Esempio (**`signal`**)

```
/* sigex --- shows how signal works */
#include <signal.h>


/* trivial function to handle SIGINT */
int catchint(signo)

int signo;
{
printf("\nCATCHINT: signo=%d\n",signo);
printf("CATCHINT: returning\n\n");

}



main()
{
int catchint();
\*
* Before signal call, SIGINT will terminate process
*/

signal(SIGINT,catchint);

/*
* After signal call control will be passed to
* catchint, when SIGINT received
*/

printf("Sleep call #1\n");
sleep(1);

printf("Sleep call #2\n");
sleep(1);

printf("Sleep call #3\n");
sleep(1);

printf("Sleep call #4\n");
sleep(1);

printf("Exiting\n");
exit(0);
}
```

Esempi

Se il programma è lasciato "da solo" si ha:

```
$ sigex
Sleep call #1
Sleep call #2
Sleep call #3
Sleep call #4
Exiting
```

Se l'utente digita CTRL-C si ottiene:

```
$ sigex
Sleep call #1
^C

CATCHINT: signo=2
CATCHINT: returning


Sleep call #2
Sleep call #3
Sleep call #4
Exiting
```

E' possibile ignorare il segnale con:

```
signal(SIGINT,SIG_IGN)
```

o ripristinare il comportamento di default:

```
signal(SIGINT,SIG_DFL)
```

L'azione associata alla maggior parte dei segnali è "resettata" immediatamente dopo essere stati catturati.

Esempio

```
#include <signal.h>

int interrupt()
{
printf("Interrupt called\n");
sleep(10);
}

main()
{
signal(SIGINT,interrupt);
printf("Interrupt set for SIGINT\n");
sleep(10);
}
```

Esempi di esecuzione:

```
$reset
Interrupt set for SIGINT
^C                              (l'utente digita ^D)
Interrupt called
^C                              (... una seconda volta)
$
```

Per risolvere il problema si può modificare la funzione come segue:

```
int interrupt()
{
signal(SIGINT,SIG_IGN);
printf("Interrupt called\n");
.
.
signal(SIGINT,interrupt);
}
```

Esempio (`kill`)

```
/* synchro --- example for "kill" */

#include <signal.h>

int ntimes = 0;

int p_action()
{
printf("Parent caught signal#%d\n", ++ntimes);
signal(SIGUSR1,p_action);
}

int c_action()
{
printf("Child caught signal#%d\n", ++ntimes);
signal(SIGUSR1,c_action);
}

main()
{
int pid, ppid;

/* set SIGUSR1 action for parent */
signal(SIGUSR1,p_action);

switch( pid=fork() ) {

  case -1:   /* error */
    error("...");
    exit(1);

  case 0:    /* child */
    /* set action for child */
    signal(SIGUSR1,c_action);

    /* get parent PID */
    ppid = getppid();

    for(;;) {
      sleep(1);
      kill(ppid,SIGUSR1);
      pause();
      }
```

```
  default:    /* parent */
    for(;;) {
      pause();
      sleep(1);
      kill(pid,SIGUSR1);
      }

  }
} /* end main */
```

```
=-=-=-=-=-= Esecuzione =-=-=-=-=-=
```

```
$ synchro
Parent caught signal #1
Child caught signal #1
Parent caught signal #2
Child caught signal #2
^D
$
```

Esempio (**`pause & alarm`**)

Scrivere un programma che stampi un messaggio dopo un numero specificato di minuti.

```
/* tml --- tell-me-later program */

#include <stdio.h>
#include <signal.h>

#define TRUE       1
#define FALSE      0
#define BELLS      "\007\007\007"


int alarm_flag = FALSE;

/* routine to handle SIGALARM */
setflag()
{
alarm_flag = TRUE;
}

main(argc,argv)

int argc;
int *argv[];

{
int nsecs, pid;
int j;

if(argc <=2){
  fprintf(stderr, "Usage: tml #minutes message\n");
  exit(1);
  }

/* translate 1st argument into nsecs (int) */
nsecs = atoi(argv[1])*60;
```

```
/* fork to creat background process */
switch(pid=fork())  {
  case -1:  /* error */
    error("...");
    exit(1);
  case 0:   /* child */
    break;
  default:  /* parent */
    printf("tml process-id %d\n",pid);
    exit(0);
  }


/* set action for alarm */
signal(SIGALARM,setflag);

/* turn on alarm clock */
alarm(nsecs);

/* pause until signal... */
pause();

/* if signal was SIGALARM, print message */
if(alarm_flag == TRUE) {
  print(BELLS);
  for(j=2; j < argc; j++)
    printf("%s ", arg[j]);
  printf("\n");
  }

exit (0);

}  /* end main */
```