1);
            }
         /*3*/ dup2(fd1[1],1); close(fd1[0]); close(fd1[1]); close(fd2[1]);
        gestion(  );
         return 0;
}
```

Solution 6

```
1. int main()
{       char Data [Taille];
        int fd[2];
        pipe (fd);
        dup2 (fd[1], 1);
        close (fd[1]);
        F ();
        close(1) ;
        read (fd[0], Data, Taille);
        close (fd[0]);

        utiliser_résultat (Data);
}
```

2. Si la taille des données émises par F dépasse celle de Data, il y aura perte de données car il y a une seule lecture du pipe et le nombre de caractères lus ne peut pas dépasser Taille.

3. Si la taille des données redirigées est trop importante, le processus se bloquera lorsque F() tentera d'écrire dans un pipe plein, et sans jamais pouvoir être réveillé puisque les lectures sur le pipe ne pourront avoir lieu qu'à la sortie de F().

4.
```
int main ()
{       char Data[Taille];
        int fd[2];

        pipe (fd);
        if (fork () == 0)
        {       /* le fils */
                close (fd[0]);
                dup2 (fd[1], 1);
                close (fd[1]);
                F ();
                close (1);
                exit (0);
        }
        else {
                close (fd[1]);
                while (read (fd[0], Data, Taille) > 0)
                        utiliser_resultat (Data);
                close (fd[0]);
                wait (NULL);
                }
}
```