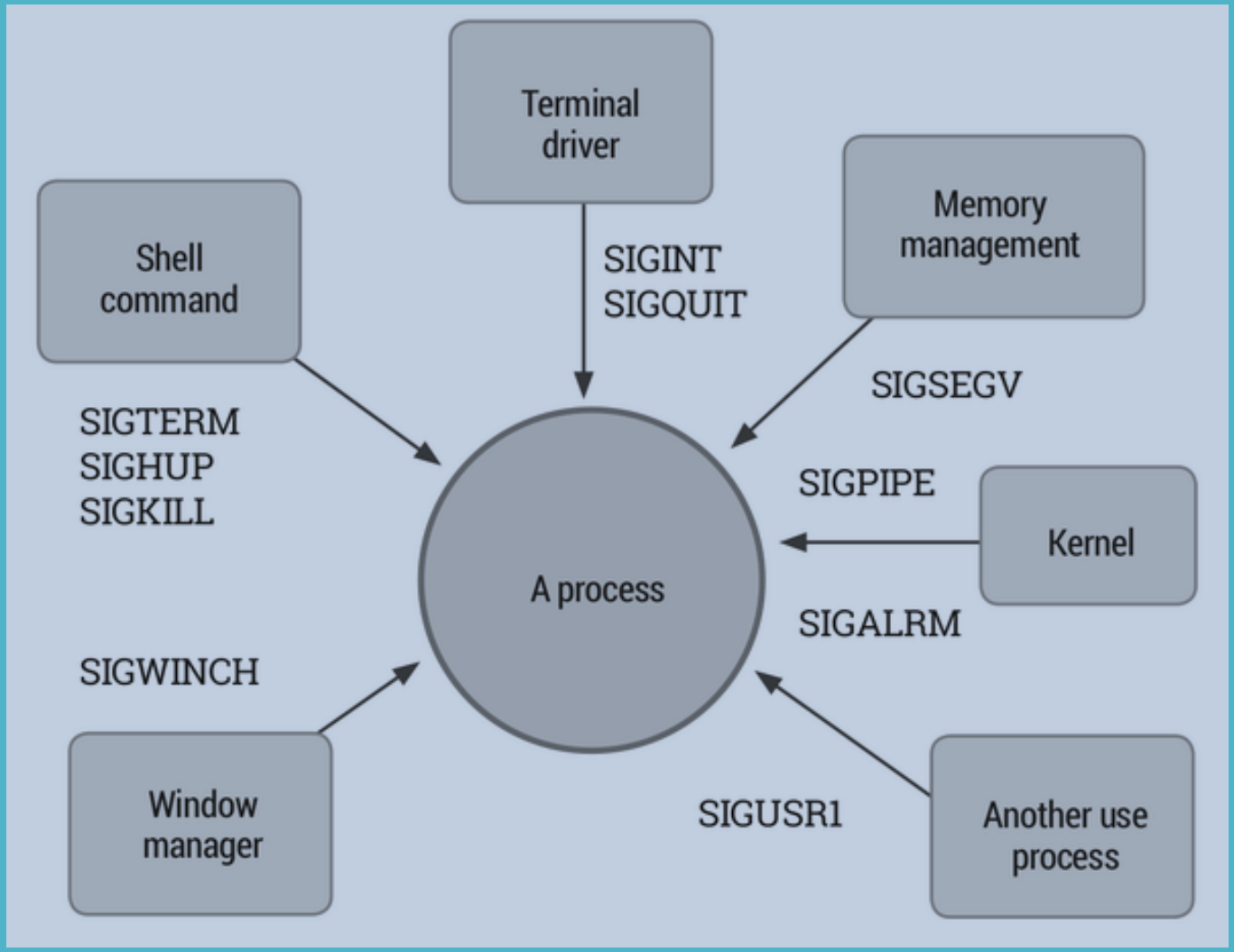
Σήματα


Terminal driver
SIGINT
SIGQUIT
Shell command
SIGTERM
SIGHUP
SIGKILL
Memory management
SIGSEGV
SIGPIPE
Kernel
A process
SIGALRM
SIGWINCH
Window manager
SIGUSR1
Another use process

Σήματα

Τα σήματα (signals) αποτελούν την απλούστερη μορφή επικοινωνίας ανάμεσα στις διεργασίες.

Επιτρέπουν τη διακοπή μιας διεργασίας από τον πυρήνα ή μία άλλη διεργασία με στόχο τη διαχείριση κάποιου συμβάντος. Μετά το χειρισμό του συμβάντος η διεργασία συνεχίζει τη λειτουργία της.

Τα σήματα είναι ασύγχρονα. Μπορούν να εμφανιστούν από τον οποιοδήποτε και ανά πάσα στιγμή.

Όταν μία διεργασία παραλάβει κάποιο σήμα, πραγματοποιεί κάποιο από τα ακόλουθα:

- Το αγνοεί.
- Ζητά από τον πυρήνα να συλλάβει (catch) το σήμα πριν επιτρέψει στη διεργασία να συνεχίσει.
- Αφήνει τον πυρήνα να εκτελέσει την προεπιλεγμένη απόκριση για το εν λόγω σήμα.

Ορισμός σήματος (signal.h)

Αρχικά ένα σήμα ορίζονταν από τη συνάρτηση signal (δεν χρησιμοποιείται σήμερα) που δηλώνεται ως

void * signal (int signum, void * handler);

όπου signum ένας αριθμός που ταυτοποιεί το σήμα και handler ένας δείκτης σε μία συνάρτηση χωρίς ορίσματα και επιστρεφόμενη τιμή η οποία εκτελείται από τον πυρήνα και η ολοκλήρωση της οποίας σηματοδοτεί τη συνέχιση της εκτέλεσης της διεργασίας από το σημείο που είχε σταματήσει.

Σήματα

Το σύνολο των διαθέσιμων σημάτων στο Linux εμφανίζεται με την εντολή kill –l η οποία επίσης χρησιμοποιείται και για την αποστολή τους.

```
amarg@amarg-vbox:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Τα σήματα με αριθμούς από 1 έως 31 είναι τα συνήθη σήματα του λειτουργικού συστήματος Linux.

Τα σήματα με αριθμούς από 32 έως 64 αποτελούν σήματα πραγματικού χρόνου και χρησιμοποιούνται από τα ομώνυμα συστήματα. Παρατηρήστε πως τα σήματα 32 και 33 ΔΕΝ εκτυπώνονται από την εντολή kill –l και ο λόγος για αυτό είναι πως χρησιμοποιούνται εσωτερικά από τη βιβλιοθήκη NPTL (Native POSIX Threads Library). Τα σήματα αυτά ορίζονται ως

SIGWAITING	32	Ignore All LWPs blocked
SIGLWP	33	Ignore Virtual Interprocessor Interrupt for Threads Library

και οι διαδικασίες που υλοποιούν ΔΕΝ υποστηρίζονται από το Linux.

Σήματα

Οι τιμές των σημάτων είναι δηλωμένες στο αρχείο signal.h ως ακέραιοι αριθμοί που ορίζονται με define
(οι παραπάνω τιμές αφορούν στο BSD UNIX ενώ για το Linux είναι αυτές που δίνει η kill -l)

```
#define     SIGHUP      1     /* hangup */
#define     SIGINT      2     /* interrupt */
#define     SIGQUIT     3     /* quit */
#define     SIGILL      4     /* illegal instruction (not reset when caught) */
#define     SIGTRAP     5     /* trace trap (not reset when caught) */
#define     SIGABRT     6     /* abort() */
#define     SIGEMT      7     /* EMT instruction */
#define     SIGFPE      8     /* floating point exception */
#define     SIGKILL     9     /* kill (cannot be caught or ignored) */
#define     SIGBUS      10    /* bus error */
#define     SIGSEGV     11    /* segmentation violation */
#define     SIGSYS      12    /* bad argument to system call */
#define     SIGPIPE     13    /* write on a pipe with no one to read it */
#define     SIGALRM     14    /* alarm clock */
#define     SIGTERM     15    /* software termination signal from kill */
#define     SIGURG      16    /* urgent condition on IO channel */
#define     SIGSTOP     17    /* sendable stop signal not from tty */
#define     SIGTSTP     18    /* stop signal from tty */
#define     SIGCONT     19    /* continue a stopped process */
#define     SIGCHLD     20    /* to parent on child stop or exit */
#define     SIGTTIN     21    /* to readers pgrp upon background tty read */
#define     SIGTTOU     22    /* like TTIN for output if (tp->t_local&LTOSTOP) */
#define     SIGIO       23    /* input/output possible signal */
#define     SIGXCPU     24    /* exceeded CPU time limit */
#define     SIGXFSZ     25    /* exceeded file size limit */
#define     SIGVTALRM   26    /* virtual time alarm */
#define     SIGPROF     27    /* profiling time alarm */
#define     SIGWINCH    28    /* window size changes */
#define     SIGINFO     29    /* information request */
#define     SIGUSR1     30    /* user defined signal 1 */
#define     SIGUSR2     31    /* user defined signal 2 */
```

Σήματα

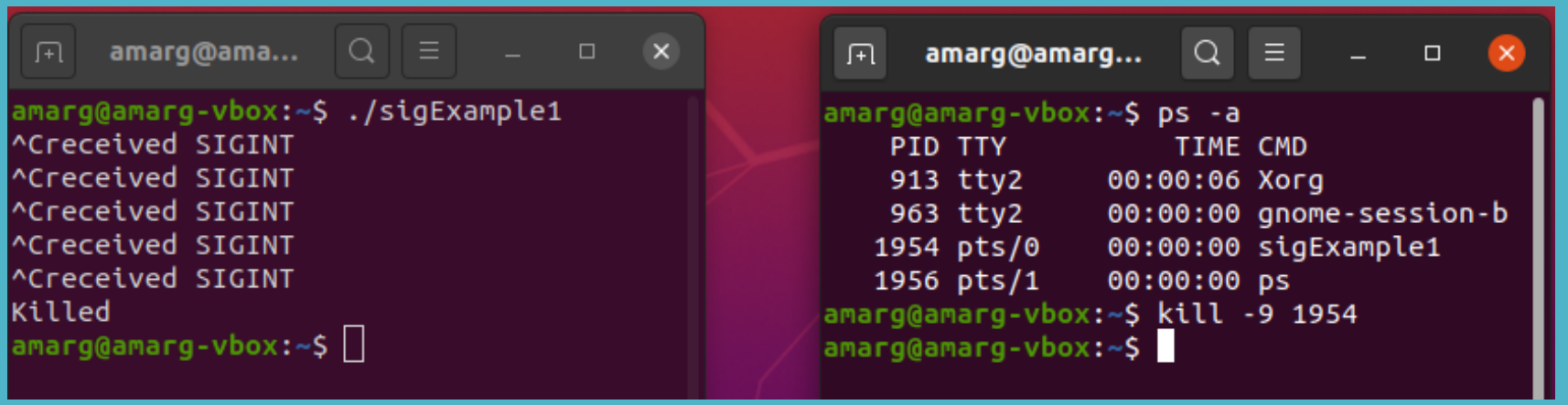
Παράδειγμα χρήσης της συνάρτησης signal – η διεργασία συλλαμβάνει το σήμα SIGINT

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>

void sig_handler(int signo) {
  if (signo == SIGINT)
    printf("received SIGINT\n"); }

int main(void) {
  if (signal(SIGINT, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGINT\n");
  // A long long wait so that we can easily
  // issue a signal to this process
  while(1)
    sleep(1);
  return 0; }
```

Η διεργασία ΔΕΝ τερματίζει με Ctrl – C αφού ο handler του σήματος SIGINT δεν καλεί την exit ή τη return. Για τον τερματισμό της διεργασίας ανοίγουμε ένα άλλο terminal, εκτελούμε την εντολή ps –a για να προσδιορίσουμε το pid της και την τερματίζουμε με την εντολή kill.



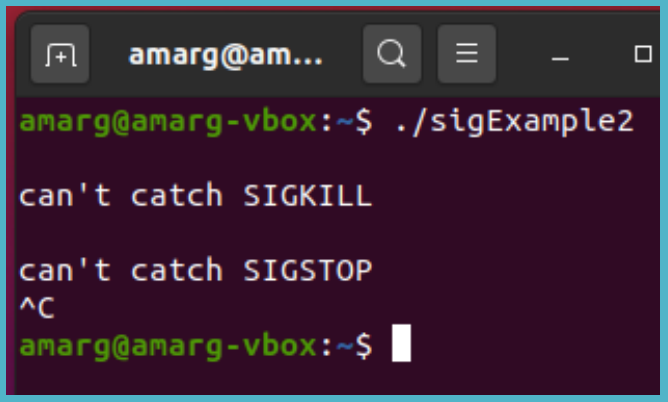
Σήματα

Τα σήματα SIGKILL και SIGSTOP δεν μπορούν να υποστούν χειρισμό από το χρήστη !!

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include <stdlib.h>

void sig_handler(int signo) {
  if (signo == SIGKILL)
    printf("received SIGKILL\n");
  if (signo == SIGSTOP)
    printf("received SIGSTOP\n"); }

int main(void) {
  if (signal(SIGKILL, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGKILL\n");
  if (signal(SIGSTOP, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGSTOP\n");
  // A long long wait so that we can easily
  // issue a signal to this process
  while(1)
    sleep(1);
  return 0; }
```



Ο χειρισμός του SIGINT αυτή τη φορά έγινε από τον πυρήνα και ως εκ τούτου η χρήση του συνδυασμού Ctrl-C οδήγησε στον τερματισμό της διεργασίας.

#define SIG_KERNEL_ONLY_MASK (rt_sigmask(SIGKILL) | rt_sigmask(SIGSTOP))

Ο λόγος για αυτό είναι τα εν λόγω σήματα επιτρέπουν τον δια της βίας τερματισμό (kill) μας διεργασίας η οποία έχει σταματήσει να αποκρίνεται. Εάν αναθέσουμε το χειρισμό αυτών των σημάτων στη διεργασία και η διεργασία σταματήσει να αποκρίνεται, δεν υπάρχει κανένας τρόπος για να τερματιστεί. Για το λόγο αυτό ο χειρισμός αυτών των σημάτων γίνεται από τον πυρήνα.

Σήματα

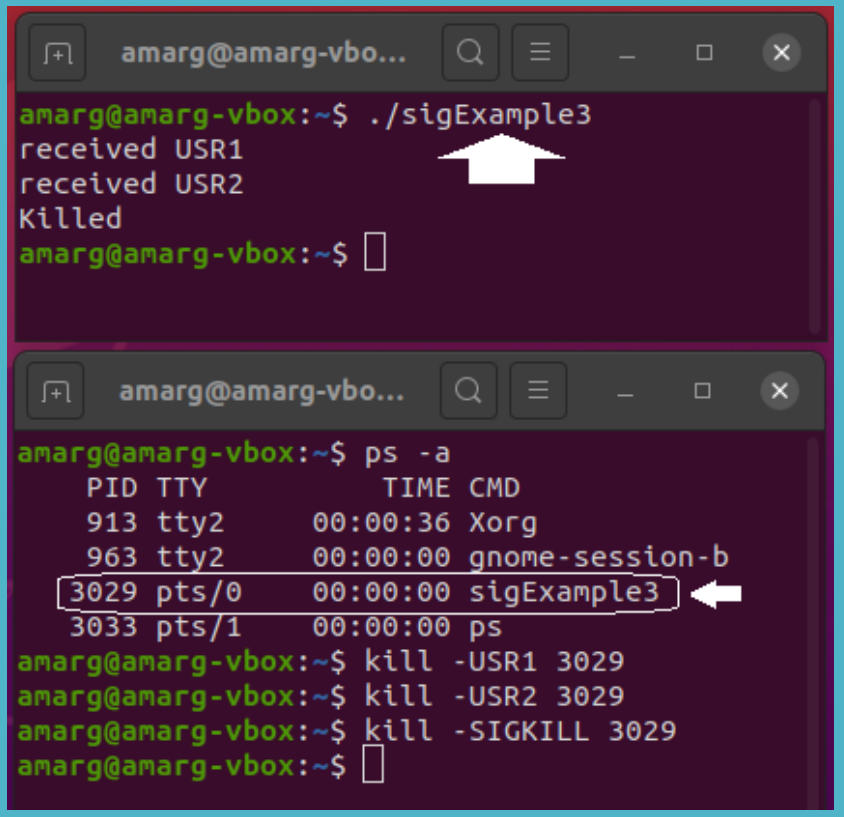
Τα σήματα γενικής χρήσεως SIGUSR1 και SIGUSR2

Το σύστημα προσφέρει τα σήματα SIGUSR1 και SIGUSR2 τα οποία είναι γενικής χρήσεως και μπορούν να χρησιμοποιηθούν σύμφωνα με τις ανάγκες του χρήστη, σε αντίθεση με όλα τα υπόλοιπα τα οποία είναι εντελώς συγκεκριμένα ως προς τον τρόπο λειτουργίας τους.

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include <stdlib.h>

void sig_handler(int signo) {
  if (signo == SIGUSR1)
    printf("received USR1\n");
  if (signo == SIGUSR2)
    printf("received USR2\n"); }

int main(void) {
  if (signal(SIGUSR1, sig_handler) == SIG_ERR)
      printf("\ncan't catch USR1\n");
  if (signal(SIGUSR2, sig_handler) == SIG_ERR)
      printf("\ncan't catch USR2\n");
  // A long long wait so that we can easily
  // issue a signal to this process
  while(1)
    sleep(1);
  return 0; }
```



Αυτά τα σήματα – όπως και όλα τα υπόλοιπα – στέλνονται χρησιμοποιώντας την εντολή kill.

Σήματα

Πώς συμπεριφέρεται ο πυρήνας όταν παραλάβει ένα σήμα:

- Το αγνοεί.
- Αναστέλλει την εκτέλεση της διεργασίας.
- Τερματίζει απότομα τη διεργασία
- Τερματίζει απότομα τη διεργασία, αποτυπώνοντας τον πυρήνα (core dump).

Τι συμβαίνει όταν μία διεργασία λάβει ένα σήμα του οποίου εκτελεί τον χειριστή?

Σύμφωνα με τα όσα αναφέραμε, αναστέλλει την εκτέλεσή της και καλεί εκ νέου το χειριστή. Αυτό είναι κάτι που απαιτεί προσεκτικό προγραμματισμό και τη χρήση συναρτήσεων επανεισόδου.

Ωστόσο, δεν λειτουργεί όταν χρησιμοποιούνται τεχνικές κλειδώματος, π.χ. κλείδωμα αρχείου.

ΛΥΣΗ → Δεν στέλνουμε στη διεργασία σήματα των οποίων ήδη εκτελεί το χειριστή. Ωστόσο, αυτό οδηγεί σε δυσκολία στο χειρισμό εάν στέλνονται πολλά σήματα με μεγάλη ταχύτητα και ενδέχεται να οδηγήσει σε υπερβολική αύξηση του μεγέθους της στοίβας !!!

POSIX → Στα POSIX λειτουργικά, η διεργασία αναμένει για την μεταβίβαση του δεύτερου σήματος μέχρι να ολοκληρωθεί ο χειρισμός του πρώτου σήματος. Τα σήματα που αναμένουν τη μεταβίβασή τους χαρακτηρίζονται ως εκκρεμή σήματα (pending signal).

Εάν σταλεί ένα σήμα ενώ ήδη υπάρχει ένα εκκρεμές στιγμιότυπό του, τα δύο σήματα συγχωνεύονται.

Σήματα

Ανάκτηση των πληροφοριών της διεργασίας όσον αφορά στο χειρισμό των σημάτων της

Οι πληροφορίες αυτού του τύπου μπορούν να βρεθούν στα πεδία SigBlk, SigIgn και SigCgt οι τιμές των οποίων αποθηκεύονται στο αρχείο /proc/[pid]/status όπου pid το process id της διεργασίας.

```
# cat /proc/1/status
...
SigBlk: 0000000000000000
SigIgn: fffffffe57f0d8fc
SigCgt: 0000000280b2603
...
```

SigBlk → ομάδα μπλοκαρισμένων σημάτων
SigIgn → ομάδα σημάτων που αγνοούνται
SigCgt → ομάδα σημάτων που έχουν συλληφθεί

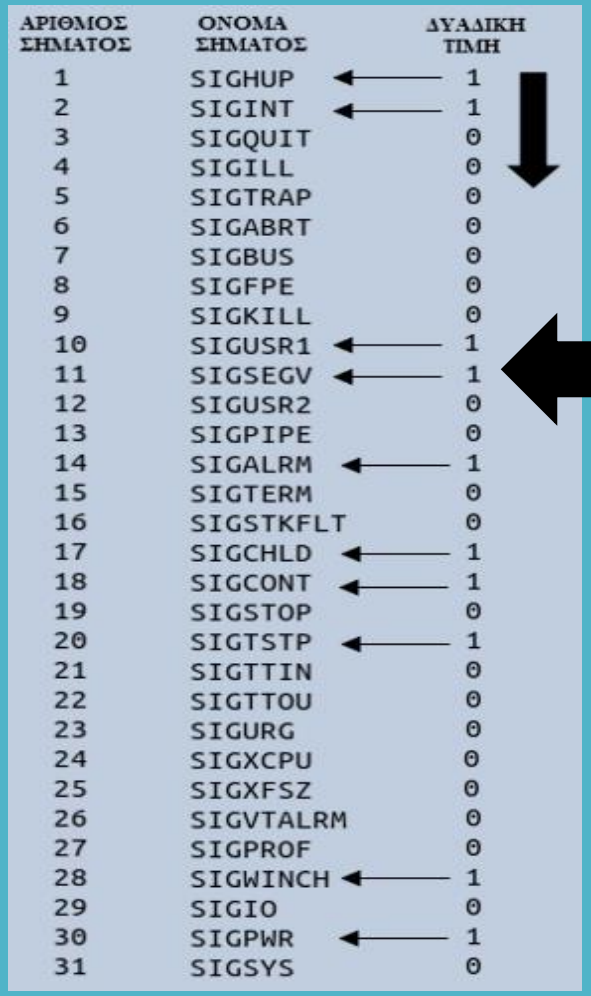
Ο δεκαεξαδικός αριθμος που βρίσκετα στα δεξιά της κάθε γραμμής αποτελεί μία μάσκα bit που εάν μετατραπεί στο δυαδικό σύστημα, δηλαδή σε ακολουθία από 0 και 1 περιέχει την πληροφορία σχετικά με το εάν το σήμα που αντιστοιχεί στο κάθε bit ικανοποιεί ή όχι την αντίστοιχη ιδιότητα.

00000000280b2603 ==> 0101000000010110010011000000011

Η μάσκα περιέχει 16 δεκαεξαδικούς αριθμούς και κατά συνέπεια αντιστοιχεί σε έναν δυαδικό αριθμό μήκους 64 bits. Από αυτά τα bits τα πρώτα 32 αντιστοιχούν στα standard signals ενώ τα υπόλοιπα 32 αντιστοιχούν στα real time signals του προτύπου POSIX.

Σήματα

Ανάκτηση των πληροφοριών της διεργασίας όσον αφορά στο χειρισμό των σημάτων της



ΑΡΙΘΜΟΣ ΣΗΜΑΤΟΣ	ΟΝΟΜΑ ΣΗΜΑΤΟΣ	ΔΥΑΔΙΚΗ ΤΙΜΗ
1	SIGHUP	1
2	SIGINT	1
3	SIGQUIT	0
4	SIGILL	0
5	SIGTRAP	0
6	SIGABRT	0
7	SIGBUS	0
8	SIGFPE	0
9	SIGKILL	0
10	SIGUSR1	1
11	SIGSEGV	1
12	SIGUSR2	0
13	SIGPIPE	0
14	SIGALRM	1
15	SIGTERM	0
16	SIGSTKFLT	0
17	SIGCHLD	1
18	SIGCONT	1
19	SIGSTOP	0
20	SIGTSTP	1
21	SIGTTIN	0
22	SIGTTOU	0
23	SIGURG	0
24	SIGXCPU	0
25	SIGXFSZ	0
26	SIGVTALRM	0
27	SIGPROF	0
28	SIGWINCH	1
29	SIGIO	0
30	SIGPWR	1
31	SIGSYS	0

Τα σήματα που αντιστοιχούν σε bit με τιμή 1 ικανοποιούν την ιδιότητα που σχετίζονται με τη μάσκα.

Στην προκειμένη περίπτωση η μάσκα είναι η

SigCgt → ομάδα σημάτων που έχουν συλληφθεί

και κατά συνέπεια τη χρονική στιγμή εξέτασης της μάσκας έχουν συλληφθεί τα σήματα

SIGHUP	SIGCHLD
SIGINT	SIGCONT
SIGUSR1	SIGSTP
SIGSEGV	SIGWINCH
SIGALARM	SIGTPWR

SigBlk → ομάδα μπλοκαρισμένων σημάτων
(όλα τα στοιχεία μηδέν → κανένα μπλοκαρισμένο σήμα)

Σήματα

Ανάκτηση των πληροφοριών της διεργασίας όσον αφορά στο χειρισμό των σημάτων της

```
SigIgn: fffffffe57f0d8fc
```

SigIgn → ομάδα σημάτων που αγνοούνται

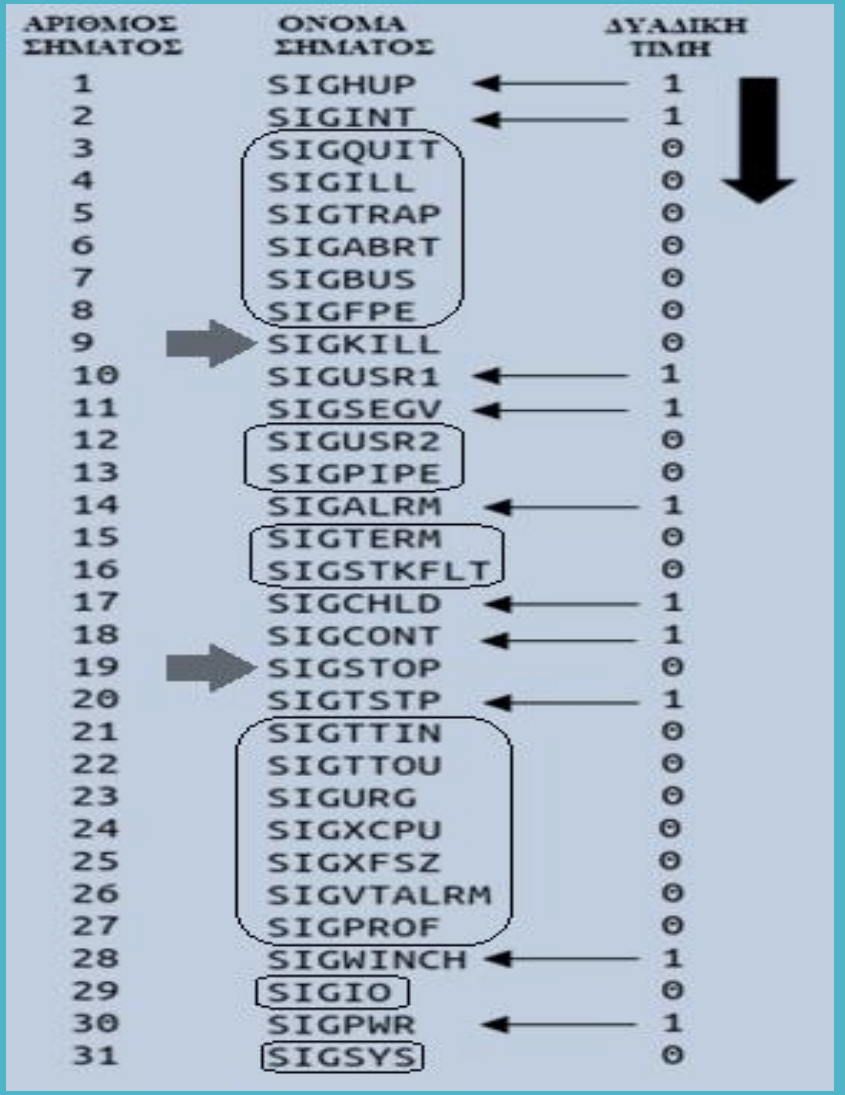
11111111111111111111111111111111

00101011111110000110110001111110 0

Τα πρώτα τριανταδύο 1 αντιστοιχούν στα σήματα πραγματικού χρόνου που όλα αγνοούνται

10101111111000011011000111111000

Το σύνολο των σημάτων που αγνοούνται είναι (προφανώς) ξένο του συνόλου των σημάτων που συλλαμβάνονται. Εάν πάρουμε την ένωση αυτών των δύο συνόλων και προσθέσουμε και τα δύο σήματα SIGKILL και SIGSTOP που υφίστανται χειρισμό MONO από τον πυρήνα καταλήγουμε στο σύνολο των 32 διαθέσιμων standard signals.



Σήματα

Παράδειγμα ανταλλαγής σήματος μεταξύ γονικής και θυγατρικής διεργασίας (Παράδειγμα Α)

```
void sighup() { // handler for SIGHUP
     signal(SIGHUP, sighup);
     printf("CHILD: I have received a SIGHUP\n"); }

void sigint() { // handler for SIGINT
     signal(SIGINT, sigint); /* reset signal */
     printf("CHILD: I have received a SIGINT\n"); }

void sigquit() { // handler for SIGQUIT
    printf("My DADDY has Killed me!!!\n");
    exit(0); }

void main() {
     int pid;
     signal(SIGHUP, sighup);
     signal(SIGINT, sigint);
     signal(SIGQUIT, sigquit);
     /* get child process */
     pid = fork ();
     if (pid < 0) {
          perror("fork");
          exit(1); }
     if (pid == 0) { for (;;);  }
     else {
        printf("\nPARENT: sending SIGHUP\n\n");
        kill(pid, SIGHUP);
        sleep(3); // pause for 3 secs
        printf("\nPARENT: sending SIGINT\n\n");
        kill(pid, SIGINT);
        sleep(3); // pause for 3 secs
        printf("\nPARENT: sending SIGQUIT\n\n");
        kill(pid, SIGQUIT);
        sleep(3); }}
```

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
```

```
amarg@amarg-vbox:~$ ./sigExample4

PARENT: sending SIGHUP

CHILD: I have received a SIGHUP

PARENT: sending SIGINT

CHILD: I have received a SIGINT

PARENT: sending SIGQUIT

My DADDY has Killed me!!!
```

Εδώ η διεργασία – γονέας στέλνει διαδοχικά στη θυγατρική διεργασία τα σήματα

SIGHUP
SIGINT
SIGQUIT

στα οποία αποκρίνεται καλώντας τον κατάλληλο χειριστή (sighup, sigint & sigquit)

Σήματα

Παράδειγμα ανταλλαγής σήματος μεταξύ γονικής και θυγατρικής διεργασίας (Παράδειγμα Β)

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <unistd.h>

// SIGALARM = 14
// SIGCHLD =  17

void signalHandler(int signal) {
     printf("Cought signal %d!\n",signal);
     if (signal==SIGCHLD) {
         printf("Child ended\n");
         wait(NULL); }}

int main() {
        signal(SIGALRM,signalHandler);
        signal(SIGUSR1,signalHandler);
        signal(SIGCHLD,signalHandler);
        if (!fork()) {
            printf("Child running...\n");
            sleep(2);
            printf("Child sending SIGALRM...\n");
            kill(getppid(),SIGALRM);
            sleep(5);
            printf("Child exitting...\n");
            return 0; }
        printf("Parent running, PID=%d. Press ENTER to exit.\n",getpid());
        getchar();
        printf("Parent exitting...\n");
        return 0; }
```

```
amarg@amarg-vbox:~$ ./sigExample5
Parent running, PID=1962. Press ENTER to exit.
Child running...
Child sending SIGALRM...
Cought signal 14!
Child exitting...
Cought signal 17!
Child ended

Parent exitting...
```

Εδώ η θυγατρική διεργασία στέλνει στη γονική διεργασία μόνο το σήμα SIGALARM.

Ωστόσο, το μήνυμα Caught signal 17! Σημαίνει πως η διεργασία έλαβε και το σήμα SIGCHLD το οποίο στάλθηκε αυτόματα από το σύστημα όταν ολοκληρώθηκε η θυγατρική διεργασία.

Σήματα

Παράδειγμα ανταλλαγής σήματος μεταξύ γονικής και πολλών θυγατρικών διεργασιών

```
#define NUMPROCS 4      /* number of processes to fork */
int nprocs;             /* number of child processes */

void child(int n) {
        printf("\tChild[%d]: child pid=%d, sleeping for %d seconds\n", n, getpid(), n);
        sleep(n);
        printf("\tchild[%d]: I'm exiting\n", n);
        exit(100+n); }

void catch(int snum) {
     int pid, status;
     pid = wait(&status);
     printf("Parent process: child process pid=%d exited with value
             pid, WEXITSTATUS(status));
     nprocs--; }

int main(int argc, char **argv) {

        int pid, i;
        signal(SIGCHLD, catch);
        for (i=0;i<NUMPROCS;i++) {
                pid=fork();
                if (pid<0) { perror("fork"); exit(1); }
                if (pid==0) child(i);
                else nprocs++; }
        printf("Parent process: going to sleep\n");
        while (nprocs != 0) {
                printf("parent: sleeping\n");
                sleep(60); }
        printf("Parent process: exiting\n");
        exit(0); }
```

```
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
```

Η γονική διεργασία δημιουργεί τέσσερις θυγατρικές διεργασίες και ανιχνεύει τον τερματισμό τους μέσω του σήματος SIGCHLD.

```
amarg@amarg-vbox:~$ ./sigExample6
Parent process: going to sleep
parent: sleeping
        Child[3]: child pid=1702, sleeping for 3 seconds
        Child[2]: child pid=1701, sleeping for 2 seconds
        Child[1]: child pid=1700, sleeping for 1 seconds
        Child[0]: child pid=1699, sleeping for 0 seconds
        child[0]: I'm exiting
Parent process: child process pid=1699 exited with value 100
parent: sleeping
        child[1]: I'm exiting
Parent process: child process pid=1700 exited with value 101
parent: sleeping
        child[2]: I'm exiting
Parent process: child process pid=1701 exited with value 102
parent: sleeping
        child[3]: I'm exiting
Parent process: child process pid=1702 exited with value 103
Parent process: exiting
```

Σήματα

Σύνολα σημάτων. Ένας πολύ χρήσιμος τύπος δεδομένων είναι ο τύπος sigset_t (signal.h) ο οποίος χρησιμοποιείται για την περιγραφή συνόλου σημάτων σαν και αυτά που ορίζονται μέσω της μάσκας.

Το πρότυπο POSIX προσφέρει πέντε συναρτήσεις χειρισμού συνόλων σημάτων.

int sigemptyset (sigset_t * set) → αδειάζει το σύνολο σημάτων το οποίο εκκενώνεται.

int sigfillset (sigset_t * set) → προσθέτει στο σύνολο σημάτων όλα τα διαθέσιμα σήματα.

int sigaddset (sigset_t * set, int signum) → προσθέτει στο σύνολο σημάτων set το σήμα με κωδικό signum.

int sigdelset (sigset_t * set, int signum) → αφαιρεί από το σύνολο σημάτων set το σήμα με κωδικό signum.

int sigismember (const sigset_t * set, int signum) → επιστρέφει τιμή true (μη μηδενική) ή false (μηδενική) ανάλογα με το εάν το σήμα signum ανήκει ή όχι στην ομάδα σημάτων set.

Οι δύο τρόποι αρχικοποίησης του sigset_t είναι να το εκκενώσουμε με την sigemptyset και μετά να προσθέσουμε τα σήματα που θέλουμε με την sigaddset ή να προσθέσουμε σε αυτό όλα τα διαθέσιμα σήματα με την sigfillset και μετά να απομακρύνουμε αυτά που δεν χρειάζονται με την sigdelset.

Σήματα

Αντί για την παρωχημένη συνάρτηση signal χρησιμοποιείται η sigaction που δηλώνεται ως

int sigaction (int signum, struct sigaction * act, struct sigaction * oact);

όπου signum ο αριθμός του σήματος και act ο χειριστής του σήματος.

Εάν είναι oact != NULL αυτή περιγράφει τις ρυθμίσεις του σήματος πριν την κλήση της sigaction.
Εάν είναι act = NULL η τρέχουσα ρύθμιση του σήματος παραμένει αμετάβλητη.

Η επιστρεφόμενη τιμή είναι μηδενική για επιτυχή εκτέλεση και αρνητική όταν εμφανίζεται σφάλμα.

Η δομή sigaction ορίζεται ως

```
struct sigaction {
        sighandler_t sa_handler;
        sigset_t sa_mask;
        unsigned long sa_flags;
        void (* sa_restorer) (void); }
```

Ο sa_handler είναι δείκτης σε μία συνάρτηση της μορφής

void handler (int signum);

ή εναλλακτικά να έχει μία από τις τιμές SIG_IGN (ignore) και SIG_DFL (do something).

Σήματα

Το πεδίο sa_flags περιέχει flags που επιτρέπουν στη διεργασία να τροποποιεί διαφορετικές συμπεριφορές σημάτων.

Το πεδίο sa_mask περιέχει τα σήματα που θα πρέπει να διακόπτονται όταν καλείται ο χειριστής του σήματος. Αυτά τα σήματα ΔΕΝ απορρίπτονται αλλά η μεταβίβασή τους καθυστερεί έως ότου η διεργασία είναι σε θέση να προχωρήσει στο χειρισμό τους.

Χρήσιμες συναρτήσεις διαχείρισης μάσκας

int sigpropmask (int what, const sigset_t * set, sigset_t * oldest);

Το όρισμα what καθορίζει τον τρόπο χειρισμού της μάσκας (εάν είναι set = NULL, αγνοείται) με τιμές:

SIG_BLOCK → τα σήματα του set προτίθενται στην τρέχουσα μάσκα σήματος.
SIG_UNBLOCK → τα σήματα του set διαγράφονται από την τρέχουσα μάσκα σήματος.
SIG_SETMASK → τα σήματα του set διακόπτωνται ενώ τα άλλα παραμένουν μη διακοπτώμενα.

Το oldest είναι η αρχική μάσκα και αγνοείται εάν έχει τιμή NULL. Η τρέχουσα μάσκα βρίσκεται ως

sigpropmask (SIG_BLOCK, NULL, &currentSet);

Σήματα

Η ανάκτηση της λίστας των εκκρεμών σημάτων γίνεται με τη συνάρτηση

int sigpending (sigset_t * set);

Η διαδικασία της αναμονής ενός σήματος πραγματοποιείται από τη συνάρτηση

int pause (void);

η οποία δεν επιστρέφει παρά μόνο όταν ένα σήμα μεταβιβαστεί στη διεργασία. Εναλλακτικά μπορούμε να καταφύγουμε στη χρήση της συνάρτησης

int sigsuspend (const sigset_t * mask)

Σήματα

Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος

```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

static int got_signal = 0;
static void hdl (int sig) { got_signal = 1; }

int main (int argc, char *argv[]) {
        sigset_t mask;
        sigset_t orig_mask;
        struct sigaction act;
        memset (&act, 0, sizeof(act));
        act.sa_handler = hdl;
        if (sigaction(SIGTERM, &act, 0)) {
                perror ("sigaction");
                return 1; }
        printf ("Emptying signal set...\n");
        sigemptyset (&mask);
        printf ("Adding SIGTERM to signal set...\n");
        sigaddset (&mask, SIGTERM);
        // SIGTERM signal is added to orig_mask
        printf ("Adding signal set to the original mask...\n");
        if (sigprocmask(SIG_BLOCK, &mask, &orig_mask) < 0) {
                perror ("sigprocmask");
                return 1; }
        // SIGTERM signal is blocked
        printf ("Blocking SIGTERM signal...\n");
        if (sigprocmask(SIG_SETMASK, &orig_mask, NULL) < 0) {
                perror ("sigprocmask");
                return 1; }
        printf ("Sleeping for 2 seconds\n");
        sleep (2);
        if (got_signal) puts ("Got signal");
        return 0; }
```

Σε αυτό το παράδειγμα, η διεργασία μπλοκάρει το σήμα SIGTERM για δύο δευτερόλεπτα χρησιμοποιώντας τη συνάρτηση sigprocmask. Μετά την παρέλευση των δύο δευτερολέπτων το σήμα ξεμπλοκάρεται.

Χρησιμοποιούνται οι συναρτήσεις

- sigaction
- sigsmptyset
- sigaddset
- sigprocmask

```
amarg@amarg-vbox:~$ ./sigExample7
Emptying signal set...
Adding SIGTERM to signal set...
Adding signal set to the original mask...
Blocking SIGTERM signal...
Sleeping for 2 seconds
amarg@amarg-vbox:~$ gedit sigExample7.c
```

Σήματα

Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος

Η εφαρμογή δέχεται τα σήματα SIGHUP, SIGUSR1 και SIGINT που δίνει ο χρήστης από άλλο terminal

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void handle_signal(int signal);
void handle_sigalrm(int signal);
void do_sleep(int seconds);

int main() {
    struct sigaction sa;
    printf("My pid is: %d\n", getpid()); // we need the pid to use kill from another terminal
    sa.sa_handler = &handle_signal; // here we assign the signal handler
    sa.sa_flags = SA_RESTART;
    // Block every signal during the handler
    sigfillset(&sa.sa_mask);
    // Signals SIGHUP, SIGUSR1 and SIGINT are associated with struct sa
    if (sigaction(SIGHUP, &sa, NULL) == -1) { perror("Error: cannot handle SIGHUP"); }
    if (sigaction(SIGUSR1, &sa, NULL) == -1) { perror("Error: cannot handle SIGUSR1"); }
    if (sigaction(SIGINT, &sa, NULL) == -1) { perror("Error: cannot handle SIGINT"); }
    // Will always fail, SIGKILL is intended to force kill your process
    if (sigaction(SIGKILL, &sa, NULL) == -1) { perror("Cannot handle SIGKILL"); }
    for (;;) {
        printf("\nSleeping for ~3 seconds\n");
        do_sleep(3); }}
```

Το κυρίως πρόγραμμα: τα τρία σήματα ενδιαφέροντος συσχετίζονται με τον signal handler και στη συνέχεια το πρόγραμμα μπαίνει σε ένα infinite loop καλώντας επαναληπτικά τη συνάρτηση do_sleep.

Σήματα

Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος

```
void handle_signal (int signal) {
    const char *signal_name;
    sigset_t pending;
    // Find out which signal we're handling
    switch (signal) {
        case SIGHUP:
            signal_name = "SIGHUP";
            break;
        case SIGUSR1:
            signal_name = "SIGUSR1";
            break;
        case SIGINT:
            printf("Caught SIGINT, exiting now\n");
            exit(0);
        default:
            fprintf(stderr, "Caught wrong signal: %d\n", signal);
            return; }
    // However, printf in signal handlers IS NOT reccommented !
    printf("Caught %s, sleeping for ~3 seconds\n"
           "Try sending another SIGHUP / SIGINT / SIGALRM "
           "(or more) meanwhile\n", signal_name);
    do_sleep(3);
    printf("Done sleeping for %s\n", signal_name);
    // So what did you send me while I was asleep?
    sigpending(&pending);
    if (sigismember(&pending, SIGHUP)) { printf("A SIGHUP is waiting\n"); }
    if (sigismember(&pending, SIGUSR1)) { printf("A SIGUSR1 is waiting\n"); }
    printf("Done handling %s\n\n", signal_name); }
```

Ο signal handler εκτυπώνει ενημερωτικές πληροφορίες, καλεί την do_sleep και εκτυπώνει τα ονόματα των σημάτων που εκκρεμούν

Σήματα

Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος

```
void handle_sigalrm(int signal) {
    if (signal != SIGALRM) { fprintf(stderr, "Caught wrong signal: %d\n", signal); }
    printf("Got sigalrm, do_sleep() will end\n"); }

void do_sleep(int seconds) {
    struct sigaction sa;
    sigset_t mask;
    sa.sa_handler = &handle_sigalrm; // Intercept and ignore SIGALRM
    sa.sa_flags = SA_RESETHAND; // Remove the handler after first signal
    sigfillset(&sa.sa_mask);
    sigaction(SIGALRM, &sa, NULL);
    // Get the current signal mask
    sigprocmask(0, NULL, &mask);
    // Unblock SIGALRM
    sigdelset(&mask, SIGALRM);
    // Wait with this mask
    alarm(seconds);
    sigsuspend(&mask);
    printf("sigsuspend() returned\n"); }
```

Η handle_sigalarm εκτυπώνει απλά ένα ενημερωτικό μήνυμα σχετικά με το σήμα που παρέλαβε. Η do_sleep συσχετίζει το SIGALARM με τη δομή sa που αρχικοποιείται κατάλληλα. Ανακτάται η τρέχουσα μάσκα, αφαιρείται από αυτή το SIGALARM και στη συνέχεια στέλνεται αυτό σήμα ύστερα από seconds δευτερόλεπτα.

Σήματα

Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος