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] ;
       char c ;

       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 */
              close( p[STDIN_FILENO] ) ;
              int i=0;
              c='a';
              while(i < NMAX) {
                     write(p[STDOUT_FILENO], &c, 1);
                     i++;
                     fprintf(stderr, "Transmis %d chars\n",i);
              }
              close ( p[STDOUT_FILENO] ) ;
              wait ( 0 ) ;
```

```
	} else  {                                       /* fils */
		char chaine[NMAX+1];
		int  i = 0 ;
		close ( p[1] ) ;
		while (i < NMAX) {
			read ( p[STDIN_FILENO], &chaine[i], 1);
			i++;
		}
		//close ( p[STDIN_FILENO] ) ;
		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[STDIN_FILENO] ) ;
	int i=0;
	char c  ='a';
	while(1) {
		write(p[STDOUT_FILENO], &c, 1);
		i++;
		fprintf(stderr, "Transmis %d chars\n",i);
	}
	close ( p[STDOUT_FILENO] ) ;
	wait ( 0 ) ;
} else  {           /* fils */
	char chaine[NMAX];
	int  i = 0 ;
	close ( p[STDOUT_FILENO] ) ;
	while (1) {
		read ( p[STDIN_FILENO], &chaine[i], 1);
		i++;
		if(i==NMAX-1) break;
	}
	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?

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 ));`

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

		close( p[STDIN_FILENO] ) ;

		write(p[STDOUT_FILENO], &v1, sizeof(int));
		write(p[STDOUT_FILENO], &v2, sizeof(int));
		close ( p[STDOUT_FILENO] ) ;
		fprintf(stderr,"Pere: somme %d et %d?\n", v1, v2);

		wait ( 0 ) ;
	} else  {                                       /* fils */
		int vals[2];
		int i=0;
		while(i<2) {
			read ( p[STDIN_FILENO], &vals[i], sizeof(int));
			i++;
		}
		close ( p[STDIN_FILENO] ) ;
		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.

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

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?

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

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.

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?

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.

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?

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.

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.

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.

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.