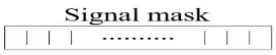
# Gestione eventi 3 Signal Mask

giovedì 27 novembre 2025 11:47

Andiamo ora a vedere come vengono gestiti gli errori/segnalazione in modo software : usiamo la **signal mask**




La quale per ogni bit (0 e 1) va a mostrare le segnalazioni avvenute. Possibile consegne multiple ( più processi su stesso segnale) ma attenzione che possono essere perse. Nota : i gestori delle segnalazioni non sono atomiche. Quindi in generale capita che vi siano azioni atomiche complesse sulle segnalazioni. Per risolvere questo problema si usa la seguente soluzione : si usa la **bit mask (Set segnali)** ovvero un'area di memoria che in base al valore dei bit , permette di utilizzare le segnalazioni o no nella signal mask (aggiungere/levare -> 1/0) . In dettaglio :

Il tipo **sigset_t** rappresenta un insieme di segnali (signal set).
Funzioni (definite in signal.h) per la gestione dei signal set:

**int sigemptyset (sigset_t *set)** svuota il set

**int sigfillset (sigset_t *set)** inserisce tutti i segnali in set

**int sigaddset (sigset_t *set, int signo)**
aggiunge il segnale signo a set

**int sigdelset (sigset_t *set, int signo)**
toglie il segnale signo da set

**int sigismember (sigset_t *set, int signo)**
controlla se signo è in set

Vediamo ora come gestire della maschera dei segnali del thread corrente :

| int sigprocmask(int *how*, const sigset_t **set*, sigset_t **oset*) | |
|---|---|
| **Descrizione** | imposta la gestione della signal mask |
| **Argomenti** | 1) how: indica in che modo intervenire sulla signal mask e può valere: SIG_BLOCK: i segnali indicati in set sono aggiunti alla signal mask, SIG_UNBLOCK: i segnali indicati in set sonorimossi dalla signal mask; SIG_SETMASK: La nuova signal mask diventa quella specificata da set<br>2) set: il signal set sulla base del quale verranno effettuate le modifiche<br>3) oset: se non è NULL, nella relativa locazione verrà scritto il valore della signal mask PRIMA di effettuare le modifiche richieste |
| **Restituzione** | –1 in caso di errore |

Vediamo ora altra syscall : vediamo i segnali pendenti :

| int sigpending(sigset_t **set*); | |
|---|---|
| **Descrizione** | restituisce l'insieme dei segnali che sono pendenti |
| **Argomenti** | set: il signal set in cui verrà scritto l'insieme dei segnali pendenti |
| **Restituzione** | –1 in caso di errore |

Permette l'implementazione di schemi di polling, in particolare nel caso di applicazioni multi-thread

Vediamo ora in realtà come gestirli (cambio stato contro uno specifico segnale) in modo atomico:

| int sigaction(int *sig*, const struct sigaction * *act*, struct sigaction * *oact*); | |
|---|---|
| **Descrizione** | permette di esaminare e/o modificare l'azione associata ad un segnale |
| **Argomenti** | 1) sig: il segnale interessato dalla modifica;<br>2) act: indica come modificare la gestione del segnale<br>3) oact: in questa struttura vengono memorizzate le impostazioni precedenti per la gestione del segnale |
| **Restituzione** | –1 in caso di errore |

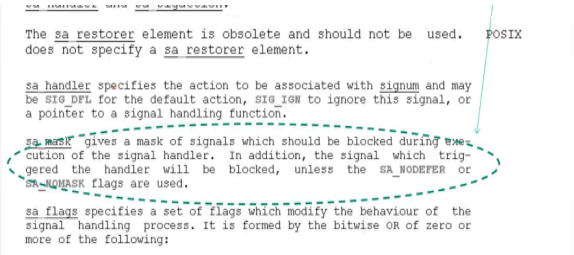
Dove la sigaction è una struttura fatta come segue:

The sigaction structure is defined as something like

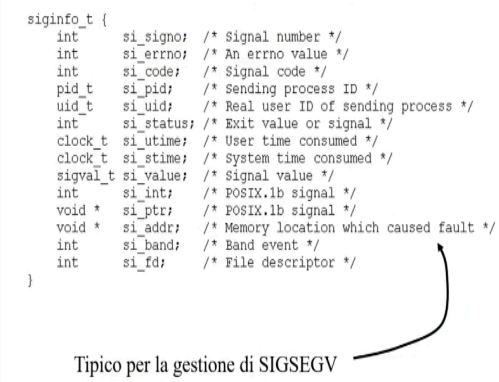
```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

Supporti per l'atomicita'

On some architectures a union is involved - do not assign to both sa_handler and sa_sigaction.




I due membri della struct rappresentano l'handler , ma sono mutuamente esclusivi. In dettaglio (il secondo membro della struct) a sua volta è una struct:




Vediamo ora degli esempi:

1. **Sigaction**
   a. Così si ha il polling ogni 5 secondi
   b.
```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

#define SLEEP_PERIOD 5

void generic_handler(int signal, siginfo_t * a, void *b){
  sigset_t set;
  printf("received signal is %d\n",signal);
  fflush(stdout);

}

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

  int  i;
  //registro 0/1 per i segnali
  sigset_t set; //bit mask
  struct sigaction act;
  //la riempio di tutti i segnali
  sigfillset(&set);
  //gestore da impostare
  act.sa_sigaction = generic_handler;
  //maschera dei segnali bloccati
  //mentre il gestore esegue
  //per arrivo segnalazione
  act.sa_mask =  set;
  //gestore che parte
  act.sa_flags = SA_SIGINFO;
  //valore di default
  act.sa_restorer = NULL;

  sigaction(SIGINT,&act,NULL);

  //il gestore dei segnali è bloccato
  sigprocmask(SIG_BLOCK,&set,NULL);

  while(1) {
        //dormo per 5 secondi
        sleep(SLEEP_PERIOD);
        //chiedo al kernel se ci
        //sono segnali pendenti
        sigpending(&set);
        //tra i segnali pendenti ciè sig*?
        if(sigismember(&set,SIGINT)){
          //svuoto la maschera dei segnali
          sigemptyset(&set);
          //aggiungo segnale
          sigaddset(&set,SIGINT);
          //sblocco maschera dove solo SIGINT
          sigprocmask(SIG_UNBLOCK,&set,NULL);
          //riblocco maschera di segnalazione
          sigprocmask(SIG_BLOCK,&set,NULL);
        }
  }
}
```
   c. Il gestore stampa il segnale dopo 5 secondi
   d. Non si riesce a buttare giù (bisogna tirarla giù da altra applicazione)
2. **Sigsegv trace**
   a. Informazioni a grana fine
   b.
```c
#include <stdio.h>
#include <pthread.h>
#include <signal.h>
#include <unistd.h>


void generic_handler(int signal, siginfo_t *info, void* unused){

        sigset_t set;
        printf("received signal is %d - address is %p\n",signal,info->si_addr);
        fflush(stdout);
        sleep(1);
}


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

  int  i;
  char c;
  sigset_t set;
  struct sigaction act;
  //pagina non mappata
  char* addr = (char*)0xffffff00;
```

```
  //riempio set segnali
  sigfillset(&set);
 //sigdelset(&set,SIGSEGV);
  //elimino il SIGINT
  sigdelset(&set,SIGINT);

  act.sa_sigaction = generic_handler;
  act.sa_mask =  set;
  act.sa_flags = SA_SIGINFO;
  act.sa_restorer = NULL;
  //gestore per segmentation fault
  sigaction(SIGSEGV,&act,NULL);


  while(1) {
        sleep(5);
        c = *addr;
  }
}
```

c. Prima volta si aspetta 5 secondi e poi segmentation fault : ogni secondo ritorna al gestore del segnale.

3. **Sigpoll**

a. L'applicazione termina dopo 5 secondi che il SIGINT è stato inviato in quanto non c'è una gestione esplicita del SIGINT

b.

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

#define SLEEP_PERIOD 5

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

  int  i;
  sigset_t set;


  sigfillset(&set);
  sigprocmask(SIG_BLOCK,&set,NULL);

  while(1) {
        sleep(SLEEP_PERIOD);
        printf("querying the sigset\n");
        sigpending(&set);

        if(sigismember(&set,SIGINT)){
          sigemptyset(&set);
          sigaddset(&set,SIGINT);
          sigprocmask(SIG_UNBLOCK,&set,NULL);
        }
  }

}
```

4. Exit value

a. Applicazione termina o se exit() oppure se segnale terminato implicitamente

b.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/wait.h>
#define COMMAND_LENGTH 1024

int main (int argc, char *argv[]){
 int i, status;
 if (argc<2) {  printf("Need at least a parameter\n"); exit(-1); }
 if ((i=fork()) == 0) { execvp(argv[1],&argv[1]);
                        printf("Can't execute file %s\n",argv[1]);
                        exit(-1);
 } else if (i<0) { printf("Can't spawn process for error %d\n", errno);   exit(-1); }
 wait(&status);

  if ((status & 255) == 0) {
     printf("\nProcess regularly exited with exit status %d\n\n", (status>>8) & 255);  }
  else if ( ((status>>8) & 255) == 0) {
       printf("\nProcess abnormally terminated by signal: %d\n\n",  status & 255);
      }
}
```

| exit code | 0 |
|---|---|
| 0 | signal number |

bytes meno significativi

c.

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

int main(int a, char **b){

        int i;
        int value;

        for (i=0; i<10; i++){
                if(fork() == 0) pause();
        }

        signal(SIGINT,SIG_IGN);

        for (i=0; i<10; i++){
                wait(&value);
                printf("last two bytes of value are %d - %d\n",
(value>>8)&255,value&255);
        }
```

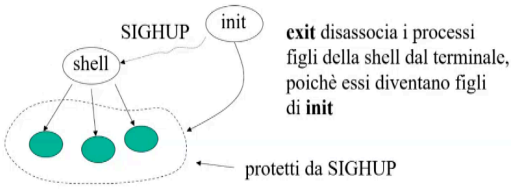
d. Il parent non termina per il ctrl+c ,ma solo i child

5. **Sighup**
   a. Segnale chiusura di terminale

comandi eseguiti in background vengono terminati o non alla chiusura del terminale associato alla shell dipendendo dalle impostazioni sul trattamento di SIGHUP

1. Non terminano se il costrutto fork/exec imposta il trattamento a SIG_IGN (argomento **nohup** sulla linea di comando)
2. Terminano in ogni altro caso a meno che il terminale sia chiuso per effetto della system call **exit** eseguita dalla shell

   b.




**exit** disassocia i processi figli della shell dal terminale, poichè essi diventano figli di **init**

In generale i segnali supportati sono i seguenti:

| Valore `sig` | Descrizione |
|---|---|
| `SIGABRT` | Terminazione anomala |
| `SIGFPE` | Errore a virgola mobile |
| `SIGILL` | Istruzione non valida |
| `SIGINT` | Segnale CTRL+C |
| `SIGSEGV` | Accesso alla memoria non valido |
| `SIGTERM` | Richiesta di terminazione |

← Compatibilità solo nominale