Mercredi 5 janvier 2005
Durée : 3 heures
Documents autorisés

Examen : systèmes d'exploitation
L3, parcours informatique, 2004-2005

Corrigé préliminaire et partiel.

Examen

Les valeurs de retour des primitives ne sont pas systématiquement testées dans les programmes de l'énoncé. On supposera donc que les primitives ne renvoient jamais un code d'erreur.

Les processus

Exercice 1.

1. Combien de processus engendre l'évaluation de la commande C

```
fork() && ( fork() || fork() ) ;
```

2. Dessiner l'arbre généalogique des processus engendrées par cette ligne.

Exercice 2. Considérer le programme suivant :

forkpause.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <unistd.h>
 4 : #include <signal.h>
 5 :
 6 : void interruption(int signum)
 7 : {
 8 :   if (signum == SIGINT)
 9 :     printf("UN\n");
10 : }
11 :
12 : int main(void)
13 : {
14 :   int pid;
15 :
16 :   signal(SIGINT, &interruption);
17 :   signal(SIGALRM, &interruption);
18 :   pid = fork();
19 :   srand(pid);
20 :   if (!pid)
21 :   {
22 :     sleep(rand() % 2);
23 :     printf("DEUX\n");
24 :     kill(getppid(), SIGINT);
25 :   } else
26 :   {
27 :     alarm(5);
28 :     sleep(rand() % 2);
29 :     printf("TROIS\n");
30 :     pause();
31 :   }
32 :   exit(EXIT_SUCCESS);
33 : }
```

1. Répondre aux questions suivants, en expliquant votre réponse.
   - Que peut se passer si l'on supprime la ligne `30` ?
   - Que peut se passer si l'on supprime la ligne `27` ?
   - Que se passe-t-il si on échange l'ordre des lignes `18-19` ?
2. Donner les différents affichages pouvant être produits par ce programme.

La communication par signaux

signaux.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <unistd.h>
 4 : #include <signal.h>
 5 :
 6 : sigset_t ens_vide, ens_tstp;
 7 : struct sigaction action;
 8 :
 9 : void handler(int sig)
10 : {
11 :   switch (sig)
12 :   {
13 :   case SIGQUIT:
14 :     sigprocmask(SIG_SETMASK, &ens_vide, NULL);
15 :     sigprocmask(SIG_SETMASK, &ens_tstp, NULL);
16 :     sigaction(SIGINT, &action, NULL);
17 :     break;
18 :   case SIGINT:
19 :     printf("A+\n");
20 :     exit(0);
21 :     break;
22 :   case SIGTSTP:
23 :     printf("Bonjour\n");
24 :     break;
25 :   default:
26 :     break;
27 :   }
28 :   return;
29 : }
30 :
31 : int main(void)
32 : {
33 :   action.sa_handler = handler;
34 :   sigemptyset(&action.sa_mask);
35 :
36 :   sigemptyset(&ens_vide);
37 :   sigemptyset(&ens_tstp);
38 :   sigaddset(&ens_tstp, SIGTSTP);
39 :   sigprocmask(SIG_SETMASK, &ens_tstp, NULL);
40 :
41 :   sigaction(SIGTSTP, &action, NULL);
42 :   sigaction(SIGQUIT, &action, NULL);
43 :
44 :   for (;;);
45 :
46 :   exit(0);
47 : }
```

On rappelle que la frappe de `CTRL-C` envoie le signal `SIGINT` au processus qui se trouve en avant-plan. De même pour `CTRL-\` et `SIGQUIT` (on suppose que l'on se trouve pas sur une machine Sun) et `CTRL-Z` et `SIGTSTP`.

Exercice 3. Commenter, ligne par ligne, le programme `signaux.c`, en donnant assez d'explications.

Exercice 4. Dire ce qui est affiché à l'écran si – pendant l'exécution en avant-plan du programme ci dessus – l'on tape :

1. `CTRL-Z`, `CTRL-C`,
2. `CTRL-\`, `CTRL-C`,
3. `CTRL-Z`, `CTRL-\`, `CTRL-C`,
4. `CTRL-\`, `CTRL-Z`, `CTRL-C`,
5. `CTRL-\`, `CTRL-Z`, `CTRL-\`, `CTRL-C`,
6. `CTRL-Z`, `CTRL-Z`, `CTRL-\`, `CTRL-C`,

La communication par tubes

Considérer le programme suivant :

tubes.c

```
 1 : #include <stdlib.h>
 2 : #include <unistd.h>
 3 : #include <fcntl.h>
 4 : #include <sys/stat.h>
 5 :
 6 : char *nom_pf = "tube_pf";
 7 : char tampon[100];
 8 : int no_lu;
 9 :
10 : void pere(d_lecture)
11 : {
12 :   int d_ecriture;
13 :
14 :   while ((no_lu = read(d_lecture, tampon, sizeof(tampon))) > 0)
15 :   {
16 :     d_ecriture = open(nom_pf, O_WRONLY);
17 :     write(d_ecriture, tampon, no_lu);
18 :     close(d_ecriture);
19 :   }
20 :   exit(0);
21 : }
22 :
23 : void fils(d_ecriture)
24 : {
25 :   char *message = "abcdefghilmno\n";
26 :   int d_lecture = open(nom_pf, O_RDONLY);
27 :
28 :   do
29 :   {
30 :     if (*message == 0)
31 :       break;
32 :     write(d_ecriture, message++, sizeof(char));
33 :     no_lu = read(d_lecture, tampon, sizeof(tampon));
34 :     write(1, tampon, no_lu);
35 :   }
36 :   while (no_lu > 0);
37 :
38 :   exit(0);
39 : }
40 :
41 : int main(void)
42 : {
43 :   int fp[2];
44 :
45 :   unlink(nom_pf);
46 :   mkfifo(nom_pf, 0600);
47 :   pipe(fp);
48 :
49 :   if (fork())
50 :     pere(fp[0]);
51 :   else
52 :     fils(fp[1]);
53 :
54 :   exit(0);
55 : }
```

.

Exercice 5. Donner un aperçu sur le programme, en expliquant de façon succincte ce qui devrait faire.

Exercice 6. Dans le programme on y trouve des erreurs de traitement des tubes : une première erreur regarde le traitement des tubes nommées, une deuxième erreur regarde le traitement des tubes anonymes.
Trouver ces deux erreurs, pour chacune erreur expliquer la raison pour lequel il s'agit d'une erreur et le corriger.

Le shell et les scripts

Exercice 7. Considérer le script `pluff.sh`.

pluff.sh

```
 1 : #!/bin/bash
 2 :
 3 : if [ "$1" -eq  0 ]
 4 :     then
 5 :     echo $1
 6 : else
 7 :     ARG=‘expr $1 - 1‘
 8 :     $0 $ARG $2 | $2
 9 : fi
```

Écrire un programme en langage C équivalent au script `pluff.sh`. On utilisera les primitives `pipe`, `dup`, `execlp`.

Exercice 8. Considérer le programme `succ.c` :

succ.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 :
 4 : int main(void)
 5 : {
 6 :   int i;
 7 :
 8 :   scanf("%d", &i);
 9 :   fprintf(stderr, "%d\n", 1);
10 :   printf("%d\n", ++i);
11 :   exit(0);
12 : }
```

On supposera qu'un exécutable `succ` (correspondant au programme `succ.c`) et le fichier `pluff.sh` se trouvent dans un des répertoires de la variable d'environnement `PATH`.

1. Dire ce qui est affiché à l'écran par la commande

   ```
   $ pluff.sh 3 succ
   ```

2. Combien de processus nommés `succ` sont engendrés par la même commande ? Combien de processus nommés `succ` sont engendrés par la commande

   ```
   $ pluff.sh n succ
   ```

   où n est un nombre entier positif.

3. Peut-on remplacer les lignes 9-10 du programme `succ.c` par la ligne

   ```
   printf("%d\n%d\n",1,++i);
                                  ?
   ```

   Expliquer votre réponse.

Un peu de programmation

Exercice 9. Donner le schéma d'un programme serveur. Le serveur écoute sur une tube nommé, et, chaque fois qu'il reçoit une requête de connexion par un processus client, il crée un fils qui s'occupe de traiter la connexion à l'aide d'une nouvelle couple de tubes nommées (la communication est bidirectionnelle) dédiée à cette connexion. Le serveur ne s'arrête jamais.
On peut faire l'hypothèse qu'un client demande une connexion en écrivant son PID dans le tube du serveur. On détaillera seulement la gestion des tubes par le fils, et non le traitement de la connexion.