Αποστολή/παραλαβή σημάτων

- Από τον πυρήνα του Unix σε μία διεργασία, σε περίπτωση εξαιρετικού γεγονότος (λάθος κινητής υποδιαστολής, αντικανονική εντολή κ.λ.π.)
- Από το πληκτρολόγιο σε μία διεργασία για οριστική (*Control-C*) ή προσωρινή (*Control-Z*) διακοπή κ.λ.π.
- Από μία διεργασία σε μία άλλη διεργασία
- Η διεργασία που παραλαμβάνει ένα σήμα μπορεί είτε να το αγνοήσει είτε να διακόψει προσωρινά τη λειτουργία της για να εκτελέσει ένα τμήμα ειδικού κώδικα που ονομάζεται διαχειριστής του σήματος

- Στο BSD Unix υπάρχουν 31 διαφορετικά σήματα, κάθε ένα από τα οποία χαρακτηρίζεται από έναν αριθμό (μεταξύ 1 και 31) ή από ένα ισοδύναμο συμβολικό όνομα και έχει έναν προκαθορισμένο από το Unix διαχειριστή σήματος, ο οποίος στις περισσότερες περιπτώσεις μπορεί να αλλάξει

- Σε προγράμματα που χρησιμοποιούν συμβολικά ονόματα σημάτων απαιτείται: `#include <signal.h>`

- Μερικά σήματα

`SIGINT`	(2)	interrupt
`SIGKILL`	(9)	kill
`SIGALRM`	(14)	alarm clock
`SIGTERM`	(15)	software termination signal
`SIGSTOP`	(17)	stop
`SIGTSTP`	(18)	stop signal from keyboard
`SIGCONT`	(19)	continue after stop
`SIGUSR1`	(30)	user-defined signal 1
`SIGUSR2`	(31)	user-defined signal 2

- Συνάρτηση βιβλιοθήκης **alarm**
  - unsigned int **alarm**(unsigned int *count*)
  - Δίνει εντολή στον πυρήνα να στείλει μετά από *count* δευτερόλεπτα το σήμα SIGALRM στην καλούσα διεργασία
  - Ο προκαθορισμένος διαχειριστής σήματος εκτυπώνει ένα μήνυμα και τερματίζει τη διεργασία
- Χρήση της συνάρτησης **alarm**

```
/* File: alarm_demo.c */
#include <stdio.h>   /* For printf */
main()
{  alarm(3);         /* Schedule an alarm signal
                        in three seconds */
   printf("Looping forever...\n");
   while (1);
   printf("This line should never be executed\n"); }
```

```
% date ; alarm_demo ; date
Tue Nov 16 11:28:08 WET 1993
Looping forever...
Alarm clock
Tue Nov 16 11:28:11 WET 1993
%
```

- Κλήση συστήματος **signal**
  - void (***signal**(int *sigCode*, void (**func*)( ))( )
  - Ορίζει την αντίδραση της καλούσας διεργασίας για το σήμα *sigCode*
  - Το δεύτερο όρισμα μπορεί να είναι `SIG_IGN`, οπότε το σήμα αγνοείται, `SIG_DFL`, που σημαίνει ότι ο προκαθορισμένος από το Unix διαχειριστής σήματος πρέπει να χρησιμοποιηθεί, ή η διεύθυνση μίας συνάρτησης που ορίζεται από τον προγραμματιστή και παίζει το ρόλο του διαχειριστή σήματος
  - Τα σήματα `SIGKILL` και `SIGSTOP` δεν είναι δυνατόν να αγνοηθούν ούτε μπορεί να αλλάξει γιά αυτά ο διαχειριστής τους
  - Ό,τι ισχύει σε σχέση με τα σήματα για μία διεργασία κληρονομείται στις διεργασίες-παιδιά που δημιουργεί μέσω **fork**
  - Η **signal** επιστρέφει τον προηγούμενο διαχειριστή σήματος σε επιτυχία ή -1 σε αποτυχία

- Χρήση της κλήσης **signal**

```
/* File: critical.c */
#include <stdio.h>   /* For printf */
#include <signal.h>  /* For SIGINT, SIG_IGN */
main()
{  void (*oldHandler)();   /* To hold old handler value */
   printf("I can be Control-C'ed\n");
   sleep(3);
   oldHandler = signal(SIGINT, SIG_IGN);  /* Ignore ^C */
   printf("I am protected from Control-C now\n");
   sleep(3);
   signal(SIGINT, oldHandler);  /* Restore old handler */
   printf("I can be Control-C'ed again\n");
   sleep(3);
   printf("Bye!\n"); }
```

```
% critical
I can be Control-C'ed
^C%
% critical
I can be Control-C'ed
I am protected from Control-C now
^CI can be Control-C'ed again
Bye!
%
```

- Κλήση συστήματος **pause**
  - int **pause**( )
  - Θέτει την καλούσα διεργασία σε αναμονή μέχρις ότου ληφθεί κάποιο σήμα
- Χρήση της κλήσης **pause**

```
/* File: new_handler.c */
#include <stdio.h>      /* For printf */
#include <signal.h>     /* For SIGALRM */
int alarmFlag = 0;      /* Global alarm flag */
void alarmHandler();    /* Forward declaration */

main()
{  signal(SIGALRM, alarmHandler); /* Install signal handler */
   alarm(3);   /* Schedule an alarm signal in three seconds */
   printf("Looping...\n");
   while (!alarmFlag)                /* Loop until flag set */
      pause();                         /* Wait for a signal */
   printf("Loop ends due to alarm signal\n"); }

void alarmHandler()
{  printf("An alarm clock signal was received\n");
   alarmFlag = 1; }             /* Set flag to stop looping */
```

```
% date ; new_handler ; date
Tue Nov 16 11:54:29 WET 1993
Looping...
An alarm clock signal was received
Loop ends due to alarm signal
Tue Nov 16 11:54:32 WET 1993
%
```

- Κλήση συστήματος **kill**
  - int **kill**(int *pid*, int *sigCode*)
  - Στέλνει το σήμα *sigCode* στη διεργασία με ταυτότητα *pid*
  - Είναι επιτυχής όταν η αποστέλλουσα διεργασία και η παραλαμβάνουσα διεργασία έχουν τον ίδιο ιδιοκτήτη ή όταν η αποστέλλουσα διεργασία ανήκει στο διαχειριστή του συστήματος
  - Επιστρέφει 0 σε επιτυχία ή -1 σε αποτυχία

- Χρήση της κλήσης **kill**

```
/* File: pulse.c */
#include <stdio.h>   /* For printf */
#include <signal.h>  /* For SIGTERM, SIGSTOP, SIGCONT */
main()
{  int pid1, pid2;
   if ((pid1 = fork()) == -1) {  /* Check for error */
      perror("fork"); exit(1); }
   if (pid1 == 0)          /* First child */
      while (1) {          /* Infinite loop */
         printf("Process 1 is alive\n"); sleep(1); }
   if ((pid2 = fork()) == -1) {  /* Check for error */
      perror("fork"); exit(1); }
   if (pid2 == 0)          /* Second child */
      while (1) {          /* Infinite loop */
         printf("Process 2 is alive\n"); sleep(1); }
   sleep(2); kill(pid1, SIGSTOP);   /* Suspend first child */
   sleep(2); kill(pid1, SIGCONT);   /* Resume first child */
   sleep(2);
   kill(pid1, SIGTERM);             /* Terminate first child */
   kill(pid2, SIGTERM); }           /* Terminate second child */
```

```
% pulse
Process 2 is alive
Process 1 is alive
Process 2 is alive
Process 1 is alive
Process 2 is alive
Process 2 is alive
Process 1 is alive
Process 2 is alive
Process 1 is alive
Process 2 is alive
%
```

```
% timeout
Usage: timeout [-10] command
% timeout -30
Usage: timeout [-10] command
% timeout blabla
execvp: No such file or directory
Command execution finished with exit code 1
% timeout -20 ps
  PID TT STAT   TIME COMMAND
 4803 p1 S      0:01 -csh (csh)
 8696 p1 S      0:00 timeout -20 ps
 8697 p1 R      0:00 ps
Command execution finished with exit code 0
% date ; timeout -12 sleep 15 ; date
Tue Nov 16 12:44:27 WET 1993
Command was killed by signal 9
Tue Nov 16 12:44:39 WET 1993
% date ; timeout sleep 7 ; date
Tue Nov 16 12:45:04 WET 1993
Command execution finished with exit code 0
Tue Nov 16 12:45:11 WET 1993
%
```

- *Να γραφεί ένα πρόγραμμα C το οποίο, σαν διεργασία, όταν δέχεται ένα σήμα (για παράδειγμα το SIGUSR1) να δημιουργεί ένα αντίγραφο του εαυτού του (διεργασία-παιδί) και μετά να τερματίζει.*

```
/* File: persistent.c */
#include <stdio.h>       /* For printf */
#include <signal.h>      /* For SIGUSR1 */
int loop;
void sigusr1_handler();  /* Forward declaration */

main()
{  int pid;
   signal(SIGUSR1, sigusr1_handler); /* Install handler */
   while(1) {                        /* Loop forever */
      printf("I am process %d and I am going to loop\n",
             getpid());
      loop = 1;
      while(loop) {                 /* Loop until loop=0 */
         sleep(1); }
      if ((pid = fork()) == -1) {    /* Check for error */
         perror("fork"); exit(1); }
      if (pid == 0)
         continue;                   /* Child process */
      else
         exit(0); } }                /* Parent process */

void sigusr1_handler()
{  printf("SIGUSR1 received\n");
   loop = 0; }
```

```
% persistent &
[1] 16026
% I am process 16026 and I am going to loop
kill -30 16026
SIGUSR1 received
%
[1]    Done                  persistent
I am process 16027 and I am going to loop
kill -30 16027
SIGUSR1 received
% I am process 16028 and I am going to loop
kill -30 16028
SIGUSR1 received
% I am process 16029 and I am going to loop
kill -30 16029
SIGUSR1 received
% I am process 16030 and I am going to loop
kill -9 16030
%
```