

# Processus : exec, signaux

## Les appels système exec

**Exercice 1.** Écrire votre implémentation de la fonction

```
int system (const char *command);
```

Cette fonction prendra en paramètre un commande avec ses paramètres, l'exécute, et il retourne la valeur de sortie du commande.

Utiliser cette fonction pour écrire un mini shell. Le mini shell, à l'intérieur d'une boucle infinie, invitera à écrire un commande, il exécutera ce commande, et affichera le résultat de l'exécution.

## Un premier exemple

```
signal.c
1 : #include <stdio.h>
2 : #include <stdlib.h>
3 : #include <signal.h>
4 :
5 : int x = 0;
6 :
7 : void interruption(int signum)
8 : {
9 :     signal(signum, interruption);      /* System V */
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 : {
29 :     signal(SIGINT, interruption);      /* Recuperation de CTRL-C */
30 :     signal(SIGQUIT, interruption);     /* Recuperation de CTRL-\\ */
31 :     signal(SIGTSTP, interruption);     /* Recuperation de CTRL-Z */
32 :     for (;;)
33 :     {
34 :         printf("-");
35 :         /* On veut vraiment l'imprimer sur l'ecran !!! */
36 :         fflush(stdout);
37 :         sleep(1);
38 :     }
39 :     return EXIT_SUCCESS;
40 : }
```

**Exercice 2.** Que se passe-t'il si pendant l'exécution de `signal.c` l'on tape :

- CTRL-C,
- CTRL-C deux fois,
- CTRL-\\,
- CTRL-Z,
- CTRL-\\ puis CTRL-Z puis CTRL-C?

Donner plusieurs façon de quitter ce programme.

## Application

**Exercice 3.** On suppose que l'on a un programme contenant la fonction `main` suivante :

```
calcul.c
1 : int main(void)
2 : {
3 :     int i;
4 :     int tab[100];
5 :
6 :     for (i = 0; i < 100; i++)
7 :         tab[i] = calculcomplese(i);
8 :
9 :     return EXIT_SUCCESS;
10 : }
```

La fonction `calculcomplese` est une fonction nécessitant un temps d'exécution important. Modifier 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 4.** Modifier le programme `signal.c` pour que la combinaison de touches `CTRL-C` arrête le processus une fois sur dix.

**Exercice 5.** Écrire un programme `ping.c` qui, lorsqu'il reçoit le signal `SIGQUIT` affiche le message `ping !` et envoie le signal `SIGQUIT` à un numéro de processus donné par l'utilisateur. On utilisera la fonction `system`. Utiliser ce programme pour afficher indéfiniment le message `ping !`.

## Le signal SIGALRM

**Exercice 6.** Utiliser le signal `SIGALRM` et les fonctions `signal` et `alarm` pour tirer au sort un nombre compris entre 0 et 3 (inclus).

**Exercice 7.** Écrire un programme qui affiche l'heure chaque minute puis le message `Dring` à une heure donnée par l'utilisateur.

## L'interface POSIX

**Exercice 8.** Implémenter le programme `signal.c` à l'aide de l'interface de programmations POSIX pour les signaux.

**Exercice 9.** Expliquer ce qu'il se passe dans le code suivant :

```
exbloquage.c
1 : #include <stdio.h>
2 : #include <signal.h>
3 : #include <unistd.h>
4 : #include <limits.h>
5 :
6 : static void handler(int);
7 : static struct sigaction action;
8 : static sigset_t ens;
9 :
10 : int main()
11 : {
12 :     action.sa_handler = handler;
13 :     action.sa_flags = 0;
14 :     sigemptyset(&action.sa_mask);
15 :     sigaction(SIGQUIT, &action, NULL);
16 :
17 :     sigaddset(&action.sa_mask, SIGQUIT);
18 :     sigaction(SIGINT, &action, NULL);
19 :
20 :     while (1)
21 :         sleep(1);
22 :     return 0;
23 : }
24 :
25 : void handler(int signum)
26 : {
27 :     int i;
28 :
29 :     printf("Reçu signal : %d\n", signum);
30 :     sigprocmask(SIG_BLOCK, NULL, &ens);
31 :     printf("Signaux bloqués : ");
32 :     for (i = 1; i < NSIG; i++)
33 :         if (sigismember(&ens, i))
34 :             printf("%d ", i);
35 :     putchar('\n');
36 :     if (signum == SIGINT)
37 :     {
38 :         action.sa_handler = SIG_DFL;
39 :         sigaction(SIGINT, &action, NULL);
40 :     }
41 :     /* Longue tache ... */
42 :     for (i = INT_MAX; i; i--);
43 :
44 :     printf("Sortie du handler\n");
45 : }
```