El Mehdi CHOUHAM

STR : TP : Signaux

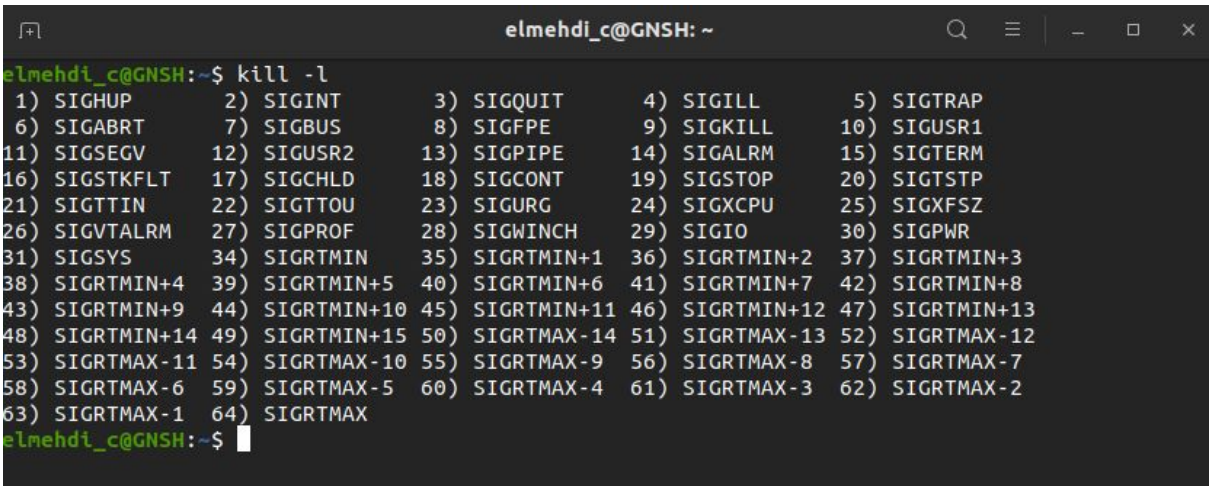
CHAPITRE 1 : Présentation Générale

Cette présentation contient des extraits du code, les programmes sources complets ainsi qu'un fichier Makefile pour les compiler peuvent être retrouvés dans l'archive rendu.

Question 1 :

Afin d'afficher les signaux supportés par le système, on utilise l'option " l " de la commande kill. On liste donc les signaux pris en charge grâce à :

kill -l



Question 2 :

a - Afin d'afficher uniquement le numéro du signal SIGTSTP, on utilise la commande :

kill -l SIGTSTP ou **kill -l TSTP**

b - Afin d'afficher le nom symbolique du signal n°10, on utilise la commande :

kill -l 10

Question 3 :

a - SIGINT représente une interruption du clavier. Pour produire un SIGINT, il suffit d'effectuer un **Ctrl+C** dans un terminal

b - SIGCHLD réveille à un processus dont un de ses fils vient de mourir (terminé ou stoppé). Ce signal est ignoré mais sert à réveiller un processus endormi en priorité interruptible.

c - SIGTSTP suspend/ interrompt le processus en cours d'exécution. SIGTSTP peut être généré par un **Ctrl+Z.** Le processus suspendu peut être repris suite à un signal SIGCONT.

CHAPITRE 2 : Traitement des signaux

Question 4 :

A l'exécution en arrière plan (./infini &) du programme compilé, on remarque à partir de l'exécution de la commande plusieurs fois :

ps -o pid,ni,pri,stat,pcpu,time | grep <pid>

que le processus consomme de plus en plus de ressources CPU (ralentissement du système, bruits de ventilateurs…)

- L'exécution d'un signal : **kill -INT <PID de infini>** : interrompt ce processus

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc infini.c -o infini
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./infini &
[1] 22342
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22342
  22342   0  19 R    99.2 00:00:52
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22342
  22342   0  19 R     100 00:00:56
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22342
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps
    PID TTY          TIME CMD
  22264 pts/0    00:00:00 bash
  22360 pts/0    00:00:00 ps
[1]+  Interrompre             ./infini
```

- L'exécution d'un signal : **kill -CONT <PID de infini>** : est ignoré et le processus reste actif.

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./infini &
[1] 22532
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
  22532   0  19 R    97.2 00:00:11
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
  22532   0  19 R    94.5 00:00:14
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
  22532   0  19 R    97.8 00:00:15
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -CONT 22532
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
  22532   0  19 R    98.0 00:00:41
```

- L'exécution d'un signal : **kill -USR1 <PID de infini>** : interrompt également le processus, avec un message d'interruption utilisateur.

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
  22532   0  19 R    98.0 00:00:41
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -USR1 22532
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 22532
[1]+  Signal #1 défini par l'usager      ./infini
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps
    PID TTY          TIME CMD
  22264 pts/0    00:00:00 bash
  22704 pts/0    00:00:00 ps
```

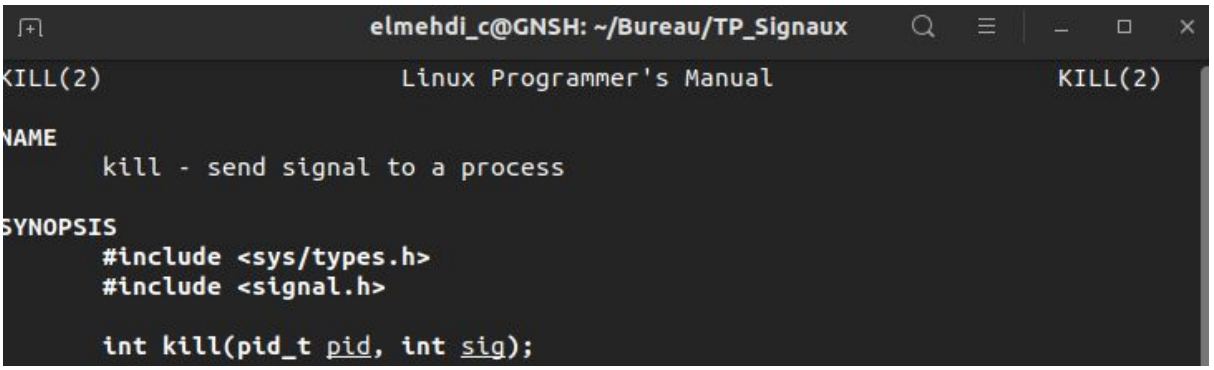
- L'exécution d'un signal : **kill -FPE <PID de infini>** : interrompt aussi le processus avec une sauvegarde de l'image mémoire du processus (core dumped).

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./infini &
[1] 22853
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -FPE 22853
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps
    PID TTY          TIME CMD
  22264 pts/0    00:00:00 bash
  22868 pts/0    00:00:00 ps
[1]+  Exception en point flottant   (core dumped) ./infini
elmehdi_c@GNSH:~/Bureau/TP_Signaux$
```

Question 5 :

a - On peut trouver de l'aide sur l'appel système *kill()* dans la section 2 des manuels, donc par la commande :

man 2 kill



b - Oui, l'appel système kill() est bien conforme au standard POSIX

c - Oui, on peut envoyer un même signal à tous les processus appartenant au groupe du processus appelant si le paramètre pid est égale à ***0***, et à tous les processus sauf init si celui-ci est égale à ***-1***.

d - **int resultat = kill(1234, 0)** : permet de vérifier l'existence ainsi que les permissions du processus 1234 sans lui envoyer de signal.

Question 6 :

De la même manière, on peut trouver de l'aide sur l'appel système *alarm()* dans la section 2 des manuels, donc par la commande :

man 2 alarm

La fonction de : **int alarm(unsigned int seconds)** : est de planifier l'envoie d'un signal SIGALRM au processus appelant dans *seconds* secondes.

```
ALARM(2)                  Linux Programmer's Manual                 ALARM(2)

NAME
       alarm - set an alarm clock for delivery of a signal

SYNOPSIS
       #include <unistd.h>

       unsigned int alarm(unsigned int seconds);

DESCRIPTION
       alarm()  arranges  for  a SIGALRM signal to be delivered to the calling
       process in seconds seconds.

       If seconds is zero, any pending alarm is canceled.

       In any event any previously set alarm() is canceled.

RETURN VALUE
       alarm() returns the number of seconds remaining  until  any  previously
       scheduled alarm was due to be delivered, or zero if there was no previ-
       ously scheduled alarm.
```

Question 7 :

On peut trouver de l'aide sur l'appel système *pause()* dans la section 2 des manuels, donc par la commande :

man 2 pause()

Sa fonction est de mettre le processus appelant au sommeil jusqu'à ce qu'un signal d'arrêt ou une fonction de gestionnaire de signal soit appelée.

```
PAUSE(2)                  Linux Programmer's Manual                 PAUSE(2)

NAME
       pause - wait for signal

SYNOPSIS
       #include <unistd.h>

       int pause(void);

DESCRIPTION
       pause()  causes the calling process (or thread) to sleep until a signal
       is delivered that either terminates the process or causes  the  invoca-
       tion of a signal-catching function.

RETURN VALUE
       pause()  returns  only when a signal was caught and the signal-catching
       function returned.  In this case, pause() returns -1, and errno is  set
       to EINTR.

ERRORS
       EINTR  a signal was caught and the signal-catching function returned.
```

Question 8 :

le programme de infini_v2 :

```
 GNU nano 4.8                          infini_v2.c
#include <unistd.h>
int main(void)
{
        alarm(10);
        pause();
        return 0;
}
```

De la même manière que “infini” de la question 4 ; On exécute le programme en arrière-plan (./infini_v2 &).

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc infini_v2.c -o infini_v2
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./infini_v2 &
[1] 25082
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 25082
  25082   0  19 S     0.0 00:00:00
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 25082
  25082   0  19 S     0.0 00:00:00
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps -o pid,ni,pri,stat,pcpu,time | grep 25082
[1]+  Minuterie d'alerte      ./infini_v2
elmehdi_c@GNSH:~/Bureau/TP_Signaux$
```

On remarque à partir de l'exécution répétée de la commande :

ps -o pid,ni,pri,stat,pcpu,time | grep <pid de infini_v2>

que la consommation du processeur reste égale à **0**, un temps cumulé du CPU égale à **0** aussi et un état de sommeil jusqu’à l’écoulement de la minuterie d'alerte qui arrête le processus via l’envoi d’un signal **SIGALRM.** C’est ce qu’on appelle une *attente passive.*

Question 9 :

a - On affiche les occurrences avec numérotation du fichier d’entête **signal.h** : par la commande : **locate signal.h | nl** par exemple.

Sur ma machine, la commande retourne *121* occurrences.

```
  99  /usr/src/linux-headers-5.4.0-53/arch/nios2/include/uapi/asm/signal.h
 100  /usr/src/linux-headers-5.4.0-53/arch/parisc/include/asm/signal.h
 101  /usr/src/linux-headers-5.4.0-53/arch/parisc/include/uapi/asm/signal.h
 102  /usr/src/linux-headers-5.4.0-53/arch/powerpc/include/asm/signal.h
 103  /usr/src/linux-headers-5.4.0-53/arch/powerpc/include/uapi/asm/signal.h
 104  /usr/src/linux-headers-5.4.0-53/arch/s390/include/asm/signal.h
 105  /usr/src/linux-headers-5.4.0-53/arch/s390/include/uapi/asm/signal.h
 106  /usr/src/linux-headers-5.4.0-53/arch/sh/include/uapi/asm/signal.h
 107  /usr/src/linux-headers-5.4.0-53/arch/sparc/include/asm/compat_signal.h
 108  /usr/src/linux-headers-5.4.0-53/arch/sparc/include/asm/signal.h
 109  /usr/src/linux-headers-5.4.0-53/arch/sparc/include/uapi/asm/signal.h
 110  /usr/src/linux-headers-5.4.0-53/arch/x86/include/asm/signal.h
 111  /usr/src/linux-headers-5.4.0-53/arch/x86/include/asm/fpu/signal.h
 112  /usr/src/linux-headers-5.4.0-53/arch/x86/include/uapi/asm/signal.h
 113  /usr/src/linux-headers-5.4.0-53/arch/xtensa/include/asm/signal.h
 114  /usr/src/linux-headers-5.4.0-53/arch/xtensa/include/uapi/asm/signal.h
 115  /usr/src/linux-headers-5.4.0-53/include/asm-generic/audit_signal.h
 116  /usr/src/linux-headers-5.4.0-53/include/asm-generic/signal.h
 117  /usr/src/linux-headers-5.4.0-53/include/linux/signal.h
 118  /usr/src/linux-headers-5.4.0-53/include/linux/sched/signal.h
 119  /usr/src/linux-headers-5.4.0-53/include/trace/events/signal.h
 120  /usr/src/linux-headers-5.4.0-53/include/uapi/asm-generic/signal.h
 121  /usr/src/linux-headers-5.4.0-53/include/uapi/linux/signal.h
elmehdi_c@GNSH:~$
```

b - Les fichiers d'entête utilisés par ggc pour les directives "include *.h" des programmes sources sont situés dans le dossier **/usr/include/**. Donc le fichier utilisé est : "**/usr/include/signal.h**"

c - La commande : **cat /usr/include/signal.h | grep "SIGSTOP"** n'affiche pas de résultat. Donc ces constantes symboliques ne sont pas définies dans signal.h mais dans un de plusieurs fichier *.h appelé via la directive "#include" dans le fichier **signal.h.**
On peut voir ces fichier en affichant signal.h via la commande **cat** par exemple.

d - On peut afficher les fichiers d'entête contenant la définition de SIGSTOP par exemple via la commande :

grep -rnw '/usr/include/' -e 'SIGSTOP'

```
elmehdi_c@GNSH:/$ grep -rnw '/usr/include/' -e 'SIGSTOP'
/usr/include/asm-generic/signal.h:30:#define SIGSTOP         19
/usr/include/x86_64-linux-gnu/asm/signal.h:40:#define SIGSTOP         19
/usr/include/x86_64-linux-gnu/bits/signum.h:44:#undef   SIGSTOP
/usr/include/x86_64-linux-gnu/bits/signum.h:45:#define  SIGSTOP        19
/usr/include/x86_64-linux-gnu/bits/signum-generic.h:67:#define  SIGSTOP       1
7       /* Stop, unblockable.  */
elmehdi_c@GNSH:/$
```

Les fichiers des dossiers /asm sont consacrés à un cross-compiler installé sur ma machine.
La définition des constantes symboliques désignant les signaux sont dans les fichiers :

/usr/include/x86_64-linux-gnu/bits/signum.h

/usr/include/x86_64-linux-gnu/bits/signum-generic.h

Question 10 :

a - On utilise un grep récursif afin de retrouver tous les fichiers présents dans l'arborescence de racine /usr/include contenant le terme sig_set_t ;

grep -r "sig_set_t ;" /usr/include

Le résultat de la commande est donc le suivant :

b - On utilise un grep récursif afin de retrouver tous les fichiers présents dans l'arborescence de racine /usr/include contenant le terme __sig_set_t ;

grep -r "__sig_set_t ;" /usr/include

Le résultat de la commande est donc le suivant :

c - On

Question 11 :

Le programme pour : *(Application n°2, appelée signal1.c dans les prochaines questions)*

- inhiber le traitement du contrôle-C
- S'arrête n secondes après son lancement où n est le nombre en secondes présent en argument de la ligne de commande, ou 10 s par défaut

```
  GNU nano 4.8                          application_n2.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#include <signal.h>

// Définition de la fonction check :
#define CHECK(sts, msg) \
        if (-1 == (sts)) {      \
                perror(msg) ;  \
        exit(EXIT_FAILURE) ;    \
        }

#define SECONDES_DEFAUT 10


int main (int argc , char *argv[])
{
        sigset_t masque_nouv;
        sigset_t masque_anc;

        int secondes = 0 ;

        if (argc > 1) {
                secondes = atoi(argv[1]) ;
        }else{
                secondes = SECONDES_DEFAUT ;
        }

        printf("Inhibition du Ctrl+C. Arrêt dans : %d \n",secondes) ;

        //Initialisation du masque des signaux à ajouter
        CHECK(sigemptyset(&masque_nouv), "sigemptyset()") ;

        //Ajout de SIGINT
        CHECK(sigaddset(&masque_nouv , SIGINT), "sigaddset(SIGINT)") ;

        //Ajout dans le masque et sauvegarde de l'ancien masque
        CHECK(sigprocmask(SIG_BLOCK , &masque_nouv , &masque_anc), "sigprocmask()") ;
        CHECK(alarm(secondes), "alarm()") ;
        CHECK(pause(), "pause()") ;

        exit(0) ;

        return 0;
}
```

- On teste le programme sans arguments puis pour une entrée de 15 secondes.

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc -Wall application_n2.c -o inhib
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib
Inhibition du Ctrl+C. Arrêt dans : 10
Minuterie d'alerte
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib 15
Inhibition du Ctrl+C. Arrêt dans : 15
Minuterie d'alerte
```

- On remarque que **Ctrl+C** n'interrompt pas l'exécution :

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib 15
Inhibition du Ctrl+C. Arrêt dans : 15
^C^C^C^C^C^C
^C^C^C^C[1]+  Minuterie d'alerte      ./inhib
Minuterie d'alerte
```

Question 12 :

On reprend le programme de la *question 11*, dans un programme *signal2.c*. Ensuite, on met en commentaire l'opération de masquage du signal SIGINT de la fonction main(), qu'on remplacera par une ignorance du signal SIGINT grâce à SIG_IGN.

Nouvelle fonction main *(programme signal2.c)* :

```
  GNU nano 4.8                          signal2.c
        exit(EXIT_FAILURE) ;    \
        }

#define SECONDES_DEFAUT 10


int main (int argc , char *argv[])
{
        //sigset_t masque_nouv;
        //sigset_t masque_anc;

        int secondes = 0 ;

        if (argc > 1) {
                secondes = atoi(argv[1]) ;
        }else{
                secondes = SECONDES_DEFAUT ;
        }

        printf("Inhibition du Ctrl+C. Arrêt dans : %d \n",secondes) ;

        /*
        //Initialisation du masque des signaux à ajouter
        CHECK(sigemptyset(&masque_nouv), "sigemptyset()") ;
        //Ajout de SIGINT
        CHECK(sigaddset(&masque_nouv , SIGINT), "sigaddset(SIGINT)") ;
        //Ajout dans le masque et sauvegarde de l'ancien masque
        CHECK(sigprocmask(SIG_BLOCK , &masque_nouv , &masque_anc), "sigprocmask>
        */

        //Initisalisation de la sigaction comportant un traitement SIG_IGN
        struct sigaction action_nouv;
        struct sigaction action_anc;

        action_nouv.sa_handler = SIG_IGN;
        CHECK(sigemptyset(&action_nouv.sa_mask), "sigemptyset()");
        action_nouv.sa_flags = 0;

        //Installation du gestionnaire pour SIGINT
        CHECK(sigaction(SIGINT, &action_nouv, &action_anc), "sigaction()")

        CHECK(alarm(secondes), "alarm()") ;
        CHECK(pause(), "pause()") ;

        exit(0) ;

        return 0;
}
```

- En effectuant des **Ctrl+C**, on remarque le même fonctionnement du programme de la *question 11. Cette fois si le signal est reconnu mais le nouveau traitement est de l'ignorer (SIG_IGN).*

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc -Wall signal2.c -o inhib2
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib2
Inhibition du Ctrl+C. Arrêt dans : 10
Minuterie d'alerte
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib2 20
Inhibition du Ctrl+C. Arrêt dans : 20
^C^C^C^C^C^C^C
^C
Minuterie d'alerte
```

Question 13 :

On reprend le programme de la *question 12*, dans un programme *signal3.c.*
Comme illustré dans l'exemple de structure d'un gestionnaire des signaux SIGCHLD, SIGUSR1, SIGUSR2 et SIGALRM, on ajoute une fonction gestionnaire qui affiche le message : *" Le contrôle-C est désactivé "* dès qu'un **SIGINT** est reçu.

(Les blocs commentés de la question 11 seront supprimés pour plus de lisibilité)

```c
static void signalHandler(int); //prototype de la fonction du gestionnaire

int main (int argc , char *argv[])
{
        int secondes = 0 ;

        if (argc > 1) {
                secondes = atoi(argv[1]) ;
        }else{
                secondes = SECONDES_DEFAUT ;
        }

        printf("Inhibition du Ctrl+C. Arrêt dans : %d \n",secondes) ;

        //Initisalisation de la sigaction comportant un traitement SIG_IGN
        struct sigaction action_nouv;
        struct sigaction action_anc;

        action_nouv.sa_handler = signalHandler; //appel du gestionnaire défini
        CHECK(sigemptyset(&action_nouv.sa_mask), "sigemptyset()");
        action_nouv.sa_flags = 0;

        //Installation du gestionnaire pour SIGINT
        CHECK(sigaction(SIGINT, &action_nouv, &action_anc), "sigaction()")

        CHECK(alarm(secondes), "alarm()") ;
        CHECK(pause(), "pause()");

        exit(0) ;

        return 0;
}

static void signalHandler(int numSig) //programme de la fonction du gestionnaire
{
        switch(numSig){
                case SIGINT :
                        printf("\n Le contrôle-C est désactivé !\n");
                        break;
                default :
                        printf("\n Signal %d non traité \n", numSig);
                        break;
        }
}
```

Le programme affiche le message à la réception d'un Ctrl+C, mais celui-ci est interrompu.

Question 14 :

a - Comme mentionné ci-dessus, en appuyant sur **Ctrl+C**, le programme affiche le message : “ Le contrôle-C est désactivé “ ,mais ce dernier est interrompu par un message d’erreur :

pause(): Interrupted system call

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc -Wall signal3.c -o inhib3
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib3
Inhibition du Ctrl+C. Arrêt dans : 10
^C
 Le contrôle-C est désactivé !
pause(): Interrupted system call
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib3 15
Inhibition du Ctrl+C. Arrêt dans : 15
^C
 Le contrôle-C est désactivé !
pause(): Interrupted system call
```

b - En consultant :

man 2 pause

```
RETURN VALUE
       pause()  returns only when a signal was caught and the signal-catching func-
       tion returned.  In this case, pause() returns -1, and errno is set to EINTR.
```

On peut voir que le seul cas où **pause()** peut avoir une valeur de retour est quand un signal a été intercepté et qu’une valeur a été retourné par le gestionnaire qui le traite.

c - Afin de corriger ce problème, on peut par exemple remplacer :

```
CHECK(pause(), "pause()");
```

par :

```
while (pause() == -1) ;
```

Ce qui donne :

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib3
Inhibition du Ctrl+C. Arrêt dans : 10
^C
 Le contrôle-C est désactivé !
^C
 Le contrôle-C est désactivé !
^C
 Le contrôle-C est désactivé !
^C
 Le contrôle-C est désactivé !
Minuterie d'alerte
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib3 15
Inhibition du Ctrl+C. Arrêt dans : 15
^C
 Le contrôle-C est désactivé !
^C
 Le contrôle-C est désactivé !
Minuterie d'alerte
```

Question 15 :

On crée un programme *signal4.c* respectant le cahier des charges de la question 15 :
Il doit :

- Commencer par afficher le temps restant indiqué en argument et désactivation de la fonction de **Ctrl+C**
- Afficher le temps restant avant réactivation à chaque fois que **Ctrl+C** est effectué
- Indiquer le retour au fonctionnement normal de **Ctrl+C**, et l'arrêt si un **Ctrl+C** est effectué

```c
static void signalHandler(int); //prototype de la fonction du gestionnaire

int secondes = 0;//la variable contenant le temps restant devient globale

int main (int argc , char *argv[])
{
        if (argc > 1) {
                secondes = atoi(argv[1]) ;
        }else{
                secondes = SECONDES_DEFAUT ;
        }

        printf("Le Ctrl-C est désactivé pendant : %d \n",secondes);

        //Initisalisation de la sigaction
        struct sigaction action_nouv;
        struct sigaction action_anc;

        action_nouv.sa_handler = signalHandler; //appel du gestionnaire défini
        CHECK(sigemptyset(&action_nouv.sa_mask), "sigemptyset()");
        action_nouv.sa_flags = 0;

        //Installation du gestionnaire pour SIGINT
        CHECK(sigaction(SIGINT, &action_nouv, &action_anc), "sigaction()")
        //Installation du gestionnaire pour SIGALRM
        CHECK(sigaction(SIGALRM, &action_nouv, &action_anc ), "sigaction")

        CHECK(alarm(1), "alarm()") ;
        while (pause() == -1);

        exit(0) ;

        return 0;
}

static void signalHandler(int numSig) //programme de la fonction du gestionnaire
{
        switch(numSig){
                case SIGINT :
                        printf("Le Ctrl-c est désactivé pendant : %d \n",second>
                        break;
                case SIGALRM :
                        secondes--;
                        if (secondes > 0){
                                CHECK(alarm(1), "alarm()");
                        }else{
                                struct sigaction action_nouv;
                                struct sigaction action_anc;
                                //initialisation de la stucture sigaction
                                action_nouv.sa_handler = SIG_DFL;
                                CHECK(sigemptyset(&action_nouv.sa_mask),"sigemp>
                                action_nouv.sa_flags = 0;
                                //Installation du gestionnaire
                                CHECK(sigaction(SIGINT, &action_nouv, &action_a>
                                printf("Le traitement normal du Ctr-C est réact>
                        }
                        break;
                default :
                        printf("\n Signal %d non traité \n", numSig);
                        break;
        }
```

Tests et exemples d'utilisation :

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc -Wall signal4.c -o inhib4
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib4
Le Ctrl-C est désactivé pendant : 10
^CLe Ctrl-c est désactivé pendant : 9
^CLe Ctrl-c est désactivé pendant : 7
Le traitement normal du Ctr-C est réactivé
^C
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./inhib4 5
Le Ctrl-C est désactivé pendant : 5
^CLe Ctrl-c est désactivé pendant : 5
^CLe Ctrl-c est désactivé pendant : 4
^CLe Ctrl-c est désactivé pendant : 3
Le traitement normal du Ctr-C est réactivé
^C
```

Question 16 :

En s'aidant du code de la partie a) de l'énoncé, on crée le programme *signal5*.
En exécutant la commande :

kill -INT <pid du processus fils>

On obtient :

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./signal5 20
Le Ctrl-C est désactivé pendant : 20
Je suis le père du processus ..... n°22628
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 14
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 13
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 12
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 11
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 10
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ kill -INT 22628
Le Ctrl-c est désactivé pendant : 1
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ Le traitement normal du Ctr-C est réactivé
```

On modifie ensuite le code pour avoir *signal5b.c* comme indiqué dans la question 16 :

```
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ gcc -Wall signal5b.c -o signal5b
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps
    PID TTY          TIME CMD
  22253 pts/0    00:00:00 bash
  24396 pts/0    00:00:00 ps
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ./signal5b 20
Le Ctrl-C est désactivé pendant : 20
Je suis le père du processus ..... n°24407
^CLe Ctrl-c est désactivé pendant : 20
Le Ctrl-c est désactivé pendant : 19
wait(): Interrupted system call
elmehdi_c@GNSH:~/Bureau/TP_Signaux$ ps
    PID TTY          TIME CMD
  22253 pts/0    00:00:00 bash
  24407 pts/0    00:00:00 signal5b
  24412 pts/0    00:00:00 ps
```

On remarque que le processus père ne s'arrête pas, même si le **Ctrl+C** est traité par les deux processus.
A la fin des 20s de test le processus fils reçoit le SIGINT et s'arrête.

Question 17 :

De la manière qu'à la *question 13 - c,* on peut remplacer :

```
CHECK(wait(&status), "wait()");
```

par :

```
while (wait(&status) == -1);
```

CHAPITRE 3 : Exercice de synthèse

Question 18 :

On écrit un programme respectant le cahier des charges.

```
 GNU nano 4.8                        Q18/Synthese.c
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>

// Définition de la fonction check :
#define CHECK(sts, msg) \
        if (-1 == (sts)) {      \
                perror(msg) ;  \
        exit(EXIT_FAILURE) ;    \
        }

#define PASSWORD "ESSAI"
#define MAXLEN  8
#define DUREE_MAX   30
#define ESSAIS_MAX  3

int temps_restant = 0;

// Prototypes :
int proc_fils(int);
static void signalHandler(int);

//Code du main :
int main(int argc, char *argv[])
{
    int max_essais, max_duree, status;
    int nb_echecs = 0;
    struct sigaction action_nouv;
    pid_t procFils, w;

    if (argc > 2){
        max_duree = atoi(argv[1]);
        max_essais = atoi(argv[2]);
    }else{
        max_duree = DUREE_MAX;
        max_essais = ESSAIS_MAX;
    }

    if(max_duree <= 0 || max_essais <= 0){
        max_duree = DUREE_MAX;
        max_essais = ESSAIS_MAX;
    }

    printf("Vous avez %d secondes pour entrer votre mot de passe \n", max_duree>

    action_nouv.sa_handler = signalHandler;
    CHECK(sigemptyset(&action_nouv.sa_mask),"sigemptyset()");
    action_nouv.sa_flags = 0;

    CHECK(sigaction(SIGINT, &action_nouv, NULL),"sigaction()");
```

(voir archive remis /Q18/Synthese.c)

Exemple de fonctionnement :