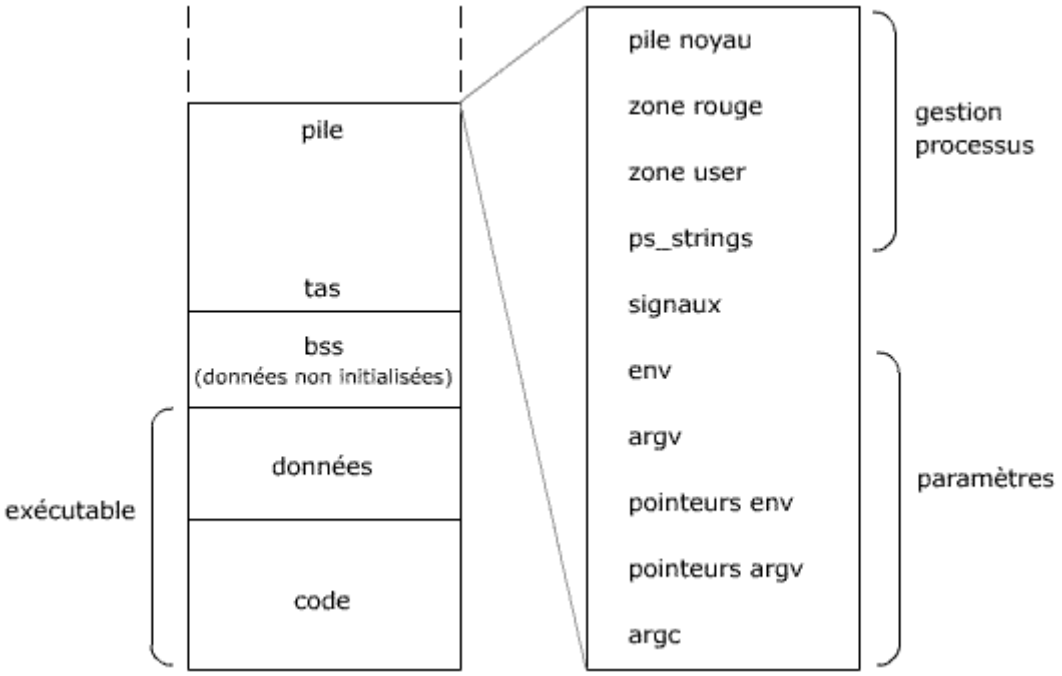
Raphaël CASTANIER

Joffrey RIUS BERNAL

Groupe TP2/ OS308

Compte Rendu TP5

Manipulation des signaux



Année Académique 2017/2018

Sommaire

I. Ignorer les signaux 2
Code source 2
Trace 2
Réponses 3
II. Utilisation du signal SIGINT 4
Code 4
Trace 5
Analyse 5
III. Utilisation des signaux SIGUSR1 et SIGUSR2 6
Code 6
Trace 7
Analyse 8

I. Ignorer les signaux

EXERCICE 1

Ecrire un programme qui ignore TOUS les signaux.

Code source

```
/*
    Commande de compilation : gcc -o prog exo1.c -Wall -Werror
*/

// Inclusions nécessaires à l'utilisation des fonctions
#include <stdio.h>          // printf
#include <signal.h>         // signal
#include <string.h>         // strsignal
#include <unistd.h>         // sleep

typedef void (*sighandler_t)(int);

int main(void)
{
    // PID du programme pour lui envoyer un sigal SIGKILL
    printf("--- PID = %d\n", getpid());

    int Nb_Sig;
    for(Nb_Sig = 1; Nb_Sig < NSIG ; Nb_Sig ++)
    {
        // pour chaque signal du PCB, on remplace le signal par defaut
        // par la fonction SIG_IGN
        sighandler_t s = signal(Nb_Sig, SIG_IGN);
        if(s==SIG_ERR)
        {
            printf("Le signal %s (%i) ne peut pas etre modifie.\n",
strsignal(Nb_Sig), Nb_Sig);
        }
    }

    /* Attendre des signaux */
    while(1)
    {
        sleep(5);
    }
    return 0;
}
```

Trace

```
--- PID = 762
Le signal Killed (9) ne peut pas etre modifie.
Le signal Stopped (signal) (19) ne peut pas etre modifie.
Le signal Unknown signal 32 (32) ne peut pas etre modifie.
Le signal Unknown signal 33 (33) ne peut pas etre modifie.
^C
^Z
```

```
[1]+  Arrêté                 ./prog
(…)$ fg
^C
^Z
Processus arrêté
```

Réponses

1. La trace ci-dessus nous indique les signaux qu'on ne peut pas ignorer :
   - 9 (SIGKILL) : ce signal permet de tuer le processus. Il ne peut pas être modifié car c'est un appel système indispensable, il permet de tuer immédiatement un processus qui peut être nuisible, ou tout simplement d'arrêter la machine !

   - 19 (SIGSTOP) : ce signal permet de mettre le processus en pause. Il interrompt l'exécution du processus immédiatement. Le processus garde ses propriétés et peut reprendre (par exemple avec fg ou bg).

   - 32 et 33 : ces deux signaux ne sont pas modifiables, car ils n'existent pas dans un noyau linux x64. En effet, ces deux numéros ne sont pas attribués à des signaux (peut être pour que le système fasse la jointure entre les 31 premiers signaux et les 31 derniers, dans la table des signaux). Lorsque signal est appelée sur ces deux numéros, le kernel a renvoyé la fonction SIG_ERR, non pas parce qu'on ne peut pas « modifier » ces signaux, mais tout simplement parce qu'ils n'existent pas !

2. On constate que les constantes symboliques sont des chaines de caractère en anglais, et ne sont pas les constantes de pré-compilation comme SIGINT. Cela permet par exemple de retourner à l'utilisateur (quoique plutôt réservé au debug), un message d'erreur ou d'information « explicite » lors de la réception du signal.

3. Les signaux CTRL+Z ou CTRL+C n'ont pas d'effet sur le processus, car ils sont associés à des signaux ignorés (respectivement 2 et 20). Cependant, le signal SIGSTOP (19) arrête le processus car il ne peut pas être ignoré. On fera attention à la trace ci-dessus qui montre un appel à la commande fg après la mise en pause du processus. Le programme reçoit alors le signal 18 (voir exercice 3) qui correspond à la reprise du fonctionnement du processus. Ce signal n'est pas ignoré car le processus reprend bel et bien (le prompt qui ne revient pas en est un indice). La dernière ligne de la trace correspond à la réception du signal 9 (SIGKILL), ce qui est cohérent.

II. Utilisation du signal SIGINT

EXERCICE 2

Ecrire un programme qui génère deux processus, père et fils. Le fils se met dans une boucle d'attente. Après 5 secondes, le père lui envoie le signal `SIGINT` pour l'arrêter. A la réception, le fils attend 2 secondes et s'arrête. Le père attend la fin du fils avant de s'arrêter lui aussi.

Code

```
/*
    Commande de compilation : gcc -o prog exo2.c -Wall -Werror
*/


// Inclusions nécessaires à l'utilisation des fonctions
#include <stdio.h>          // printf
#include <signal.h>         // kill
#include <unistd.h>         // sleep
#include <stdlib.h>         // exit
#include <sys/wait.h>       // wait
#include <sys/types.h>      // pid_t


/* Fonction de fin du fils */
void finDuFils(int sig)
{
    if(sig==SIGINT)
    {
        printf("Fils : Je vais m'arreter dans 2 secondes.\n");
        sleep(2);
        exit(0);
    }
}

int main(void)
{
    // PID du programme pour lui envoyer un sigal SIGKILL
    printf("--- PID = %d\n", getpid());

    pid_t val= fork(); // création du fils

    switch(val)
    {
        case 0 : // fils
            signal(SIGINT, finDuFils); // modification du fils pour qu'il
réagisse au signal de son père.
            printf("Fils : Je vais aller dormir\n");
            while(1)
            {
                sleep(1);
            }
            break;

        case -1 : // erreur
            printf("Erreur lors du fork\n");
            break;
```

```
        default : // père
            sleep(5);
            printf("Pere : J'envoie le signal d'interruption a mon
fils.\n");
            kill(val, SIGINT); // envoi du signal d'interruption au fils
            printf("Pere : J'attends mon fils\n");
            wait(NULL); // attente du fils
            printf("Pere : Mon fils a terminé, je vais terminer aussi.\n");
    }
    return 0;
}
```

Trace

```
--- PID = 800

Fils : Je vais aller dormir

Pere : J'envoie le signal d'interruption a mon fils.

Pere : J'attends mon fils

Fils : Je vais m'arreter dans 2 secondes.

Pere : Mon fils a terminé, je vais terminer aussi.
```

Analyse

Le programme réagit selon la consigne, le comportement du fork a été respecté et les délais sont corrects.

III. Utilisation des signaux SIGUSR1 et SIGUSR2

EXERCICE 3

Ecrire un programme qui :

1. Affiche son numéro (`pid`) via l'appel à `getpid()`
2. Traite tous les signaux, sauf `SIGUSR1` et `SIGUSR2`, par une fonction `fonc` qui se contente d'afficher le numéro du signal reçu.
3. Traite le signal `SIGUSR1` par une fonction `fonc1` et le signal `SIGUSR2` par `fonc2` :

(a) `fonc1` affiche le numéro du signal reçu et la liste des utilisateurs de la machine (appel à la commande `who` par `system("who")`)

(b) `fonc2` affiche le numéro du signal reçu et l'espace disque utilisé sur la machine (appel à la commande `df .` par `system("df .")`)

Code

```c
/*
    Commande de compilation : gcc -o prog exo3.c -Wall -Werror
*/

// Inclusions nécessaires à l'utilisation des fonctions
#include <stdio.h>          // printf
#include <signal.h>         // kill
#include <unistd.h>         // sleep
#include <stdlib.h>         // exit
#include <string.h>         // strsignal
#include <sys/wait.h>       // wait
#include <sys/types.h>      // pid_t

/*************** La fonction fonc *************/
void fonc (int NumSignal)
{
    printf("Signal recu : %d (%s)\n", NumSignal, strsignal(NumSignal));
}

/*************** La fonction fonc1 *************/
void fonc1 (int NumSignal)
{
    fonc(NumSignal); // appelle la fonction par défaut
    system("who");
}

/*************** La fonction fonc2 *************/
void fonc2 (int NumSignal)
{
    fonc(NumSignal); // appelle la fonction par défaut
    system("df .");
}


int main (void)
{
    // PID du programme pour lui envoyer un sigal SIGKILL
    printf("--- PID = %d\n", getpid());
```

```
    // on remplace toutes les fonctions que l'on peut par func
    int Nb_Sig;
    for(Nb_Sig = 1; Nb_Sig < NSIG ; Nb_Sig ++)
    {
        signal(Nb_Sig, fonc);
    }

    // on modifie spécialement SIGUSR1 et SIGUSR2
    signal(SIGUSR1, fonc1);
    signal(SIGUSR2, fonc2);

    while (1)
    {
        sleep(5); /* Attendre les signaux */
    }

    return 0;
}
```

Trace

```
--- PID = 832

^CSignal recu : 2 (Interrupt)

^ZSignal recu : 20 (Stopped)

Signal recu : 1 (Hangup)

Signal recu : 2 (Interrupt)

Signal recu : 3 (Quit)

Signal recu : 10 (User defined signal 1)

Signal recu : 17 (Child exited)

Signal recu : 12 (User defined signal 2)

Sys. de fichiers blocs de 1K  Utilisé Disponible Uti% Monté sur

C:                  73039868 60537084   12502784  83% /mnt/c

Signal recu : 17 (Child exited)


[1]+  Arrêté                 ./prog

$ fg

./prog

Signal recu : 18 (Continued)

Processus arrêté
```

Analyse

- Les signaux peuvent être envoyés :
    - Par une commande dans le terminal (CTRL+Z ou CTRL+C)
    - Par un autre programme (kill)
    - Par un fils (SIGCHLD)
- Les commandes CTRL+C et CTRL+Z envoient respectivement les signaux 2 et 20 au processus. Ces deux signaux sont interceptés et traités de manière différente du défaut, c'est pourquoi on n'a pas le comportement habituel (CTRL+C doit interrompre le processus, CTRL+Z le met en pause).
- On notera que sur la trace ci-dessus, la commande who n'a rien affiché (lancé sous Windows Subsystem Linux). Ce résultat est normal avec cette distribution Linux.
- Lorsque les commandes appelées par la fonction system sont terminées, le programme reçoit le signal SIGCHLD (17). En effet, la commande system crée un fils du programme et exécute un binaire particulier (celui passé en paramètre). Donc la fin du fils renvoie correctement le signal SIGCHLD si il s'est terminé. Cependant, nous n'avons pas observé ici la valeur retournée par le fils (EXIT_SUCCESS ou autre…).
- Lorsque le programme reprend avec la commande fg, le signal SIGCONT (18) est envoyé au processus.