Fiche de TP-TD no. 6

En TP

Vous pouvez (mieux, devez) utiliser, pour ce TP, le code disponible sur la page du cours : créez un répertoire pour ce TP, télé-chargez ces sources, et commencez. En particulier, vous pouvez vous servir de la fonction `mysignal` (voir aussi le cours).

Exercice 1 : *Arrêter un programme.* Considérez le programme suivant :

```
 1 #define TAILLETAMPON 10000
 2
 3 int calculcomplexe(int i){
 4   int somme,j,k;
 5   for(somme=0,j=0;j<i;j++) for(k=0;k<j;k++) somme+=k;
 6   return somme;
 7 }
 8
 9 int main(void){
10   int tab[TAILLETAMPON],i;
11
12   for(i=0;i<TAILLETAMPON;i++)
13     tab[i] = calculcomplexe(i);
14
15   return 0;
16 }
```

La fonction `calculcomplexe` est une fonction nécessitant un temps d'exécution important ! Modifiez ce programme pour que l'utilisateur puisse interrompre le calcul en tapant `CTRL-C` ; l'utilisateur peut alors choisir entre confirmer le calcul ou l'arrêter définitivement.

Exercice 2 : *Timeouts.* Écrivez une source `lirelignetimeout.c` contenant la définition d'une fonction

```
char *lirelignetimeout(char* prompt, int nsecs)
```

Cette fonction demande à l'utilisateur de rentrer une ligne (elle affiche le `prompt` et ensuite fait appel à la fonction `fgets`) et renvoie cette cette ligne (ou `NULL` si erreur) ; si au bout de `nsecs` secondes l'utilisateur n'a rien rentré, alors elle renvoie `NULL`. (Utilisez l'appel système `alarm` pour programmer l'envoi d'un signal `SIGALRM` au bout de `nsecs` secondes.)
Testez votre source, en l'intégrant avec le programme `lirelignetimeout_test.c` (contenant le `main`, pret à l'usage dans les sources).

Exercice 3 : *Comptage de signaux.*

1. Écrire un programme C qui crée 1 fils, puis attend la fin du fils. Le fils envoie le signal `SIGUSR1` au père puis se termine.
2. Modifier le père pour qu'il affiche un message lorsque le signal `SIGUSR1` est capté.
3. Modifier le programme pour qu'il reçoive un entier n sur la ligne de commande, et modifier le fils de telle sorte qu'il envoie à la suite n signaux `SIGUSR1` au père avant de se terminer.
4. Modifier le père pour qu'il compte le nombre de signaux `SIGUSR1` reçus ; supprimer tout affichage dans le handler de signal du père, et afficher le nombre final de signaux reçus après la fin du fils.
5. Tester avec $n = 10, 1000, 100000$. Que peut-on conclure ?

Exercice 4 : *Ping-pong de signaux.*

1. Écrire un programme C qui crée 1 fils, puis attend la fin du fils. Le fils envoie le signal `SIGUSR1` au père. À la réception du signal, le père affiche un message puis envoie le signal `SIGUSR1` au fils. À la réception du signal, le fils affiche un message puis se termine. Le père attend la fin du fils, affiche un message puis se termine.
2. Modifier le programme pour qu'il reçoive un entier n sur la ligne de commande, et modifier le fils et le père pour que l'échange de signaux se passe successivement n fois.

Rappels

– La fonction `signal` n'étant pas portable, on se donne la fonction `mysignal` qui simplifie l'usage de `sigaction` pour placer un handler de signal : voir le cours et les sources.
– Voir `man alarm`.

En TD

Pour ce TD, nous pouvons nous servir de la fonction `mysignal`, vue en TP (voir les rappels).

Exercice 5. Considérez le programme suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <signal.h>
4  #include "mysignal.h"
5  #define SIGNAL(sig,handler) mysignal((sig),(handler),0)
6
7  int x = 0;
8
9  void interruption(int signum) {
10   switch (signum)
11     {
12     case SIGINT:
13       printf("\nCTRL-C\n");
14       x++;
15       break;
16     case SIGQUIT:
17       printf("\nCTRL-\\\n");
18       x--;
19       break;
20     default:
21       printf("\nAutre signal\n");
22       if(!x)
23         SIGNAL(SIGINT, SIG_DFL);
24     }
25 }
26
27 int main(void){
28   SIGNAL(SIGINT, interruption);        /*  Recuperation de CRTL-C */
29   SIGNAL(SIGQUIT, interruption);      /*  Recuperation de CRTL-\ */
30   SIGNAL(SIGTSTP, interruption);      /*  Recuperation de CRTL-Z */
31
32    if (!x)
33      for(;;){
34        printf("-");
35        fflush(stdout);   /* On souhaite imprimer a l'ecran, a chaque seconde !!! */
36        sleep(1);
37      }
38    return EXIT_SUCCESS;
39 }
```

1. Que se passe-t'il si pendant l'exécution de ce programme si l'on tape :
   - `CTRL-C`,
   - `CTRL-C` deux fois,
   - `CTRL-\`,
   - `CTRL-Z`,
   - `CTRL-\` puis `CTRL-Z` puis `CTRL-C` ?
2. Proposez plusieurs façon de quitter ce programme.
3. Que se passe t'il si on enlève la ligne 35 ?

Exercice 6 : *Alarme.*

1. Écrire un programme C qui recopie l'entrée standard sur la sortie standard, caractère par caractère. Toutes les 2 secondes, le programme incrémente un compteur et l'affiche ; il se termine au bout de 5 incrémentations.
2. Écrire un programme C qui crée un fils. Le père recopie l'entrée standard sur la sortie standard, caractère par caractère. Le fils vérifie chaque seconde l'existence du père, et affiche un message et se termine lorsqu'il a détecté la fin du père.

Exercice 7. Écrire un programme `atkill.c` dont l'usage sera `atkill nbsec sig com arg1 .. argn` et qui effectue les opérations suivantes :

1. Il lance la commande `com arg1 .. argn` et signale une éventuelle erreur lors du lancement.
2. Il provoque l'envoi, au bout de `nbsec` secondes, du signal `sig` à la commande puis attend sa terminaison.

Rappels

– La fonction `alarm(unsigned int n)` de `<unistd.h>` provoque l'envoi d'un signal `SIGALRM` au processus appelant au bout de `n` secondes.
– La fonction `kill(pid_t pid, int sig)` appelée avec `sig=0` n'envoie pas de signal mais renvoie `0` si le processus `pid` existe, sinon renvoie `-1`.
– La fonction *mysignal*, voir les sources du TP, est définie ainsi :

```
1  #include <signal.h>
2  #include <stdlib.h>
3  #include "mysignal.h"
4
5  /* Place le handler de signal void h(int) pour le signal sig avec sigaction().
6     Le signal est automatiquement masque' pendant sa delivrance.
7     Si options = 0,
8     - les appels bloquants sont interrompus avec retour -1 et errno = EINTR.
9     - le handler est rearme' automatiquement apres chaque delivrance de signal.
10    Si options est une combinaison bit a bit (operateur |)  de
11    - SA_RESTART : les appels bloquants sont silencieusement repris.
12    - SA_RESETHAND : le handler n'est pas reearme'  .
13    Renvoie le resultat de sigaction.
14 */
15 int mysignal (int sig, void (*h)(int), int options) {
16   struct sigaction s;
17   s.sa_handler = h;
18   sigemptyset (&s.sa_mask);
19   s.sa_flags = options;
20   return sigaction (sig, &s, NULL);
21 }
```