TP3 - Tubes; Système de Fichiers; Signaux

Vous trouvez le code pour ce TP dans le fichier http://lipn.fr/~buscaldi/TP3.tar.gz

A. Tubes

1. Le programme suivant (tube1.c) crée deux processus, un processus père qui envoie NMAX caractères au processus fils, qui affiche la chaîne de caractères reçus:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define  NMAX   10


int main ( void )  {
       int    p[2] ; //p[0] sera destiné à lire du tube, p[1] sera destiné à écrire
dans le tube
       char c ; // taille d'un char en C: 1 octet

       if ( pipe ( p ) == -1 ) { //création du tube
              fprintf ( stderr, "Erreur : tube \n" ) ;
              exit(1) ;
       }

       pid_t pid = fork();

       if (  pid  ==  -1  )  {
              fprintf ( stderr, "Erreur : fork\n");
              exit(2);
       }

       if ( pid > 0 ) {                                    /* père */
              close( p[0] ) ; //le père il ne va pas lire, donc pas besoin
              int i=0;
              c='a';
              while(i < NMAX) {
                     write(p[1], &c, 1); //destination, contenu, taille du contenu (en
octets)
                     i++;
                     fprintf(stderr, "Transmis %d chars\n",i);
              }
```

```
            close ( p[1] ) ; //on ferme le côté écriture du tube parce que on a fini
            wait ( 0 ) ; //on attend le fils
      } else  {                                       /* fils */
            char chaine[NMAX+1]; //on ajoute 1 pour le caractère terminaison '\0'
            int  i = 0 ;
            close ( p[1] ) ;
            while (i < NMAX) {
                  read ( p[0], &chaine[i], 1); //source, destination, taille du
contenu
                  i++;
            }
            //close ( p[0] ) ;
            chaine[i] = '\0';
            printf(" Chaine recue = %s \n",chaine);
            }

      return 0;
}
```

Compilez et exécutez le programme. Vérifiez le résultat pour des différents valeurs de NMAX.
- On modifie le code du père et du fils de la façon suivante:

```
if ( pid > 0 ) {    /* pere */
      close( p[0] ) ; //fermer le côté lecture
      int i=0;
      char c  ='a';
      while(1) {
            write(p[1], &c, 1);
            i++;
            fprintf(stderr, "Transmis %d chars\n",i);
      }
      close ( p[1] ) ;
      wait ( 0 ) ;
} else  {           /* fils */
      char chaine[NMAX];
      int  i = 0 ;
      close ( p[1] ) ;
      while (1) {
            read ( p[0], &chaine[i], 1);
            i++;
            if(i==NMAX-1) break; //on va s'arrêter plus tôt
      }
      chaine[i] = '\0';
      printf("Chaine recue = %s \n",chaine);
}
```

Expliquez le fonctionnement du programme. Vérifiez le nombre de caractères transmis en l'exécutant plusieurs fois. Quel est le résultat si on modifie NMAX? Pourquoi le programme s'arrête?

Le père continue à écrire dans le tube, jusqu'à quand il y a un signal SIGPIPE qui arrive au père, car il essaie d'écrire sur le tube mais il n'y a pas de processus lecteur.

2. On réutilise le code du programme écrit pour le TP2, exercice 4, modifié pour utiliser deux processus: le père qui recherche dans la première moitié du tableau, et le fils dans la seconde. Modifier le programme en introduisant un tube pour communiquer le résultat de la recherche entre le fils et le père. Le programme doit afficher la première occurrence du valeur recherché dans le tableau. Utilisez la valeur -1 pour indiquer que la valeur cherchée n'a pas été trouvée.
Suggestion: pour écrire un entier **n** dans un tube, utiliser:
`write ( p[1], &n, sizeof ( int ));`

```
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#define MAX 10


void init_tab(int n, int* a) {
	int i;
	fprintf(stderr, "tableau:\n");
	for(i=0; i< n; i++) {
		int r = rand() % MAX;
		a[i]= r;
		fprintf(stderr, "%d ", r);
	}
	fprintf(stderr, "\n");
}

/* fonction qui prend un tableau, une valeur à chercher
et les limites de la zone dans le tableau où chercher la
valeur */
int cherche(int* a, int val, int debut, int fin) {
	if(debut > fin || fin < debut) {
		fprintf(stderr, "erreur des limites de recherche");
		return -1;
	}

	int i;
	for(i=debut; i< fin; i++) {
		int tmp=a[i];
```

```
			if (val==tmp) {
				return i;
			}
		}
		return -1; //on a pas trouvé val
}

int main ( int argc, char *argv[] ) {
		int n=0;
		int val;

		if ( argc != 3 ) {
			printf( "usage: %s taille_tableau valeur_a_chercher \n", argv[0] );
		} else {
			n=atoi(argv[1]);
			val=atoi(argv[2]);
		}
		srand(time(NULL));

		int a[n];
		init_tab(n, a);

		int h=(n/2);

		int	p[2] ;
		pipe ( p );

		pid_t pid=fork();


		if(pid > 0) {
			//PERE
			int ppos, fpos;
			close(p[1]);
			ppos=cherche(a, val, 0, h);
			read(p[0], &fpos, sizeof(int));
			wait(NULL);
			close(p[0]);

			if (ppos==-1 && fpos==-1) printf("La valeur n'est pas dans le
tableau\n");
			else {
				if(ppos==-1) printf("La première occurrence de la valeur est à
%d\n", fpos);
				else if (fpos==-1) printf("La première occurrence de la valeur
est à %d\n", ppos);
				else {
```

```
                        if ((ppos <= fpos)) printf("La première occurrence de la
valeur est à %d\n", ppos);
                        else printf("La première occurrence de la valeur est à
%d\n", fpos);
                    }
            }
            exit(0);
      } else if (pid==0) {
            //FILS
            int fpos;
            close(p[0]);
            fpos=cherche(a, val, h+1, n);

            write(p[1], &fpos, sizeof(int));
            close(p[1]);
            exit(0)  ;  //Le fils se termine.
      }

}
```

3. Créez un tube nommé **fifo** en utilisant la commande **mkfifo**.
Tapez la commande **cat < fifo** dans une première fenêtre terminal.
Ouvrez une seconde fenêtre terminal et tapez la commande **cat tubes1.c > fifo**
Expliquez ce qui s'est passé. Exécutez les commandes dans l'ordre inverse et vérifiez le comportement.

4. Le programme suivante (`tubes2.c`) prend en paramètre deux entiers; le père envoie au fils les deux entiers, le fils en fait la somme et affiche le résultat:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define  NMAX   10


int main ( int argc, char** argv )  {
      int    p[2] ; //vecteur de pointeurs entrée/sortie du tube
      int v1, v2;
      if (argc !=3) {
            printf("Specifier 2 nombres à sommer");
            exit(0);
      }

      if ( pipe ( p ) == -1 ) {
            fprintf ( stderr, "Erreur : tube \n" ) ;
```

```
		exit(1) ;
	}

	pid_t pid = fork();

	if (  pid  ==  -1  )  {
		fprintf ( stderr, "Erreur : fork\n");
		exit(2);
	}

	if ( pid > 0 ) {                                        /* père */
		v1=atoi(argv[1]); //lecture des valeurs à partir des paramètres
		v2=atoi(argv[2]);

		close( p[0] ) ; //on ferme le côté lecture du tube

		write(p[1], &v1, sizeof(int)); //sizeof() nous permet de savoir le
nombre d'octets qu'il faut reserver pour le type de donnée en argument
		write(p[1], &v2, sizeof(int));
		close ( p[1] ) ;
		fprintf(stderr,"Pere: somme %d et %d?\n", v1, v2);

		wait ( 0 ) ; //il faut s'assurer que le père termine après le fils
	} else  {                                       /* fils */
		int vals[2];
		int i=0;
		while(i<2) {
			read ( p[0], &vals[i], sizeof(int));
			i++;
		}
		close ( p[0] ) ;
		int somme=vals[0]+vals[1];
		printf("Fils: somme = %d\n",somme);
		}

	return 0;
}
```

Écrire deux programmes qui communiquent avec un tube nommé (**mkfifo**). Un programme lit du tube deux entiers et en fait la somme, l'autre programme prend les entiers de la ligne de commande et les renvoie au premier programme par le tube.
Créez le tube par ligne de commande avec mkfifo.

1. On va créer le tube dans la console avec "mkfifo fifo"
2. Voici le code du premier programme (client):

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

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

int main ( int argc, char** argv )  {
       int v1, v2, res;
       int fd;

       if (argc !=3) {
       printf("Specifier 2 nombres à sommer");
       exit(0);
       }

       fd = open("fifo", O_WRONLY ); //on ouvre le tube en écriture

       v1=atoi(argv[1]);
       v2=atoi(argv[2]);

       write(fd, &v1, sizeof(int));
       write(fd, &v2, sizeof(int));

       close(fd);

       fprintf(stderr,"Somme %d et %d?\n", v1, v2);

       fd = open("fifo", O_RDONLY); //on ouvre le tube en lecture
       read (fd, &res, sizeof(int));

       fprintf(stderr,"Resultat: %d\n", res);
       close(fd);
}
```

3. Code du deuxième programme (serveur):

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

int main ( int argc, char** argv )  {
       int vals[2];
       int fd;
       while(1) {
       fd = open("fifo", O_RDONLY);
    int i=0;
        while(i<2) {
       read (fd, &vals[i], sizeof(int));
```

```
      i++;
       }
    close(fd);

    printf("%d + %d\n", vals[0], vals[1]);

    int somme=vals[0]+vals[1];

       fd = open("fifo", O_WRONLY);
      write(fd, &somme, sizeof(int));
      close(fd);
      }

      return 0;
}
```

```
N'oubliez pas de lancer prog2 avant prog1...
```

4.2 Modifiez le programme pour utiliser la primitive *mkfifo* au lieu de créer le tube par ligne de commande.

```
Il suffit de rajouter dans le serveur mkfifo("fifo", 0666); /* 0666 donne
écriture/lecture à tous */
```

B. Système de Fichiers

1. Dans un système de fichiers, un inode a la structure suivante:
10 pointeurs directs vers un bloc de données
1 pointeur en simple indirection vers un bloc de pointeurs directs
1 pointeur en double indirection vers un bloc de pointeurs en simple indirection

1.1 Si la taille de chaque bloc est de 1024 octets (1Kio) et un pointeur est representé sur 4 octets, quelle est la taille maximale d'un fichier dans ce système de fichiers?

```
10 * 1Kio = 10Kio pour les pointeurs directs

Nombre de pointeurs dans un bloc de pointeurs:
1024/4 = 256

256 * 1Kio = 256Kio pour le pointeur en simple indirection

Double indirection:
256 * 256 blocs -> 65536 blocs -> 64Ki blocs * 1Kio -> 64Mio

donc 64Mio + 266Kio
```

1.2 Même question avec la taille de chaque bloc = 8192 octets (8Kio)

```
10 * 8Kio = 80Kio pour les pointeurs directs

Nombre de pointeurs dans un bloc de pointeurs:
8192/4 = 2048

2048 * 8Kio = 16384 Kio = 16Mio pour le pointeur en simple indirection

Double indirection:
2048 * 2048 blocs -> 4194304 blocs -> 4Mi blocs * 8Kio -> 32Gio

donc ~32Gio (32Gio + 16Mio + 80Kio)
```

C. Signaux

1. Exécuter le programme suivant (exo1.c) qui affiche son PID et se mette en attente:

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

void main () {
    printf("Processus %d en attente de signaux...\n", getpid());
    while (1) {
       sleep(5);
    }
    exit(0);
}
```

Utilisez la commande **kill** pour envoyer des signaux au processus. On peut obtenir une liste des signaux disponibles avec la commande **kill -l**.

1.1 Il se passe quoi si vous envoyez le signal 19 (SIGSTOP)? Quel signal il faut envoyer parce que le processus continue? Vérifiez avec **ps** le status du processus.

```
Le processus arrête  avec un message
kill -19 3183
[1]+  Stoppé                 ./exo1
```

Pour continuer il faut envoyer SIGCONT (18)
`kill -18 3183`
Attention le processus restera actif en arrière plan, avec ps on peut voir qu'il est toujour en marche.

1.2 Envoyez le signal 1 (SIGHUP). Quel est le résultat? Utilisez la commande **nohup** pour exécuter le programme et envoyez à nouveau le signal SIGHUP. Quel est le résultat?
Le processus arrête avec le message :
`[1]+  Fin de la connexion (raccroché)     ./exo1`
Avec nohup le processus continue en ignorant la SIGHUP

1.3 Associez au signal SIGHUP l'handler vide (SIG_IGN vu en cours) pour l'ignorer. Exécutez le programme modifié. Envoyez le signal SIGHUP. Quel est le résultat? Notez la différence avec le comportement du programme non modifié au point 1.2. Envoyez le signal 15 (SIGTERM) pour terminer le processus.

il suffit d'ajouter au début du main:
signal(SIGHUP , SIG_IGN) ;
Le comportement c'est le même que en exécutant avec la commande nohup

1.4 Est-il possible d'ignorer le signal SIGTERM? Vérifiez en essayant d'ignorer le signal. Quel signal faut-il envoyer pour arrêter le processus?

Oui on peut ignorer SIGTERM, ils peuvent faire
signal(SIGTERM, SIG_IGN);
Pour arrêter le processus il faudra envoyer SIGKILL

1.5 Écrivez un handler pour gérer le signal SIGHUP de façon qu'il soit toujours ignoré par le processus mais en affichant un message pour montrer que le signal a été reçu. Modifiez le programme pour installer l'handler et exécutez-le. Vérifiez son bon fonctionnement en lui envoyant des SIGHUP.

Par exemple:

```
void myhandler() {
   printf("SIGHUP recu\n");
}

void main () {
   signal( SIGHUP , myhandler ) ;

   printf("Processus %d en attente de signaux...\n", getpid());
   while (1) {
         sleep(5);
```

```
   }
   exit(0);
}
```

1.6 Rajoutez une variable pour compter le nombre de fois que SIGHUP a été reçu. Modifiez le handler pour afficher à chaque fois le nombre de SIGHUP reçus.

modification triviale

2. On réutilise le code du programme écrit pour le TP2, exercice 4, modifié pour utiliser deux processus: le père qui recherche dans la première moitié du tableau, et le fils dans la seconde. Modifiez le programme pour faire que le fils envoie au père SIGUSR1 si la valeur cherchée est contenue dans le tableau, et SIGUSR2 si la valeur cherchée n'est pas contenue dans le tableau. Le père doit gérer SIGUSR1 et SIGUSR2 et afficher si la valeur a été trouvée ou pas, et en cas affirmatif, si elle a été trouvée dans la première moitié du tableau, dans la deuxième ou dans les deux.

Possible solution (en gras les parties liées aux signaux):

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include <sys/types.h>
#define MAX 10

int fils_trouve=0; /* variable globale pour stocker le résultat du fils */

void init_tab(int n, int* a) {
        int i;
        fprintf(stderr, "tableau:\n");
        for(i=0; i< n; i++) {
        int r = rand() % MAX;
        a[i]= r;
        fprintf(stderr, "%d ", r);
        }
        fprintf(stderr, "\n");
}

/* fonction qui prend un tableau, une valeur à chercher
et les limites de la zone dans le tableau où chercher la
valeur */
int cherche(int* a, int val, int debut, int fin) {
```

```
        if(debut > fin || fin < debut) {
        fprintf(stderr, "erreur des limites de recherche");
        return 0;
        }

        int i;
        for(i=debut; i< fin; i++) {
        int tmp=a[i];
        if (val==tmp) {
        return 1;
        }
        }
        return 0; //on a pas trouvé val
}

/* les handlers utilisés par le père */
void handler_1() {
   fils_trouve=1;
}

void handler_2() {
   fils_trouve=0;
}

int main ( int argc, char *argv[] ) {
        int n=0;
        int val;

        /* on installe les handlers */
        signal(SIGUSR1, handler_1);
        signal(SIGUSR2, handler_2);

        clock_t start, end;

        if ( argc != 3 ) {
        printf( "usage: %s taille_tableau valeur_a_chercher \n", argv[0] );
        } else {
        n=atoi(argv[1]);
        val=atoi(argv[2]);
        }
        srand(time(NULL)); //Initialisation du générateur de nombres aléatoires

        int a[n];
```

```
        init_tab(n, a);

        int trouve;

        pid_t pid=fork();

        start = clock();

        int h=n/2;

        if(pid > 0) {
        //PERE
        trouve=cherche(a, val, 0, h);
        wait(NULL);

  if (trouve) fprintf(stdout, "%d est dans la première partie du tableau\n", val);
  if (fils_trouve) fprintf(stdout, "%d est dans la deuxième partie du tableau\n", val);

  if(trouve==0 && fils_trouve==0)  fprintf(stdout, "%d n'est pas dans le tableau\n", val);
        end= clock();
        fprintf(stdout, "\nrecherche finalisée en %f millisecondes\n",
(end-start)*1000/(double)(CLOCKS_PER_SEC));
        exit(0);
        } else if (pid==0) {
         //FILS
         trouve=cherche(a, val, h+1, n);
         pid_t ppid;
         ppid=getppid(); /* on prend le pid du père pour lui envoyer les signaux */
         if(trouve) kill(ppid, SIGUSR1);
         else kill(ppid, SIGUSR2);
         exit(0)  ;  //Le fils se termine.
        }

}
```

3. Ecrire un programme qui crée un fils. Quand le fils reçoit SIGUSR1, il doit afficher le nombre de signaux SIGUSR1 effectivement reçus. Quand le fils reçoit SIGUSR2, il doit terminer avec un message. Le père rentre dans une boucle pour demander quel signal envoyer au fils, 1 pour SIGUSR1 et 2 pour SIGUSR2 (On rentre la valeur par clavier, par exemple avec *`fscanf(stdin, "%d", &val)`*). Si le choix est SIGUSR2, il attend le fils pour terminer et il sort du boucle.

Une possible solution:

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

int nsig=0;
int continuer=1;

void compte_sig() {
   nsig++;
   fprintf(stdout, "J'ai reçu %d SIGUSR1\n", nsig);
}

void terminate() {
   fprintf(stdout, "Adieu monde...\n");
   continuer=0;
}

int main()
{
   pid_t pif;

   pif = fork() ;
   if ( pif == 0 ) {          /***  FILS ***/

          signal(SIGUSR1, compte_sig);    // Avant tout l'installation du handler
          signal(SIGUSR2, terminate);    // Avant tout l'installation du handler

          printf("FILS ID : %d  du pere (PPID) : %d \n",getpid(),getppid());
          while(continuer) {
                 sleep(5);
          }
          printf("Fils termine\n",getpid(),getppid());
          exit(0);
   } else {                          /*** PERE ***/
          int choix;
          while(1) {
                 fprintf(stdout, "Choisir 1 pour SIGUSR1 ou 2 pour SIGUSR2\n");
                 fscanf(stdin, "%d", &choix);
                 if(choix==1) kill(pif, SIGUSR1);
                 if(choix==2) {
                        kill(pif, SIGUSR2);
```

```
                wait(NULL);
                break;
            }
        }
        fprintf(stdout, "Pere termine\n");
    }
```