```
                    Correction du TD−TP3 − Processus, tubes, redirection,...


1) Description et rappels sur les tubes

   Insister sur l'aspect synchronisation des lectures/ecritures dans le tube.

   read bloquant  : − le tube est vide et
                    − il existe encore au moins un ecrivain dans le tube

   write bloquant : si le tube est plein (PIPE BUF)

   La fin du tube est atteinte si: − le tube est vide et
                                   − il n'existe plus d'ecrivain dans le tube
                                     => read retourne 0 pour indiquer la fin
                                        du tube.

2) Exercice 1 : 1er utilisation des tubes

   Bien decrire le programme donne en insistant sur le close(tube[1])
   dans le fils permettant, lorsque le pere fera lui aussi close(tube[1]),
   d'atteindre la fin du tube dans le fils.

   Modification du programme pour utiliser wc dans le fils
   => expliquer rapidement execlp et dup2

   void fils(int tube[2])
   {
    dup2(tube[0],STDIN_FILENO);     /* tube[0] => Entree standard         */

    close(tube[0]);  /* Il est donc maintenant possible de fermer tube[0] */

    close(tube[1]);  /* ESSENTIEL : pour la fin du tube                   */

    execlp("wc","wc","−l",NULL);

    perror("execlp");
   }

3) Exercice 2: a commencer en TD, a faire en TP

   Expliquer le module partage.h/partage.c pour la gestion de la memoire
   partagee.

   Expliquer la gestion des signaux avec sigaction (signal vs sigaction).

   Expliquer l'architecture du programme (le dessin)

   Expliquer le pourquoi de l'utilisation de fdopen



#include <stdio.h>             /* MynbOctets.c */
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "partage.h"
```



```
/* Decrire le handler de signal pour SIGUSR1 */
/* ========================================= */

void handler(int signum)
{
 if (signum!=SIGUSR1)
 {
  fprintf(stderr,"Erreur, mauvais signal recu (!=SIGUSR1)\n");
  exit(1);
 }
#if DEBUG
 else
 {
  fprintf(stderr,"Recu signal SIGUSR1 => Donnee disponible\n");
 }
#endif
}

/* Le main */
/* ======= */

int main(int argc,char **argv)
{
 pid_t  pidWC;
 pid_t  pidREAD;
 int    status;   /* Pour les waitpid                          */

 int    tube[2];
 FILE  *fIn;      /* Pour faire un fdopen : int −> FILE *      */

 struct sigaction action;

 Zone     z;
 int *ptDeb;      /* Un pointeur (int*) sur la zone debut      */

 char  *fileName=NULL;

 if (argc!=2) { fprintf(stderr,"Usage: %s fileName\n",argv[0]); return 1; }

 fileName=argv[1];

/* Gestion des signaux */
/* =================== */

 action.sa_handler=handler;      /* Preparation pour recevoir SIGUSR1        */
 sigfillset(&(action.sa_mask));  /* Signaux bloques pendant le handler: tous */
 action.sa_flags = 0;

 if (sigaction(SIGUSR1,&action, NULL) != 0)                /* SIGUSR1        */
 {
  fprintf(stderr,"Erreur: signal SIGUSR1 non capture\n");
  return 1;
 }

/* Creation de la zone de memoire partagee */
/* ======================================= */

 if (creerZonePartagee(sizeof(int),&z)==−1) return 1;

 ptDeb=(int*)z.debut;    /* *ptDeb <=> *((int*)z.debut) */
```

```
/* Creation du tube */
/* ================ */

 if (pipe(tube)==−1) { perror("pipe");
                       supprimerZonePartagee(&z);
                       return 1;
 }

/* Creation du processus qui fera le exec ...    */
/* ========================================== */

 pidWC=fork();
 switch (pidWC)
 {
  case −1 : {
             perror("fork numero 1");
             close(tube[0]);
             close(tube[1]);
             supprimerZonePartagee(&z);
             return 1;
            }
  case  0 : {
             dup2(tube[1],STDOUT_FILENO); /* Redirection sortie sur tube[1] */
             close(tube[0]);              /* Pas de lecture                 */
             close(tube[1]);              /* Ne sert maintenant a rien      */
             execlp("wc","wc","−c",fileName,NULL);
             perror("execlp");
             return 1;
            }
 }

/* Creation du processus qui fera la lecture ... */
/* ============================================= */

 pidREAD=fork();
 switch (pidREAD)
 {
  case −1 : {
             perror("fork numero 2");
             close(tube[0]);
             close(tube[1]);
             supprimerZonePartagee(&z);
             waitpid(pidWC,&status,0);        /* Le pere attend le 1er ...  */
             return 1;
            }

  case  0 : {
             int nb = −1;
             close(tube[1]);                 /* Pas d'ecriture dans le tube */
             fIn=fdopen(tube[0],"r");
             if (fIn==NULL) { perror("fdopen"); }
             else { fscanf(fIn,"%d",&nb);
                    fclose(fIn);
             }
             close(tube[0]);                 /* Pour faire propre           */
             *ptDeb=nb;     /* −1 en cas d'erreur ...  le main doit tester! */
             usleep(100);        /* Histoire de laisser le pere faire pause */
             kill(getppid(),SIGUSR1);
             return 0;
            }
 }
```



```
/* La suite du pere */
/* ================ */

 /* Fermer les descripteurs de tube inutiles au pere */

 close(tube[0]);    /*  Le processus pere ne  */
 close(tube[1]);    /* fait rien avec le tube */

 /* Attente d'un signal */

 pause();

 /* Recuperer le resultat dans la memoire partagee */

 if (*ptDeb!=−1)
 {
  printf("Il y a %d octets dans le fichier %s\n", *ptDeb, fileName);
 }

 /* Attendre le 1er enfant   */

 waitpid(pidWC,&status,0);                 /* Le pere attend le 1er  ...  */

 /* Attendre le 2eme enfant  */

 waitpid(pidREAD,&status,0);               /* Le pere attend le 2eme ...  */

 /* Supprimer la memoire partagee */

 supprimerZonePartagee(&z);

 return 0;
}
```