Signal

Meccanismo per notificare i processi

- eventi asincroni (es. l'utente preme ctrl-C al terminale)
- eventi sincroni (es. accesso a un indirizzo illegale)

Sequenza di eventi:
0. il segnale è *generato*
... il segnale è *pendente* ...
1. il segnale è *consegnato* (handled) quando il processo destinatario reagisce in maniera appropriata

I segnali sono identificati da *numeri* o *costanti simboliche*
es. SIGINT o 2: interrupt from keyboard

Unix supporta 31 diversi segnali

Gestione dei segnali

Ogni segnale ha una *default action*
- abort: termina il processo con *core dump*
- exit: termina il processo
- ignore: ignora il segnale
- stop: sospende il processo
- continue: riavvia il processo, se sospeso

Un processo può *cambiare l'azione di default*:
- ignorare il segnale
- eseguire una subroutine definita dall'utente detta *signal handler*

Un processo può bloccare temporaneamente un segnale

per SIGKILL e SIGSTOP non si può cambiare l'azione di default

Quando vengono gestiti?

Solo il processo destinatario può gestire il segnale

Il kernel controlla se ci sono segnali pendenti
- prima di ritornare in modo utente dopo un interrupt o syscall
- prima che il processo chiami wait_interruptible()
- all'uscita da wait_interruptible()

Se ci sono segnali pendenti, e il processo è in user-mode:
- il kernel salva il contesto corrente del processo ed esegue l'azione specificata

Se il processo è in kernel-mode:
- la syscall termina con errore EINTR (interrupted system call)

Molte chiamate di sistema vengono fatte *ripartire automaticamente* dopo la gestione dell'interrrupt

Quando vengono generati?

- Eccezioni
  Es. tentativo di eseguire un istruzione illegale
- Altri processi
  Possono eseguire la kill(2) per mandare un segnale
- Interrupt da terminale
  I tasti ctrl-C o ctrl-\ mandano un interrupt al processo in *foreground*
- Job control
  Un processo in *background* che cerca di leggere o scrivere da terminale riceve un interrupt.
- Quote
  Es. quota di cpu ecceduta
- Notifiche
  I\O non bloccante: un processo riceve un signal quando l'operazione è terminata
- Alarm
  Un processo può richiedere un SIGALRM a una certa ora

Esempio: ctrl-C da terminale

L'utente preme ctrl-C

Viene generato un interrupt (come per qualsiasi pressione di tasto)

Il driver del terminale riconosce che è un carattere speciale e manda SIGINT al processo in foreground

Quando il processo viene schedulato, gestirà il segnale al ritorno dal context switch

Se invece il processo è corrente, gestirà il segnale al ritorno dall'interrupt handler

Esempio: eccezione hardware

Il programma fa accesso a indirizzo di memoria illegale

La CPU genera un interrupt sincrono

L'interrupt handler manda SIGSEGV al processo

Quando l'interrupt handler ritorna dal kernel mode, il processo gestisce l'interrupt

E se il processo è bloccato?

Dipende da che cosa sta aspettando

- attese brevi (I/O completion): non vale la pena di interrompere l'attesa
- attese lunghe (input da terminale): l'attesa viene interrotta

Due categorie di wait:

- interruptible
- uninterruptible

Segnali inaffidabili

La prima implementazione di signal in Unix System V R3 era inaffidabile

```
void handler(int sig) {
  signal(SIGINT, handler);  // reinstalla lo handler
  ...                       // gestisci il segnale
}

main() {
  signal(SIGINT, handler); // installa lo handler
  ...
}
```

Gli handler non sono persistenti, e devono essere reinstallati ogni volta
⇒ race condition!

Segnali affidabili

- Handler persistenti
- Masking
- Atomic "unblock and wait"


```
SYNOPSIS
       #include <signal.h>

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

DESCRIPTION
       The sigaction system call is used  to  change  the  action
       taken by a process on receipt of a specific signal.

       signum  specifies  the  signal and can be any valid signal
       except SIGKILL and SIGSTOP.

       If act is non-null, the new action for  signal  signum  is
       installed  from  act.  If oldact is non-null, the previous
       action is saved in oldact.
```



```
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;
       }
sa_handler specifies the  action  to  be  associated  with
signum  and may be SIG_DFL for the default action, SIG_IGN
to ignore this signal, or a pointer to a  signal  handling
function.  This function receives the signal number as its
only argument.

sa_sigaction also specifies the action  to  be  associated
with  signum.  This function receives the signal number as
its first argument, a pointer to a siginfo_t as its second
argument and a pointer to a ucontext_t (cast to void *) as
its third argument.

sa_mask gives a mask of signals which  should  be  blocked
during  execution of the signal handler.  In addition, the
signal which triggered the handler will be blocked, unless
the SA_NODEFER or SA_NOMASK flags are used.
```