(Stevens)
Capitolo 10: Signals

Segnali

- Un segnale è un interrupt software che permette di gestire eventi asincroni,
  - come ad esempio un CTRL-C inviato da un utente seduto ad un terminale o la fine prematura di un processo per un errore di sistema.
- Un segnale può essere generato in qualsiasi istante,
  - a volte da un processo utente, più spesso dal kernel a seguito di un errore software o hardware o comunque di un evento eccezionale.
- Ogni segnale ha un nome che comincia con SIG (ad es. SIGABRT, SIGALARM) a cui viene associato una costante intera ($\neq 0$) positiva definita in signal.h
- I segnali non contengono informazioni aggiuntive,
  - in particolare chi li riceve non può scoprire l'identità di chi li manda.

System Calls e Funzioni di libreria

- Il segnale e' un evento asincrono;
  - esso puo' arrivare in un momento qualunque ad un processo ma non necessariamente quando il segnale viene ricevuto verrà fatta qualcosa.
  - Vi sono azioni di default che vengono compiute
  - oppure il processo può scegliere di ignorare il segnale o gestirlo in maniera diversa dal default.
- Quindi le azioni associate ad un segnale sono le seguenti:
  - Ignorare il segnale
    - ✓ tranne che per SIGKILL e SIGSTOP che non possono essere ignorati perché sono il modo che il superuser ha di terminare o stoppare momentaneamente qualsiasi processo
  - Catturare il segnale
    - ✓ equivale ad associare all'occorrenza del segnale l'esecuzione di una funzione utente;
    - ✓ ad es. se il segnale SIGTERM e' catturato possiamo voler ripulire tutti i file temporanei generati dal processo
  - Eseguire l'azione di default associata
    - ✓ terminazione del processo per la maggior parte dei segnali

Segnali Unix

Name	Description	ANSI C	POSIX.1	SVR4	4.3+BSD	Default action
SIGABRT	abnormal termination (abort)	•	•	•	•	terminate w/core
SIGALRM	time out (alarm)		•	•	•	terminate
SIGBUS	hardware fault			•	•	terminate w/core
SIGCHLD	change in status of child		job	•	•	ignore
SIGCONT	continue stopped process		job	•	•	continue/ignore
SIGEMT	hardware fault			•	•	terminate w/core
SIGFPE	arithmetic exception	•	•	•	•	terminate w/core
SIGHUP	hangup		•	•	•	terminate
SIGILL	illegal hardware instruction	•	•	•	•	terminate w/core
SIGINFO	status request from keyboard				•	ignore
SIGINT	terminal interrupt character	•	•	•	•	terminate
SIGIO	asynchronous I/O			•	•	terminate/ignore
SIGIOT	hardware fault			•	•	terminate w/core
SIGKILL	termination		•	•	•	terminate
SIGPIPE	write to pipe with no readers		•	•	•	terminate
SIGPOLL	pollable event (poll)			•		terminate
SIGPROF	profiling time alarm (setitimer)			•	•	terminate
SIGPWR	power fail/restart			•		ignore
SIGQUIT	terminal quit character		•	•	•	terminate w/core
SIGSEGV	invalid memory reference	•	•	•	•	terminate w/core
SIGSTOP	stop		job	•	•	stop process
SIGSYS	invalid system call			•	•	terminate w/core
SIGTERM	termination	•	•	•	•	terminate
SIGTRAP	hardware fault			•	•	terminate w/core
SIGTSTP	terminal stop character		job	•	•	stop process
SIGTTIN	background read from control tty		job	•	•	stop process
SIGTTOU	background write to control tty		job	•	•	stop process
SIGURG	urgent condition			•	•	ignore
SIGUSR1	user-defined signal		•	•	•	terminate
SIGUSR2	user-defined signal		•	•	•	terminate
SIGVTALRM	virtual time alarm (setitimer)			•	•	terminate
SIGWINCH	terminal window size change			•	•	ignore
SIGXCPU	CPU limit exceeded (setrlimit)			•	•	terminate w/core
SIGXFSZ	file size limit exceeded (setrlimit)			•	•	terminate w/core

Tipi di segnali (I)

- SIGABRT
  - questo segnale è generato da una chiamata alla system call ***abort***. Il processo quindi termina abnormalmente
- SIGCHLD
  - Ogni volta che un processo termina o viene fermato questo segnale viene inviato al padre.
  - Per default il padre lo ignora, se invece il padre vuole intercettarlo chiamerà una delle wait per ottenere il PID del processo ed il suo termination status.
- SIGCONT
  - E' inviato ad un processo che è "fermo" per fare riprendere la sua esecuzione.
- SIGFPE
  - (Floating point exception) viene ad esempio inviato quando c'e' una divisione per zero.

Tipi di segnali (II)

- SIGILL
  - (illegal instruction) viene inviato quando l'hardware individua una istruzione illegale, come ad es. istruzioni su floating point in una macchina che non ha hardware per gestire la virgola.
- SIGINT
  - (interrupt) viene inviato quando viene premuto il tasto di interrupt (DELETE o CTRL-C).
  - In genere viene utilizzato per terminare a run time un processo.
- SIGALRM
  - Generato dalla system call ***alarm***.
- SIGQUIT
  - (Quit) simile a SIGINT viene inviato quando viene digitato CTRL-/, generando un core file: una immagine in memoria del processo che può essere utilizzata per debugging.

Tipi di segnali (III)

- SIGKILL
  - E' uno dei due segnali che non possono essere ignorati, termina il processo che lo riceve.
- SIGSEGV
  - (segment violation) il processo ha fatto un riferimento ad un indirizzo che non è nel suo spazio indirizzi.
- SIGSTOP
  - Ferma un processo, non può essere ignorato.
- SIGSYS
  - (invalid system call) il processo ha eseguito una istruzione che il kernel ha interpretato come system call senza però fornire i parametri adeguati.
- SIGTERM
  - Il segnale di terminazione inviato per default dalla system call ***kill***.

Tipi di segnali (IV)

- SIGBUS, SIGEMT, SIGIOT, SIGTRAP
  - inviati quando sussistono problemi hardware. La loro interpretazione è system dependent.
- SIGUSR1, SIGUSR2
  - (user defined signals) possono essere utilizzati e definiti dall'utente. A volte sono usati (in maniera impropria) per la comunicazione tra processi.

signal

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);
```

- Restituisce:
  - SIG_ERR in caso di errore
  - il puntatore al precedente gestore del segnale se OK

signal (II)

- ***signal*** prende due argomenti: il nome del segnale ***signo*** ed il puntatore alla funzione ***func*** da eseguire come azione da associare all'arrivo di ***signo*** (nuovo signal handler).
- Restituisce il puntatore ad una funzione che prende come argomento un intero (e non restituisce niente) che rappresenta il puntatore al precedente signal handler.
- In caso di errore restituisce SIG_ERR (cioé -1).

signal (III)

- Il valore di ***func*** può essere:
  - SIG_IGN per ignorare il segnale (tranne che per SIGKILL e SIGSTOP)
  - SIG_DFL per settare l'azione associata al suo default
  - L'indirizzo di una funzione che sarà eseguita quando il segnale occorre
- L'azione del padre su un segnale viene ereditata dai figli.

kill e raise

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

int kill (pid_t pid, int signo);

int raise (int signo);
```

- Descrizione:
  - inviano il segnale ***signo*** specificato come argomento
  - ***kill*** invia
- Restituiscono:
  - 0 se OK,
  - -1 in caso di errore

kill e raise (II)

- ***kill*** manda un segnale ad un processo o ad un gruppo di processi specificato da pid
- ***raise*** consente ad un processo di mandare un segnale a se stesso
- Se il valore di ***pid*** è:
  - > 0, il segnale è inviato al processo con PID ***pid***.
  - == 0, il segnale è inviato a tutti i processi il cui process group ID è uguale a quello del processo che invia il segnale.
  - < 0 il segnale è inviato a tutti i processi il cui process group ID è uguale al valore assoluto di ***pid***

kill e raise (III)

- Il superuser può inviare segnali a qualsiasi processo,
- ma un processo necessita degli appositi permessi per inviare segnali ad un altro processo.
- I segnali non sono consigliabili come forma di comunicazione tra processi.
  - Un messaggio inviato sotto forma di segnale può essere perso se quel segnale è momentaneamente disattivato dal processo che lo deve ricevere.
  - Inoltre i segnali interrompono qualsiasi cosa viene eseguita in quel momento e questo può causare problemi.

Il comando shell kill

- *kill –l*

  lista i possibili segnali
- *kill -s 10 3211*

  invia il segnale 10 al processo 3211

sleep

#include <unistd.h>

unsigned int sleep (unsigned int secs);

- Descrizione:
  - Il processo che chiama ***sleep*** dorme finché il numero di secondi indicati da ***seconds*** è trascorso oppure interviene un segnale ed il signal handler ritorna.
  - Nel primo caso restituisce 0, altrimenti il numero di secondi mancanti.
- Restituisce
  - 0
  - oppure il numero di secondi “non dormiti”.

abort

#include <stdlib.h>

void abort (void);

- Descrizione:
  - Non ritorna mai
  - Invia il segnal SIGABRT al processo che la invoca.

alarm

#include <unistd.h>

unsigned int alarm (unsigned int secs);

- Descrizione:
  - Fa partire un conto alla rovescia di ***secs*** secondi.
  - Allo scadere, al processo viene inviato SIGALRM.
  - In realtà potrebbero passare più secondi ad esempio per ritardi di schedulazione.
  - Vi è un'unica "sveglia" per processo, quindi quando chiamiamo alarm se vi era un'altra "sveglia" precedentemente caricata verrà risettata e verrà restituito il numero di secondi mancanti alla sveglia precedente.
  - Se ***seconds*** è 0 la sveglia precedente è cancellata (e non si invia SIGALRM).
- Restituisce:
  - 0
  - il numero di secondi mancanti alla "sveglia" precedentemente settata.

pause

#include <unistd.h>

int pause (void);

- Descrizione:
  - ***pause*** sospende l'esecuzione finché non interviene un segnale (ed il signal handler ritorna).
- Restituisce:
  - -1 con ***errno*** settato a EINTTR.

Fattoriale e Fibonacci

```
int fatt (int n)
{
if (n==0) return (1);
    else return (n * fatt (n-1));
}

int fib (int n)
{
if (n<=1) return (n);
    else return (fib(n-1) + fib(n-2));
}
```