

Les signaux sous Linux

Mécanisme fondamental de **communication inter-processus**
Le SE communique avec les processus des utilisateurs en cas d'erreurs
(**violation mémoire, erreur** dans une entrée-sortie),
ou
à la demande de l'utilisateur lui-même
(caractères d'**interruption**).

Des fonctions d'émission de signaux asynchrones vers un processus donné,
et l'indication de la fonction à exécuter à la réception d'un signal donné.

Les signaux sont identifiés dans le système par un nombre entier.

La liste des signaux et le fichier `/usr/include/signal.h`

<code>#define SIGHUP</code>	<code>1</code>	<code>/* Hangup (POSIX). */</code>
<code>#define SIGINT</code>	<code>2</code>	<code>/* Interrupt (ANSI). */</code>
<code>#define SIGQUIT</code>	<code>3</code>	<code>/* Quit (POSIX). */</code>
<code>#define SIGILL</code>	<code>4</code>	<code>/* Illegal instruction (ANSI). */</code>
<code>#define SIGTRAP</code>	<code>5</code>	<code>/* Trace trap (POSIX). */</code>
<code>#define SIGABRT</code>	<code>6</code>	<code>/* Abort (ANSI). */</code>
<code>#define SIGIOT</code>	<code>6</code>	<code>/* IOT trap (4.2 BSD). */</code>
<code>#define SIGBUS</code>	<code>7</code>	<code>/* BUS error (4.2 BSD). */</code>
<code>#define SIGFPE</code>	<code>8</code>	<code>/* Floating-point exception (ANSI). */</code>
<code>#define SIGKILL</code>	<code>9</code>	<code>/* Kill, unblockable (POSIX). */</code>
<code>#define SIGUSR1</code>	<code>10</code>	<code>/* User-defined signal 1 (POSIX). */</code>
<code>#define SIGSEGV</code>	<code>11</code>	<code>/* Segmentation violation (ANSI). */</code>
<code>#define SIGUSR2</code>	<code>12</code>	<code>/* User-defined signal 2 (POSIX). */</code>
<code>#define SIGPIPE</code>	<code>13</code>	<code>/* Broken pipe (POSIX). */</code>
<code>#define SIGALRM</code>	<code>14</code>	<code>/* Alarm clock (POSIX). */</code>
<code>#define SIGTERM</code>	<code>15</code>	<code>/* Termination (ANSI). */</code>
<code>#define SIGSTKFLT</code>	<code>16</code>	<code>/* Stack fault. */</code>
<code>#define SIGCLD</code>	<code>SIGCHLD</code>	<code>/* Same as SIGCHLD (System V). */</code>

Les signaux sont identifiés dans le système par un nombre entier.
La liste des signaux et le fichier `/usr/include/signal.h`
Suite

<code>#define SIGCHLD</code>	17	<code>/* Child status has changed (POSIX). */</code>
<code>#define SIGCONT</code>	18	<code>/* Continue (POSIX). */</code>
<code>#define SIGSTOP</code>	19	<code>/* Stop, unblockable (POSIX). */</code>
<code>#define SIGTSTP</code>	20	<code>/* Keyboard stop (POSIX). */</code>
<code>#define SIGTTIN</code>	21	<code>/* Background read from tty (POSIX). */</code>
<code>#define SIGTTOU</code>	22	<code>/* Background write to tty (POSIX). */</code>
<code>#define SIGURG</code>	23	<code>/* Urgent condition on socket (4.2 BSD). */</code>
<code>#define SIGXCPU</code>	24	<code>/* CPU limit exceeded (4.2 BSD). */</code>
<code>#define SIGXFSZ</code>	25	<code>/* File size limit exceeded (4.2 BSD). */</code>
<code>#define SIGVTALRM</code>	26	<code>/* Virtual alarm clock (4.2 BSD). */</code>
<code>#define SIGPROF</code>	27	<code>/* Profiling alarm clock (4.2 BSD). */</code>
<code>#define SIGWINCH</code>	28	<code>/* Window size change (4.3 BSD, Sun). */</code>
<code>#define SIGPOLL</code>	SIGIO	<code>/* Pollable event occurred (System V). */</code>
<code>#define SIGIO</code>	29	<code>/* I/O now possible (4.2 BSD). */</code>
<code>#define SIGPWR</code>	30	<code>/* Power failure restart (System V). */</code>
<code>#define SIGUNUSED</code>	31	

Emission d'un signal

kill(pid , sig)

émet à destination du processus de numéro pid le signal de numéro sig
Si sig est nul, aucun signal n'est envoyé,

La valeur de retour de kill()
permet de savoir si pid est un numéro de processus ou non
(valeur de retour en cas d'opération réussie, et sinon).

Réception des signaux

La réception d'un signal par un processus



la **terminaison** de ce processus



la création d'un fichier **core** sur disque.

La réception d'un signal par un processus peut être provoquée par
une erreur de programme
une interruption de l'utilisateur depuis son terminal (SIGINT ou SIGQUIT)
un envoi par un autre processus utilisateur (primitive kill précédente).

Le programmeur peut définir le comportement des processus
à la réception des différents signaux (excepté pour le signal SIGKILL)

signal(int sig , int (*fonc)())

la fonction **fonc** spécifie le comportement est définie

SIG_DFL et SIG_IGN

permettent de spécifier respectivement un comportement standard
ou l'ignorance du signal

l'adresse d'une fonction

choisie et définie par l'utilisateur, qui sera alors exécutée à la réception du signal **sig**.

Exemples

ignorance des interruption clavier

```
#include <signal.h>
int main() {
    signal( SIGINT , SIG_IGN);
    signal( SIGQUIT , SIG_IGN);
    for( ;; );
    return 0;
}
```

\$ ignore &

[5] 1221

\$ ps

PID	TTY	TIME	CMD
583	pts/0	00:00:00	bash
1221	pts/0	00:00:01	ignore
1222	pts/0	00:00:00	ps

\$ kill -2 1221

\$ kill -3 1221

\$ kill -9 1221

\$

[5]+ Processus arrêté ignore

\$

Exemples

Restitution du comportement par défaut

```
#include <signal.h>
#define N 100000000
int main() {
    long int i;
    int j;
    signal(SIGINT, SIG_IGN);
    signal(SIGQUIT, SIG_IGN);

    printf("\nPhase non interruptible commencee.\n\n");
    for(i=0 , j=0 ; i<N ; i++) {
        if(i%(N/10)==0) printf("%d0%\n",++j);
    }
    printf("\nPhase non interruptible terminee.\n\n");
    signal(SIGINT,SIG_DFL);
    signal(SIGQUIT,SIG_DFL);
    for(i=0,j=0;i<N;i++) {
        if(i%(N/10) == 0) printf("%d0%\n",++j);
    }
    return 0;
}
```

Captage et déroutement

```
#include <signal.h>
char nom[30];
int message(int n) {
    printf("signal SIGUSR%d bien reçu\n",n);
}
int supp(int n) {
    unlink(nom);
    printf("n=%d\n", n);
    if(n==SIGQUIT) {
        printf("SIGQUIT reçu\n");
        exit(1);
    }
    else printf("SIGINT reçu\n");
}

int main() {
    int n;
    signal(SIGINT,supp); signal(SIGQUIT,supp);
    signal(SIGUSR1,message); signal(SIGUSR2,message);
    sprintf(nom, "/tmp/pp%d", getpid());
    n=creat(nom,0666);
    for(;;) ;
    return 0;
}
```


Restitution du comportement par défaut

```
#include <signal.h>
int message(int n) {
    signal(SIGUSR1,message);
    signal(SIGUSR2,message);
    printf("signal SIGUSR%d bien reçu\n",n);
}

int supp(int n) {
    unlink(nom);
    printf("n=%d\n",n);
    if(n==SIGQUIT) {
        printf("SIGQUIT reçu\n");
        exit(1);
    }
    else {
        printf("SIGINT reçu\n");
    }
}
```

Exemples

Captage de SIGPIPE

```
#include <signal.h>
void ecr( ) {
    printf("reception du signal SIGPIPE\n");
    exit(1);
}

int main() {
    int p[2];
    signal(SIGPIPE , ecr) ;
    pipe(p) ;
    close(p[0]) ;
    write(p[1] , "0" , 1) ;
    printf("ecriture de 1 caractere\n") ;
    return 0 ;
}
```

Héritage des signaux par fork()

Exemples

```
#include <signal.h>
void fin() {
    printf("SIGINT pour processus %d\n",getpid());
    exit(1);
}

int main() {
    signal(SIGQUIT,SIG_IGN);
    signal(SIGINT,fin);
    if(fork()>0) {
        printf("processus pere : %d\n",getpid());
        for(;;) ;
    }
    else {
        printf("processus fils : %d\n",getpid());
        for(;;) ;
    }
    return 0;
}
```

Transmission des signaux par **exec()**

Exemples

```
#include <signal.h>
```

```
int fin() {  
    printf("SIGINT pour processus %d\n",getpid());  
    exit(1);  
}
```

```
int main() {  
    signal(SIGQUIT , SIG_IGN);  
    signal(SIGINT , fin);  
    execl("boucle","boucle",0);  
    return 0;  
}
```

Utilisation de la valeur de la fonction **signal()**

Exemples

```
#include <signal.h>

int inter() {
    printf("SIGQUIT pour processus %d\n",getpid());
    exit(1);
}

int main() {
    signal(SIGQUIT , inter);
    signal(SIGINT , SIG_DFL);
    for(;;);
    return 0;
}
```

Exemples

Utilisation de la valeur de la fonction **signal()**

```
#include <signal.h>
```

```
int inter() {  
    printf("SIGQUIT pour processus %d\n",getpid());  
    exit(1);  
}
```

```
int main() {  
    /* Si la valeur de retour de l'appel signal et la fonction  
    SIG_IGN, c'est que le processus a été lancé en background. On force alors le processus à ignorer SIGQUIT  
    ; sinon on réalise la demande de déroutement. */  
    if(signal(SIGQUIT,inter) == SIG_IGN)  
        signal(SIGQUIT , SIG_IGN);  
    /* même traitement pour le signal SIGINT */  
    if(signal(SIGINT,SIG_DFL) == SIG_IGN)  
        signal(SIGINT , SIG_IGN);  
    for(;;);  
    return 0;  
}
```

Exemples

La fonction **pause()**

```
#include <signal.h>
```

```
inter() {  
    printf("signal reçu\n");  
}  
int main() {  
    int n;  
    signal(SIGINT,inter);  
    n=pause();  
    printf("valeur de pause : %d\n",n);  
    return 0;  
}
```

```
$ pause  
signal reçu  
valeur de pause : -1  
$
```

Exemples

```
#include <signal.h>
```

```
inter( ) { return(0) ; }
```

```
int main() {  
    int n ;  
    if(fork()) {  
        signal(SIGINT , inter);  
        n = wait(0) ;  
        printf("valeur de retour de wait : %d\n",n) ;  
    }  
    else {  
        signal(SIGINT , inter);  
        for(;;);  
    }  
    return 0;  
}
```


Exemples

```
#include <signal.h>
filsmort() {
    printf("père réveillé par mort d'un fils\n");
}

int main() {
    int n;
    if(fork()) {
        signal(SIGINT , SIG_IGN);
        signal(SIGCLD , filsmort);
        n=pause();
        printf("valeur de retour de pause : %d\n",n);
    }
    else for(;;);
    return 0;
}
```

```
$ pause2
père réveillé par mort d'un fils
valeur de retour de pause : -1
$
```

COMMUNICATION ENTRE DEUX PROCESSUS PAR ECHANGE DE SIGNAUX

```
#include <signal.h>
void Acknowledge(int sig) {
    signal(sig, Acknowledge);
}

int main(int argc, char* argv[]) {
    int pid;
    unsigned long value;
    printf("%u\n", getpid());
    if (scanf("%d", &pid) != 1) {
        return 1;
    }

    signal(SIGUSR1, Acknowledge);
    while (scanf("%lu", &value) == 1) {
        int i;
        unsigned long mask;
        mask = 1;
        for (i = 0 ; i < 8 * sizeof(unsigned long) ; ++i) {
            kill(pid, (value & mask) ? SIGUSR2 : SIGUSR1);
            mask <<= 1;
            pause();
        }
    }
}
```

COMMUNICATION ENTRE DEUX PROCESSUS PAR ECHANGE DE SIGNAUX

```
int i = 0; unsigned long mask = 1; unsigned long value = 0;
int pid;
```

```
void Bit0(int sig) {
    if (++i == 8 * sizeof(unsigned long)) {
        printf("%lu\n", value);
        fflush(stdout);
        i = 0; mask = 1; value = 0;
    } else {
        mask <<= 1;
    }
    kill(pid, SIGUSR1);
    signal(sig, Bit0);
}

int main(int argc, char* argv[]) {
    printf("%u\n", getpid());
    if (scanf("%d", &pid) != 1) {
        return 1;
    }
    signal(SIGUSR1, Bit0);
    signal(SIGUSR2, Bit1);
    for (;;) {
        pause();
    }
    return 0;
}
```

```
void Bit1(int sig) {
    if (++i == 8 * sizeof(unsigned long)) {
        printf("%lu\n", value);
        fflush(stdout);
        i = 0;
        mask = 1;
        value = 0;
    } else {
        value |= mask;
        mask <<= 1;
    }
    kill(pid, SIGUSR1);
    signal(sig, Bit1);
}
```

```
void handlerSignal(int sig) {  
    printf("\t\t[%d] handler%d => signal capté\n\n", getpid(), sig);  
}  
  
void sendSignal(int pid, int sig) {  
    sleep(1);  
    if (kill(pid, sig) == -1) {  
        printf("***Erreur kill *** PID = %d\n", pid); exit(1);  
    }  
    printf("#%d[%d] signal #%d envoyé à %d\n", getpid(), sig, pid);  
}
```

```

int Proc1, cptr, limite;
int main (int argc, char **argv) {
    if (argv[1] == NULL) {“ *** Erreur format *** \n”}; return 0; }
    limite = atoi(argv[1]);
    cptr = 0;
    Proc1 = fork();
    switch(Proc1) {
    case 0 :    signal(SIGUSR2,handlerSignal);
                printf(“Fils => signal armé, PID = %d\n\n”,getpid());
                pause();
                while(cptr < limite) {
                    cptr ++;
                    sendSignal(getppid(), SIGUSR1);
                    pause();
                }
                cptr++;
                sendSignal(getppid(SIGUSR1); exit(0);
                break;
    default :  signal(SIGUSR1,handlerSignal);
                printf(“Père => signal armé, PID = %d\n\n”,getpid());
                cptr++;
                sendSignal(Proc1, SIGUSR2);
                pause() ;
                while(cptr < limite) {
                    cptr ++;
                    sendSignal(Proc1, SIGUSR2);
                    pause();
                }
                break;
    }
}
return 0; }

```