Mercredi 17 novembre 2004 — Partiel : systèmes d'exploitation
Durée : 2 heures — L3, parcours informatique, 2004-2005

Corrigé préliminaire : version définitive à apparaître ce fin de semaine.

Examen partiel

Le système des fichiers

Exercice 1. Le programme suivant présente une erreur.

idem.c

```
 1 : #include <stdlib.h>
 2 : #include <sys/stat.h>
 3 : #include <stdio.h>
 4 :
 5 : #define E_ARGS -2
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     struct stat info1, info2;
10 :
11 :     if (argc != 3)
12 :         exit(E_ARGS);
13 :
14 :     stat(argv[1], &info1);
15 :     stat(argv[2], &info2);
16 :
17 :     if (info1.st_dev == info2.st_dev && info1.st_ino == info2.st_ino)
18 :     {
19 :         printf("Fichiers identiques\n");
20 :         exit(EXIT_SUCCESS);
21 :     } else
22 :     {
23 :         printf("Fichiers différents\n");
24 :         exit(EXIT_FAILURE);
25 :     }
26 : }
```

En effet, pendant son exécution, il présente un comportement curieux :

```
$ idem fichier1.txt fichier1.txt
Fichiers différents
$ idem fichier2.txt fichier2.txt
Fichiers identiques
```

1. Expliquer la raison du comportement de l'exécutable `idem`.
2. Appliquer au programme les corrections nécessaires.

Exercice 2. Écrire un programme C qui affiche sur la sortie standard tous les liens symboliques contenus dans un répertoire passé en paramètre par l'utilisateur. Si l'utilisateur ne passe pas un tel répertoire, le répertoire courant est parcouru.

La communication par signaux

Exercice 3. Considérer le programme suivant :

signal.c

```
 1 : #include <signal.h>
 2 : #include <stdio.h>
 3 : #include <stdlib.h>
 4 :
 5 : static int n = 0;
 6 :
 7 : void interruption(int signum)
 8 : {
 9 :     signal(signum, interruption);
10 :     switch (signum)
11 :     {
12 :     case SIGINT:
13 :         n += 2;
14 :     case SIGTSTP:
15 :         n--;
16 :         if (n != 0)
17 :             signal(SIGINT, SIG_DFL);
18 :         else
19 :             signal(SIGINT, interruption);
20 :     }
21 :     printf("Valeur de n : %d.\n", n);
22 : }
23 :
24 : int main(void)
25 : {
26 :     signal(SIGINT, interruption);
27 :     signal(SIGTSTP, interruption);
28 :     for (;;);
29 :     exit(EXIT_SUCCESS);
30 : }
```

1. Dire ce qui est affiché à l'écran si, pendant l'exécution du programme en avant-plan, on tape à la console :
   - `CTRL-C`,
   - `CTRL-C, CTRL-C`,
   - `CTRL-Z, CTRL-C`,
   - `CTRL-C, CTRL-Z, CTRL-C`.
2. Réécrire le programme `signal.c` en utilisant l'interface de programmation POSIX pour les signaux.

Fork et Wait

Exercice 4. Considérer le programme suivant :

forkwait.c

```
 1 : #include <stdlib.h>
 2 : #include <unistd.h>
 3 : #include <stdio.h>
 4 : #include <wait.h>
 5 :
 6 : static int n = 0;
 7 :
 8 : int main(void)
 9 : {
10 :     pid_t pid, pid1, pid2;
11 :
12 :     if ((pid = fork()) == -1)
13 :         exit(EXIT_FAILURE);
14 :
15 :     if (pid != 0)
16 :     {
17 :         n--;
18 :         pid1 = pid;
19 :         pid2 = getpid();
20 :         pid = wait(NULL);
21 :     } else
22 :     {
23 :         n++;
24 :         pid1 = getppid();
25 :         pid2 = getpid();
26 :     }
27 :     printf("%d\t%d\n", pid1, pid2 + n);
28 :     exit(EXIT_SUCCESS);
29 : }
```

1. Détailler (ligne par ligne) le comportement de ce programme.
2. On suppose que le PID du père est 2000, et que le PID du fils est 2001 : dire ce qui est affiché à l'écran.

Exercice 5. Considérer l'instruction C suivante :

```
fork() && fork() * fork();
```

On suppose que tout appel à la primitive `fork` ne renvoie pas un code d'erreur.

1. Dire combien de processus cette instruction engendre (on ne compte pas le père).
2. Faire un dessin de l'arbre généalogique du père et des processus engendrés.

Exercice 6. Écrire un programme qui engendre un processus et se termine. Le processus fils, dès qu'il devient orphelin, affichera à l'écran un message permettant à l'utilisateur de se convaincre que ce processus est orphelin.

Les tubes anonymes

Exercice 7. Le but du programme suivant est d'écrire la chaîne de caractères `"abcde"` dans une tube, de la lire de cette tube, et enfin de l'afficher à l'écran :

pipe.c

```
 1 : #include <stdlib.h>
 2 : #include <unistd.h>
 3 : #include <stdio.h>
 4 :
 5 : static int tube[2];
 6 :
 7 : void ecrire(void)
 8 : {
 9 :     char *message = "abcde";
10 :
11 :     while (*message != 0)
12 :         write(tube[1], message++, 1);
13 :     exit(EXIT_SUCCESS);
14 : }
15 :
16 : void lire(void)
17 : {
18 :     int no_lu;
19 :     char tampon[100], *curr = tampon;
20 :
21 :     while ((no_lu = read(tube[0], curr, 2)) > 0
22 :            && curr < (tampon + sizeof(tampon) - 1))
23 :         curr += no_lu;
24 :     printf("%s", tampon);
25 :     exit(EXIT_SUCCESS);
26 : }
27 :
28 : int main(void)
29 : {
30 :     if (pipe(tube) == -1)
31 :         exit(EXIT_FAILURE);
32 :     switch (fork())
33 :     {
34 :     case -1:
35 :         exit(EXIT_FAILURE);
36 :     case 0:
37 :         ecrire();
38 :     default:
39 :         lire();
40 :     }
41 :     exit(EXIT_FAILURE);
42 : }
```

Dans ce programme on y trouve deux erreurs : le premier est une erreur du traitement des objets système. L'autre est plus proprement une erreur du langage C. Trouvez-les, en bien expliquant de quel type d'erreur il s'agit, et proposez des corrections au programme.

Exercice 8. Écrire un programme qui crée une tube et puis engendre un processus fils. Le fils lit des caractères de l'entrée standard et transforme les minuscules en majuscules, le résultat étant écrit dans le tube. Le père lit du tube, efface les espace répétées, et imprime ces caractères à l'écran (sortie standard).