Système M1 — TD7 : Signaux

Semaine du 2 novembre 2009

Exercice 1 – *SIGALRM et SIGCHLD*

On veut comprendre le programme suivant (en annexe).

1. Donner et justifier les résultats obtenus en lançant la commande `exo1 3` si on tape successivement au clavier les caractères

   `<intr><susp><intr><o><\n><intr><intr><x><\n><o><\n>`

   Même question si on tape

   `<intr><susp><intr><intr><intr><n><\n>`
2. Même question si on tape `<^C><^C><^C>`.
3. Expliquer l'utilisation de tous les signaux par chacun des processus.
4. Que se passera-t-il si on enlève les lignes 20-21 et 32-33 ?

Exercice 2 – *Un `shell` réactif*

On supposera que les commandes n'ont pas d'argument (pas besoin de découper la ligne de commande).

1. Écrire un programme `shell-bis` qui exécute son premier argument (nommé `cmd` par la suite) en imprimant à l'écran son `PID`, puis effectue la boucle suivante : afficher un prompt (par exemple `>` qui attend l'une des deux commandes suivantes : `stop` ou `cont`.
   – Lorsque la commande est `stop`, `shell-bis` doit interrompre l'exécution de `cmd` en lui envoyant un signal `SIGSTOP`.
   – Lorsque la commande est `cont`, `shell-bis` doit faire reprendre l'exécution de `cmd`, en lui envoyant un signal `SIGCONT`.

   ```
   ~$ shell-bis xeyes
   xeyes a le PID: 25756
   > stop
   ```

   À ce moment là, `xeyes` doit être "gelé".

   ```
   > cont
   ```

   Ici, `xeyes` doit reprendre une activité normale.
2. Modifiez `shell-bis` pour que, lorsque `cmd` termine, la valeur de retour soit imprimée à l'écran.
3. Modifiez `shell-bis` pour que, si la commande attendue par le prompt est différente de `cont` ou `stop`, elle exécute cette commande en avant plan, après avoir affiché son `PID`. Afficher la valeur de retour de la commande lorsque celle-ci se termine. Par exemple :

```
> ls
je lance la commande: ls de PID: 2305
toto tata
Le fils de PID: 2305 a termine avec la valeur de retour: 0
>
```

4. Modifiez `shell-bis` de manière à ce que lorsque l'on presse `C-z` (qui génère le caractère `<susp>`, `SIGTSTP` est donc envoyé à tous les processus en avant-plan, y compris `shell-bis`), il se passe les événements suivants :
   – la commande exécutée en avant-plan est suspendue.
   – `shell-bis` reprend la main, en affichant un prompt.
   En particulier, dans `shell-bis`, on ne pourra plus utiliser `wait`, qui attendrait indéfiniment qu'un fils se termine. Par exemple :

   ```
   > xcalc
   je lance la commande: xcalc de PID: 2313
   C-z a ete presse
   >
   ```

5. Rajouter une commande (que l'on appellera `fg`) qui permet de repasser au premier plan le premier processus fils qui est dans un état stoppé. Rajouter aussi la commande `bg` qui permet de faire continuer le premier processus stoppé.
6. Modifiez les commandes `stop` et `cont` de manière à ce qu'elles prennent comme argument (obligatoire) le numéro de `PID` du processus que l'on désire stopper/reprendre.
   On pourra commencer à introduire dans le programme des informations telles que :
   – une table (que l'on pourra prendre de dimension fixée) contenant les informations suivantes : tous les `PID` des processus fils, ainsi qu'un booléen représentant leur état (stoppé ou actif).
   – une variable contenant le numéro de `PID` du dernier processus lancé en avant plan.
   – une variable comptabilisant le nombre total de processus fils, ainsi que toutes les informations que l'on jugera utiles.

   ```
   > xcalc
   je lance la commande: xcalc de PID: 2319
   C-z a ete presse, le processus de PID: 2319 est interrompu
   > cont 2319
   >
   ```

7. Rajouter la fonctionalité `exit` à `shell-bis`.

   ```
   > exit
   Au revoir ...
   ~$
   ```

   On n'oubliera pas d'envoyer les signaux adéquats aux fils, s'il y en a.
8. Intégrez toutes ces fonctionalités au shell que vous avez écrit pendant les TPs précédent. On fera attention aux commandes multiples générées par des tubes `|`, qu'il faudra toutes interrompre ou continuer en même temps.

Code source de exo1.c

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <signal.h>
 4 #include <stdlib.h>
 5 #include <string.h>
 6 #include <sys/wait.h>
 7 #include <setjmp.h>
 8
 9 int cpt=0, maxint;
10 int f;
11 sigjmp_buf env;
12 struct sigaction action;
13
14 void f_pere(int s)
15 {
16   switch (s) {
17   case SIGINT:
18     printf("Aie %d \n", cpt++);
19     if (cpt >= maxint) {
20       action.sa_handler = SIG_IGN;
21       sigaction(SIGINT,&action,NULL);
22       kill(f,SIGUSR1);
23     }
24     break;
25   case SIGCHLD: printf("Au revoir...\n");
26     wait(NULL);
27     exit(0);
28     break;
29   case SIGUSR1:
30     printf("Alors je recommence ...\n");
31     cpt=0;
32     action.sa_handler = f_pere;
33     sigaction(SIGINT,&action,NULL);
34     break;
35   }
36 }
37
38 void f_usr1_fils(int s)
39 {
40   switch (s) {
41   case SIGUSR1:
42     printf("Voulez-vous vraiment quitter?\n");
43     break;
44   case SIGALRM: printf("Timeout\n");
45     kill(getppid(), SIGUSR1);
```

```
46      siglongjmp(env,1);
47      break;
48    }
49
50  }
51  int main(int argc, char ** argv){
52    sigset_t set;
53    sigfillset(&set);
54    sigdelset(&set,SIGUSR1);
55    if (argc < 2) {
56      fprintf(stderr, "Usage: %s argument-entier \n", argv[0]); exit(-1);}
57    maxint = atoi(argv[1]);
58    switch ((f=fork())) {
59    case -1 : fprintf(stderr, "Fork a echoue\n"); exit(-1);
60    case 0: {
61      char flag[10];
62      sigdelset(&set,SIGALRM);
63      action.sa_handler = f_usr1_fils;
64      sigaction(SIGUSR1,&action,NULL);
65      sigaction(SIGALRM,&action,NULL);
66      sigprocmask(SIG_SETMASK,&set,NULL);
67      if (sigsetjmp(env,1) == 0) printf ("premier passage de sigsetjmp\n");
68      while(1){
69        sigsuspend(&set);
70        alarm(3);
71        while(1){
72          printf("Quitter? o/n\n");
73          scanf("%s",&flag);
74          if (! strcmp(flag,"o")) { exit(0);}
75          else if (!strcmp(flag,"n")) break;
76        }
77        kill(getppid(), SIGUSR1);
78      }
79    }
80    default:
81      sigdelset(&set,SIGCHLD);
82      sigdelset(&set,SIGINT);
83      action.sa_handler = f_pere;
84      sigaction(SIGINT,&action,NULL);
85      sigaction(SIGCHLD,&action,NULL);
86      sigaction(SIGUSR1,&action,NULL);
87      sigprocmask(SIG_SETMASK,&set,NULL);
88      while(1) {
89        sigsuspend(&set);
90    }}
91    return 0;
92  }
```