Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Segnali

Amos Brocco, Ricercatore, DTI / ISIN

15 marzo 2012

Cosa sono i segnali?

- Sono semplici messaggi che possono essere generati sia da un processo che da una thread e sono indirizzati a un'altra thread o a un altro processo.

- I segnali possono anche essere generati dal sistema operativo stesso

Perché utilizzare dei segnali

- I processi possono inviare dei segnali ad altri processi
  - per fermarli (SIGSTOP)
  - per farli continuare (SIGCONT)
  - per interromperli (SIGINT)
  - ...
- I segnali sono eventi asincroni:
  - Ogni segnale ha un identificatore (es. SIGKILL)
  - Un programma può definire una funzione che viene richiamata per gestire i segnali ricevuti durante la sua esecuzione

Esempi di segnali Unix

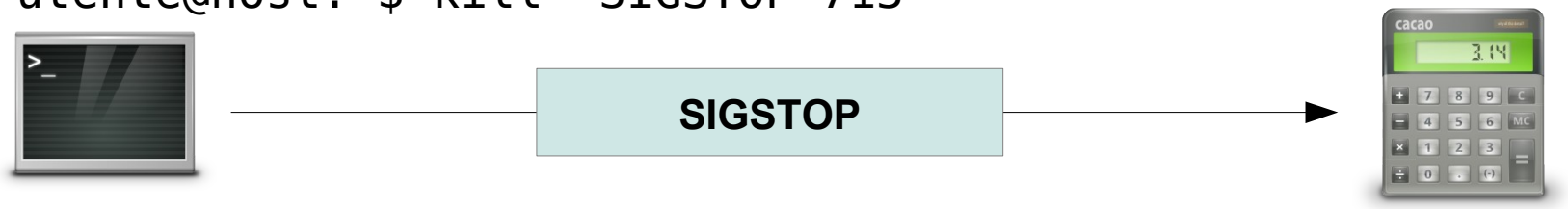
Segnale	Evento scatenante	Reazione di difetto
SIGABRT	Funzione `abort()`	Fine del processo e *core dump*
SIGALRM	Funzione `alarm()`	Fine del processo
SIGKILL[1]	Azione di *kill*	Fine del processo
SIGTERM	Uscita anormale del processo	Fine del processo
SIGCHLD[2]	Stop/uscita del processo discendente	
SIGFPE	Eccezione aritmetica	Fine del processo e *core dump*
SIGILL	Istruzione non valida	Fine del processo e *core dump*
SIGSEGV[6]	Riferimento di memoria non valido	Fine del processo e *coredump*
SIGINT	Carattere di controllo `VINTR`	Fine del processo
SIGQUIT	Carattere di controllo `QUIT`	Fine del processo e *core dump*
SIGPIPE	Scrittura in una pipe senza lettore	Fine del processo
SIGHUP	Evento di *Hangup*	Fine del processo
SIGSTOP[1]	Azione di *stop*	Il processo è sospeso (*stopped*)
SIGTSTP[3]	Caratteri di controllo `VSUSP`	Il processo è sospeso (*stopped*)
SIGTTIN	Azione di *input* per processo in *background*	
SIGTTOU	Azione di *output* per processo in *background*	
SIGCONT[4]	Continuazione dallo stato di *stop*	
SIGUSR1	Invio esplicito	Fine del processo
SIGUSR2	Invio esplicito	Fine del processo
SIGRTMIN[5]	Invio esplicito	Fine del processo
...		
SIGRTMAX	Invio esplicito	
SIGURG[7]	input/output, dati urgenti	

Segnali real time

Esempio

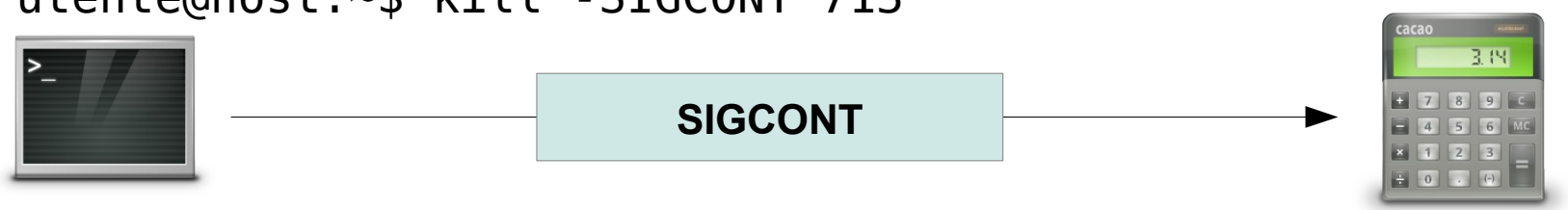
Con il comando kill possiamo generare dei segnali

```
utente@host:~$ kill -SIGSTOP 713
```



Il processo gnome-calculator (PID = 713) è fermato

```
utente@host:~$ kill -SIGCONT 713
```



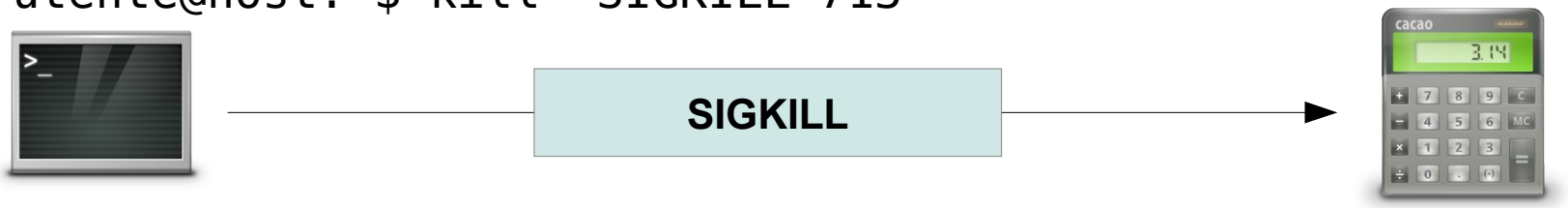
Il processo gnome-calculator (PID = 713) può continuare l'esecuzione

Esempio

```
utente@host:~$ kill -9 713
```

oppure

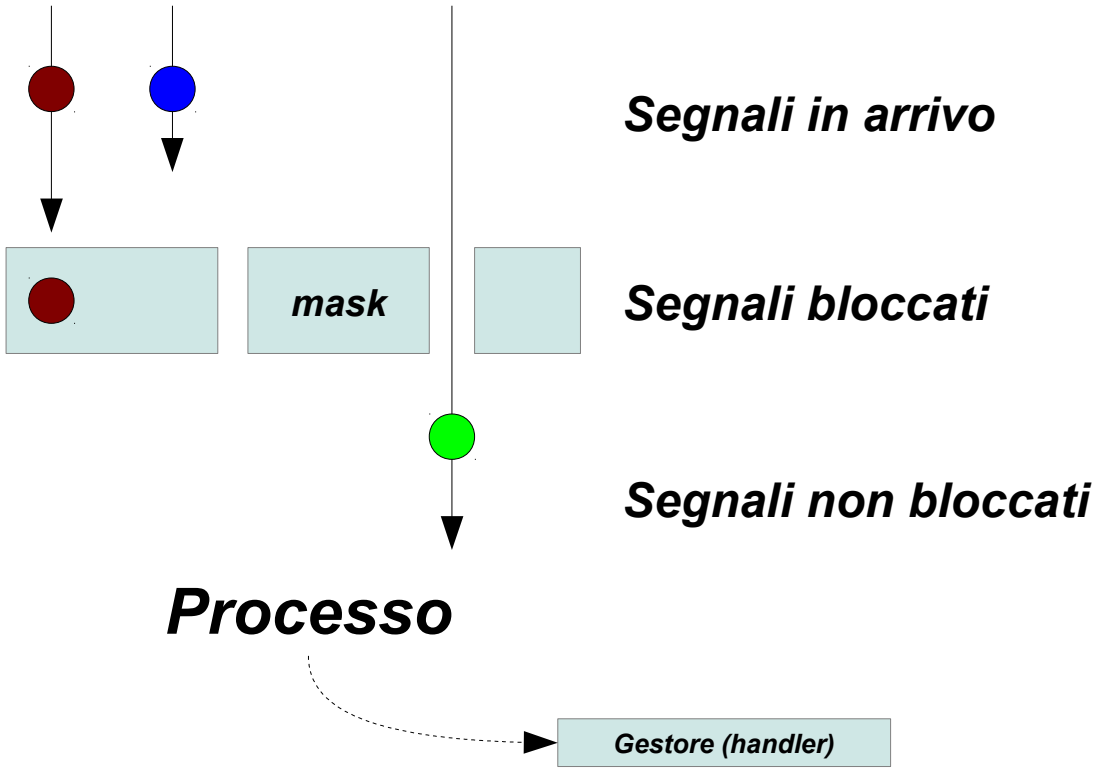
```
utente@host:~$ kill -SIGKILL 713
```



Il processo gnome-calculator (PID = 713) è terminato

Bloccare i segnali

- Un processo può bloccare un insieme di segnali (*set*) e decidere di gestirli successivamente (sbloccandoli)



Segnali non-bloccabili

- I segnali **SIGKILL** e **SIGSTOP** non possono essere intercettati e gestiti, bloccati, o ignorati.

Stato di un segnale

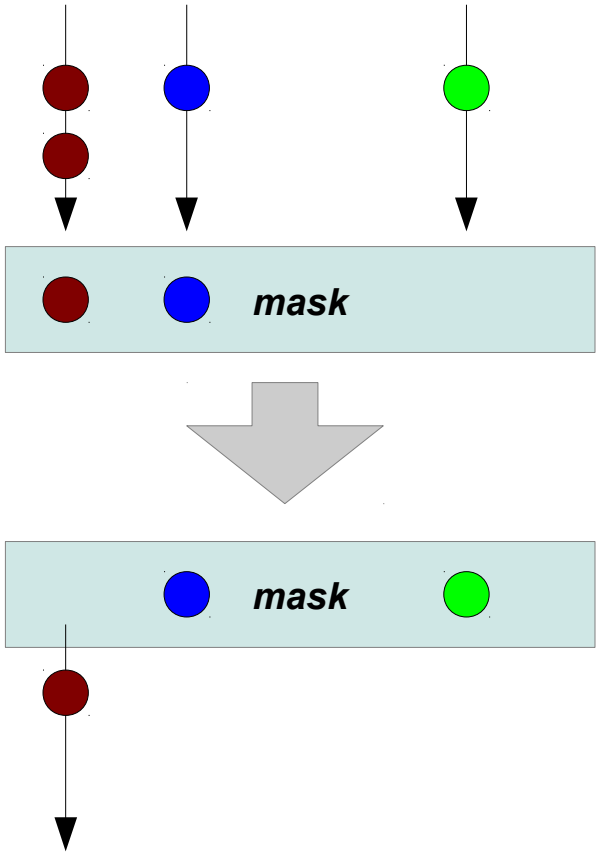
- **Pendente** (*pending*)
  - Tra l'istante della generazione e quello della consegna
- **Bloccato**
  - Quando la consegna viene impedita
    - *Un segnale bloccato è ancora pendente!*
- **Consegnato**
  - Quando ha luogo la reazione del destinatario

Tipi di segnali

- **Segnali normali**
  - Semplici valori numerici
  - Non sono memorizzabili
  - Non esiste nessuna regola sulla sequenza temporale di consegna

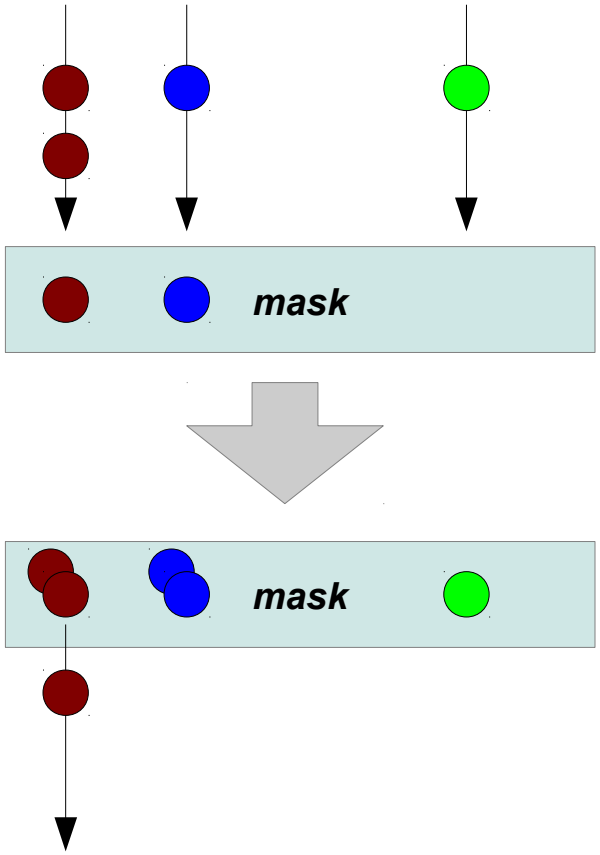
- **Segnali real time**
  - Prevedono un meccanismo di memorizzazione in coda
  - Permettono anche un piccolo allegato

Semantica dei segnali normali



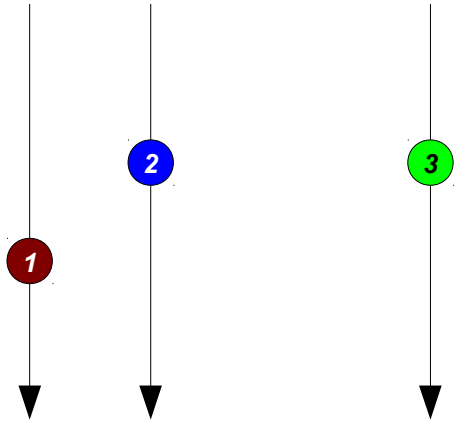
È possibile avere solo un solo segnale per ogni tipo in coda (pending o bloccato). Se il processo riceve più segnali dello stesso tipo prima che il gestore associato venga eseguito, solo il primo viene correttamente gestito

Semantica dei segnali real time



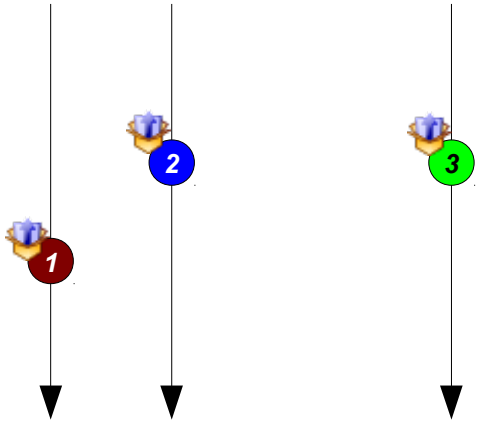
Se il processo riceve più segnali dello stesso tipo prima che il gestore associato venga eseguito, **i segnali vengono accodati**

Semantica dei segnali normali



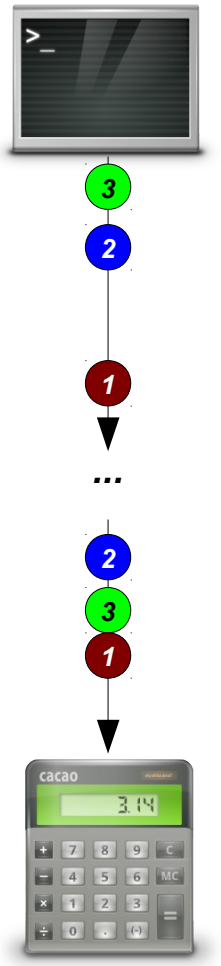
I segnali sono eventi semplici (numerici) che non trasportano nessuna informazione aggiuntiva

Semantica dei segnali real time



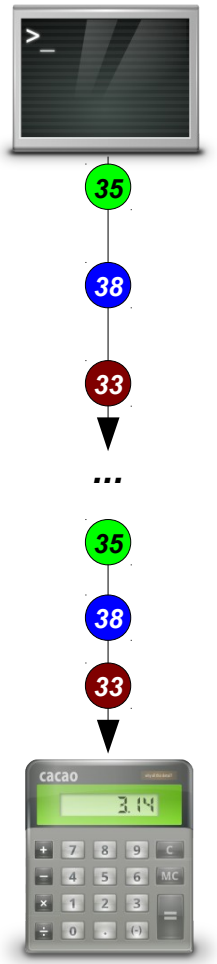
I segnali **possono trasportare una piccola informazione aggiuntiva**

Semantica dei segnali normali



I segnali **non hanno un ordine di consegna** preciso o una priorità

Semantica dei segnali real time



L'ordine di consegna rispetta l'ordine di invio. Se più segnali sono in coda, **l'ordine di consegna rispecchia la priorità del segnale, da SIGRTMAX a SIGRTMIN (i numeri di segnale più bassi indicano una priorità più alta).** Se il processo sta gestendo un segnale, solo un segnale con priorità più alta può interromperlo. *

*** I segnali normali hanno tutti la stessa priorità, che è più alta dei segnali real-time.**

Inviare un segnale

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

Inviare un segnale all'interno del processo

```
int raise(int sig);
```

```
#include <sys/types.h>
#include <signal.h>
```

Inviare un segnale a un altro processo

```
int kill(pid_t pid, int sig);
```

Inviare un segnale a un processo da shell

```
kill [ -signal | -s signal ] pid ...
```

Inviare un segnale real time

```
#include <signal.h>

int sigqueue(pid_t pid, int sig, const
                union sigval value);
```

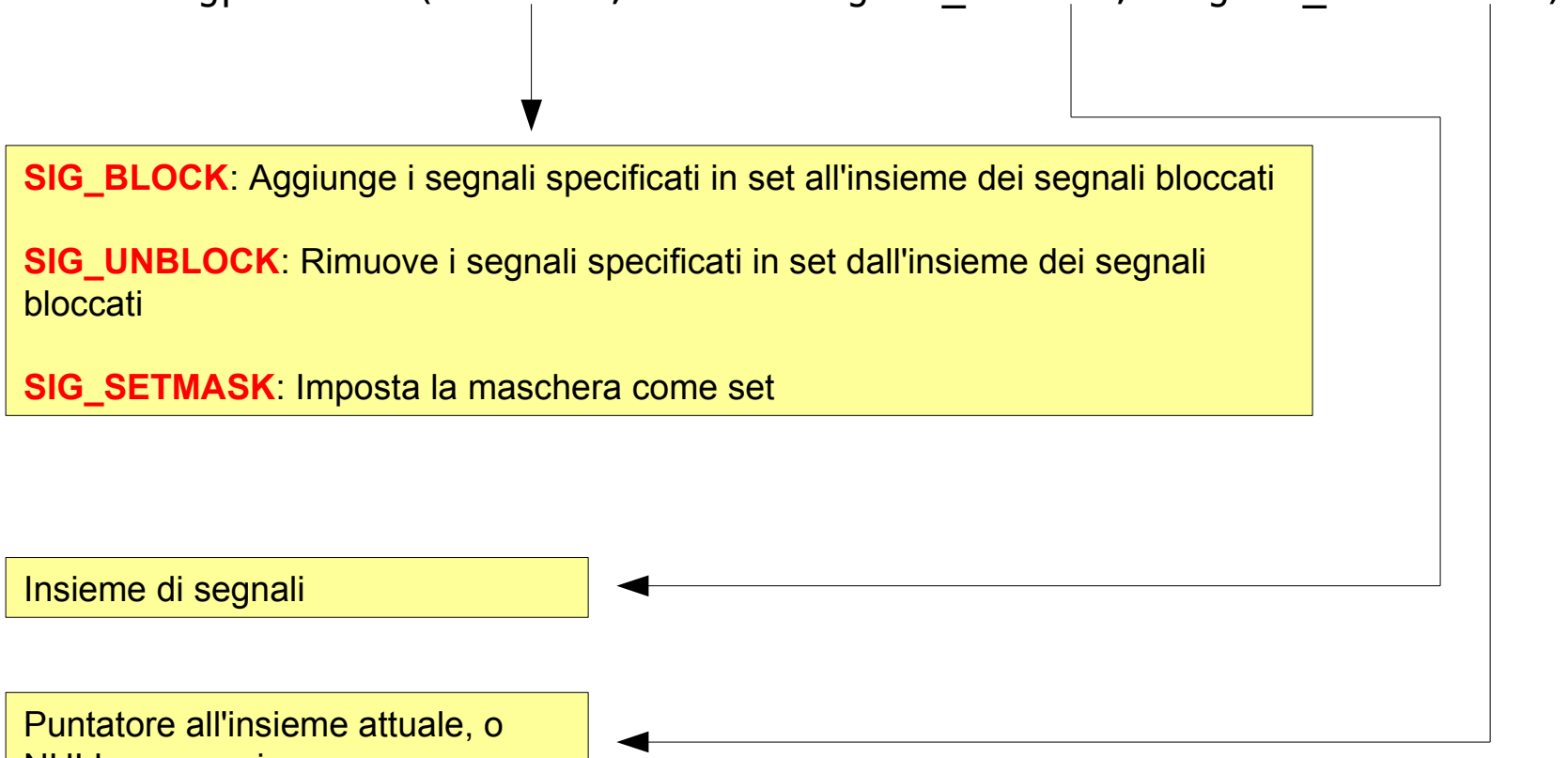
Inviare un segnale realtime con un allegato

```
union sigval {
  int   sival_int;
  void *sival_ptr;
};
```

Informazioni aggiuntive che possono essere allegate

Bloccare e sbloccare i segnali

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset)
```



Creare e modificare un insieme di segnali

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

Inizializza l'insieme senza nessun segnale

```
int sigfillset(sigset_t *set);
```

Inizializza l'insieme con tutti i segnali

```
int sigaddset(sigset_t *set, int signum);
```

Aggiunge un nuovo segnale all'insieme

```
int sigdelset(sigset_t *set, int signum);
```

Rimuove un segnale dall'insieme

```
int sigismember(const sigset_t *set, int signum);
```

Verifica se il segnale è membro dell'insieme oppure no

Esempio

```
#include <sys/types.h>
#include <signal.h>

void main() {
   sigset_t segnA, segnB;

   sigemptyset(&segnA);

   sigaddset(&segnA, SIGHUP);
   sigprocmask(SIG_BLOCK, &segnA, &segnB);
}
```

Gestire un segnale

- Tipicamente un processo non può ignorare un segnale in arrivo
  - Certi segnali possono essere intercettati, e il programma (o meglio, il programmatore) può definire cosa fare attraverso un gestore di segnale (**signal handler**)
  - Altri segnali invece non possono essere intercettati da un processo (es. SIGSTOP o SIGKILL), e solo la reazione predefinita è possibile

Gestore di segnale

- Con il comando **sigaction** è possibile associare una funzione che verrà invocata quando viene ricevuto un determinato segnale

```
#include <signal.h>

int sigaction(int signum,
              const struct sigaction *act,
              struct sigaction *oldact);
```

Puntatore alla funzione di gestione (oppure **SIG_IGN** per ignorare segnale, o **SIG_DFL** per specificare la reazione predefinita)

NULL, oppure ritorna un puntatore alla precedente sigaction

struct sigaction

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

Obsoleto, e non fa parte dello standard POSIX

Puntatori alla funzione di gestione: è possibile utilizzare il primo campo *oppure* il secondo (che permette di ottenere più informazioni)

Flag per specificare il comportamento dell'handler

Insieme di segnali da bloccare mentre è in esecuzione la gestione del segnale (a cui si aggiunge automaticamente il segnale stesso)

Flags

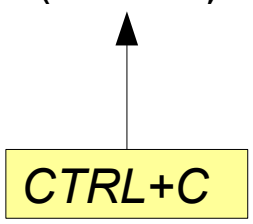
- **SA_NOCLDSTOP**
  - Il segnale SIGCHLD non è generato quando il processo figlio viene fermato (cioè quando riceve SIGSTOP, SIGTSTP, SIGTTIN or SIGTTOU) o riprende l'esecuzione (SIGCONT). Utile solo se viene definito un gestore per SIGCHLD.
- **SA_NOCLDWAIT**
  - Se il segnale è SIGCHLD non trasforma i sotto-processi in zombie quando terminano (ovvero, non è necessario che il padre utilizzi wait o waitpid)
- **SA_NODEFER**
  - Evita che il segnale corrente sia bloccato durante l'esecuzione del gestore (come lo è di regola)
- **SA_RESTART**
  - Permette di ricominciare alcune chiamate di sistema se sono state interrotte da un segnale
- **SA_SIGINFO**
  - Se definito, utilizza la seconda funzione (sa_sigaction) come gestore

siginfo_t

```
siginfo_t {
  int      si_signo;    /* Signal number */
  int      si_errno;    /* An errno value */
  int      si_code;     /* Signal code */
  int      si_trapno;   /* Trap number that caused hardware-generated signal (unused on most
                           architectures) */
  pid_t    si_pid;      /* Sending process ID */
  uid_t    si_uid;      /* Real user ID of sending process */
  int      si_status;   /* Exit value or signal */
  clock_t  si_utime;    /* User time consumed */
  clock_t  si_stime;    /* System time consumed */
  sigval_t si_value;    /* Signal value */
  int      si_int;      /* POSIX.1b signal */
  void    *si_ptr;      /* POSIX.1b signal */
  int      si_overrun;  /* Timer overrun count; POSIX.1b timers */
  int      si_timerid;  /* Timer ID; POSIX.1b timers */
  void    *si_addr;     /* Memory location which caused fault */
  long     si_band;     /* Band event (was int in glibc 2.3.2 and earlier) */
  int      si_fd;       /* File descriptor */
  short    si_addr_lsb; /* Least significant bit of address (since kernel 2.6.32) */
}
```

Esempio

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

void gestore(int sig) {
     write(0, "Ho ricevuto un segnale!\n", 24);
}

int main(void) {
     struct sigaction saction;
     saction.sa_handler = gestore;
     sigemptyset(&saction.sa_mask);
     sigaction(SIGINT, &saction, NULL);
     pause();
}
```



Gestione sincrona di un segnale: aspettare un segnale

```
#include <signal.h>

int sigwait(const sigset_t *set, int *sig);
```

- La funzione seleziona un segnale pendente in set, lo elimina dalla coda, e lo ritorna tramite il puntatore sig
  - Il segnale è eliminato dalla lista dei segnali pendenti
  - Se al momento della chiamata ci sono più istanze pendenti dello stesso segnale e il segnale è accodabile (realtime), le restanti istanze restano pendenti
  - Nel caso di segnali non accodabili, l'esistenza o meno di ulteriori istanze dipende dall'implementazione

Esempio

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

int main(void) {
  int segnale;
  sigset_t insieme;

  sigemptyset(&insieme);
  sigaddset(&insieme,SIGINT);
  sigprocmask(SIG_BLOCK, &insieme, NULL);
  sigwait(&insieme, &segnale);
  printf("Ricevuto segnale %d\n", segnale);
  sigprocmask(SIG_UNBLOCK, &insieme, NULL);
}
```

Gestione sincrona di un segnale: aspettare un segnale

```
#include <signal.h>

int sigwaitinfo(const sigset_t *set, siginfo_t *info);
```

- Simile a sigwait, ma permette di ottenere anche le informazioni aggiuntive sul segnale attraverso la struttura info

Gestione sincrona di un segnale: aspettare un segnale

```
#include <signal.h>

int sigtimedwait(const sigset_t *set, siginfo_t *info,
    const struct timespec *timeout);
```

- Come sigwaitinfo, ma è possibile definire un timeout per l'attesa

```
struct timespec {
  long tv_sec; /* secondi */
  long tv_nsec; /* nanosecondi */
};
```

Esempio

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <errno.h>

int main(void) {
  int segnale;
  sigset_t insieme;
  struct timespec timeout;
  siginfo_t info;

  timeout.tv_sec = 5;

  sigemptyset(&insieme);
  sigaddset(&insieme,SIGINT);
  sigprocmask(SIG_BLOCK, &insieme, NULL);
  if (sigtimedwait(&insieme, &info, &timeout) < 0) {
    if (errno == EAGAIN)
      printf("Timeout!\n");
    else
      perror("Errore!\n");
    exit(-1);
  }
  printf("Ricevuto segnale %d\n", info.si_signo);
}
```

Gestione sincrona di un segnale: aspettare un segnale

```
#include <signal.h>

int sigsuspend(const sigset_t *mask);
```

- La funzione sigsuspend(), a differenza della funzione sigwait(), permette di specificare alla sua chiamata un insieme di segnali da bloccare durante l'intervallo in cui il processo è sospeso (maschera dei segnali bloccati)
  - Alla ricezione di un segnale "gestito" non bloccato, la funzione asincrona associata viene eseguita. **I segnali nella maschera quindi non possono risvegliare il programma fermo su sigsuspend!**
  - Alla fine della procedura asincrona, la funzione sigsuspend() termina e la maschera dei segnali bloccati viene sostituita con quella precedente la chiamata.

Gestione sincrona di un segnale: aspettare un segnale

```
#include <signal.h>

int sigpending(sigset_t *set);
```

- Ritorna, attraverso set, l'insieme dei segnali che sono attualmente nello stato pending

Esempio

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

void main() {
    sigset_t segnA;

    if (sigpending( &segnA ) != 0 ) {
        perror("Errore\n");
    } else if (sigismember(&segnA, SIGHUP )) {
        printf("Il segnale SIGHUP è pending\n");
    } else {
        printf("SIGHUP non è pending\n");
    }
}
```

Aspettare un segnale

```
#include <unistd.h>

int pause(void);
```

- Si ferma e attende che un segnale venga consegnato
  - pause() ritorna solo quando il segnale è gestito e la funzione di gestione termina (o un segnale non gestito fa terminare il processo)

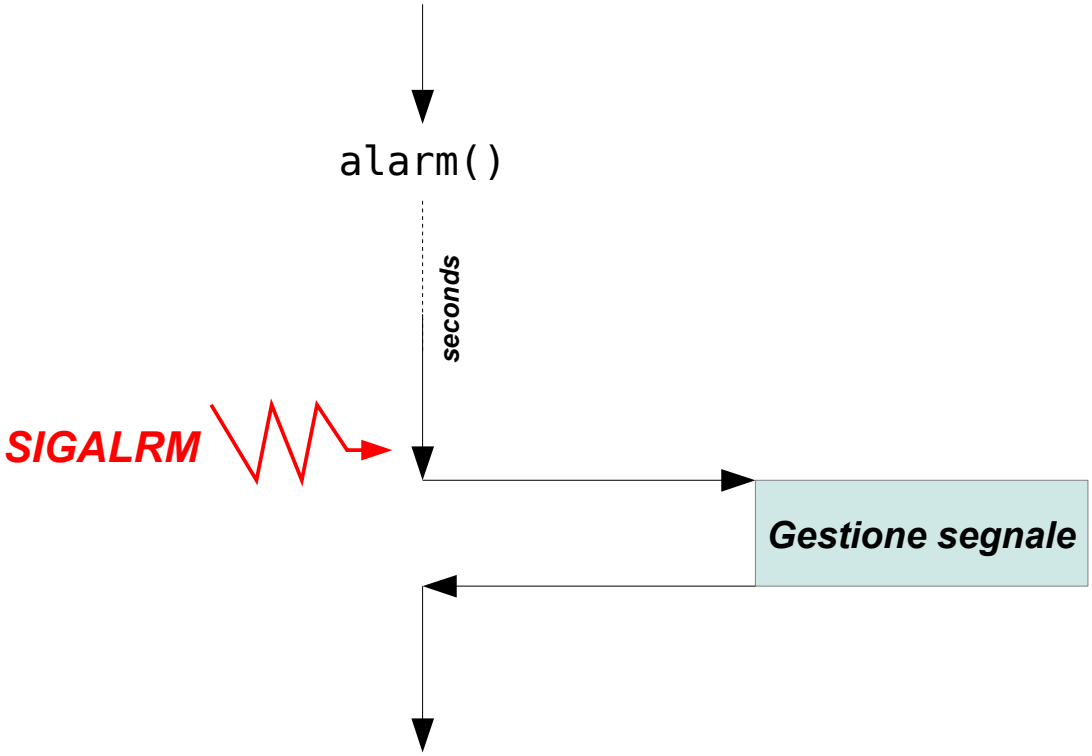
Esempio

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
void gestore(int sig) {
  write(0, "Ho ricevuto un segnale!\n", 24);
}
int main(void) {
  int segnale;
  sigset_t insieme;
  struct sigaction saction;

  saction.sa_handler = gestore;
  sigemptyset(&saction.sa_mask);
  sigaction(SIGINT, &saction, NULL);
  sigaction(SIGUSR1, &saction, NULL);
  sigemptyset(&insieme);
  sigaddset(&insieme,SIGINT);
  printf("Blocco il segnale SIGINT (CTRL+C)\n");
  sigprocmask(SIG_BLOCK, &insieme, NULL);
  pause(); /* Solo SIGUSR1 può sbloccarmi */
  printf("Termino!\n");
}
```

Esempio

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
void gestore(int sig) {
  write(0, "Ho ricevuto un segnale!\n", 24);
}
int main(void) {
  int segnale;
  sigset_t insieme;
  struct sigaction saction;

  saction.sa_handler = gestore;
  sigemptyset(&saction.sa_mask);
  sigaction(SIGUSR1, &saction, NULL);
  sigemptyset(&insieme);
  sigaddset(&insieme,SIGUSR1);
  printf("Blocco il segnale\n");
  sigprocmask(SIG_BLOCK, &insieme, NULL);
  printf("Invio il segnale\n");
  raise(SIGUSR1); // Bloccato
  printf("Sblocco il segnale\n");
  sigprocmask(SIG_UNBLOCK, &insieme, NULL);
  sleep(1);
  printf("Termino\n");
}
```

Programmazione asincrona: una sveglia




```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

Programmazione asincrona: una sveglia

```
/* sigalrm */

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

struct sigaction act;
sigset_t set;

void sveglia(int ignored) {
 printf("drin...drin...drin\n");
}

main() {
 int i,time;
 act.sa_handler=sveglia;
 if(sigaction(SIGALRM,&act,0) !=0) return(-1);
 printf("Numero di secondi di attesa: ");
 scanf("%d",&time);

 sigemptyset(&set);
 alarm(time);
 sigsuspend(&set);
}
```

Risveglio dopo che il gestore sveglia() viene terminato o se il processo viene terminato!

Scoprire la causa di un segnale con siginfo_t

Membro	Tipo	
si_signo	int	numero del segnale
si_errno	int	numero dell'errore come definito in <errno.h>
si_code	int	codice che specifica la causa del segnale
		Per i segnali dell'utente sono definiti i valori:
		SI_USER causa: kill(),sigsend(),raise(),abort()
		SI_QUEUE causa: sigqueue()
		SI_TIMER causa: timer_settime()
		SI_ASYNCIO causa: complemento richiesta asincrona
		SI_MESGQ causa: arrivo messaggio in una coda vuota (mq_notify())
		Per i segnali di sistema vale:
		SIGILL ILL_ILLOPC *illegal opcode*
		ILL_ILLOPN *illegal operand*
		ILL_ILLADR *illegal address*
		ILL_ILLTRP *illegal trap*
		ILL_PRVOPC *privileged opcode*
		ILL_PRVREG *privileged register*
		ILL_COPROG *co-processor error*
		ILL_BADSTK *internal stack-error*
		SIGFPE FPE_INTDIV *integer divide bye zero*
		FPE_INTOVF *integer overflow*
		FPE_FLTDIV *floating point divide by zero*
		FPE_FLTOVF *floating point overflow*
		FPE_FLTUND *floating point underflow*
		FPE_FLTRES *floating point inexact result*
		FPE_FLTINV *invalid floating point operation*
		FPE_FLTSUB *subscript out of range*

Scoprire la causa di un segnale con siginfo_t

```
/* pause_siginfo */

/* Per far uscire il processo dalla pausa: kill -USR1 ID_processo
   oppure usare il programma sigqueue della medesima collezione
*/

#include <stdio.h>
#include <signal.h>
#include <signal.h>
#include <sys/time.h>

char *scodice[]={"SI_USER","SI_ASYNCIO","SI_MESGQ"};
int    codice[]={SI_USER,SI_MESGQ} , i;
struct itimerval quando;

void f2(int segnale,siginfo_t *si,void *p) {
printf ("Sono la f2 richiamata dal segnale %d\n",segnale);
for (i=0;i<2;i++){
  if(si->si_code==codice[i]) printf("si_code: %s\n",scodice[i]);
}
printf ("Informazione in siginfo_t: si_signo=%d,",si->si_signo);
printf (" si_pid=%d, si_uid=%d\n",si->si_pid, si->si_uid );
if(si->si_code==SI_QUEUE)
  printf("si_code: SI_QUEUE, si_value=%d\n",si->si_value );
}
```

Scoprire la causa di un segnale con siginfo_t

```
main() {
 struct sigaction act;
 act.sa_sigaction= f2;
 act.sa_flags=SA_SIGINFO;
 sigaction(SIGUSR1,&act,0) ;
 sigaction(SIGUSR2,&act,0) ;
 sigaction(SIGALRM,&act,0) ;

 alarm(10); /* primo allarme */
 quando.it_interval.tv_sec=10;
 quando.it_interval.tv_usec=0;
 quando.it_value.tv_sec=15;
 quando.it_value.tv_usec=0;
 setitimer(ITIMER_REAL,&quando,0); /* secondo allarme */
 while(1) {
    printf("Il processo %d%s",getpid()," va in pausa\n");
    pause();
    printf("Il processo %d%s",getpid()," continua\n");
 }
}
```

Indichiamo che verrà utilizzato il secondo tipo di gestore (che permette di ottenere più informazioni sul segnale)

Generalizzazione di alarm: anche lui invia SIGALRM