Segnali ed allarmi

Contiene lucidi tratti da: 2005-2007 Marco Faella, Clemente Galdi, Giovanni Schmid (Univrsità di Napoli Federico II) 2005-2007 Francesco Pedullà, Massimo Verola (Uniroma2), 2001-2005 Renzo Davoli (Università di Bologna), Alberto Montresor (Università di Bologna)

Segnali

- Un “segnale” e' un interrupt “software”
  - La terminologia corretta e' “exception” mentre “interrupt” e' usata solo per gli interrupt “hardware”
- Consente la comunicazione asincrona tra processi e/o tra device e processo
- Ogni segnale ha un proprio nome
  - Tutti i nomi cominciano per “SIG”
  - Definiti in <signal.h>
  - Associati ad interi positivi

Segnali

- **Caratteristiche dei segnali**
  - Ogni segnale ha un identificatore
    - Identificatori di segnali iniziano con i tre caratteri SIG
    - Es. **`SIGABRT`** è il segnale di abort
  - Numero segnali: 15-40, a seconda della versione di UNIX
    - POSIX: 18
    - Linux: 38
  - I nomi simbolici corrispondono ad un intero positivo
    - Definizioni di costanti in **`bits/signum.h`**

Generazione di segnali

Pressione di tasti speciali sul terminale

- Es: Premere il tasto **`ctrl-c`** genera il segnale **`SIGINT`**

Eccezioni hardware

- Divisione per 0 (**`SIGFPE`**)
- Riferimento non valido a memoria (**`SIGSEGV`**)
- L'interrupt viene generato dall'hardware, e catturato dal kernel; questi invia il segnale al processo in esecuzione

System call `kill`

- Permette di spedire un segnale ad un altro processo
- Limitazione: uid del processo che esegue **`kill`** deve essere lo stesso del processo a cui si spedisce il segnale, oppure 0 (root)

Genereazione di Segnali

Comando `kill`

- Interfaccia shell alla system call **`kill`**

Condizioni software

- Eventi asincroni generati dal software del sistema operativo, non dall'hardware della macchina
- Esempi:
  - terminazione di un child (**`SIGCHLD`**)
  - generazione di un alarm (**`SIGALRM`**)

Segnali

- I segnali vengono inviati in modo asincrono.
  - Non e' possibile sapere quando il processo ricevera' un segnale.
- E' possibile indicare al kernel *l'azione da intraprendere* quando un segnale e' generato per un processo:
  - Ignora: Valida per quasi tutti i segnali tranne SIGKILL e SIGSTOP.
  - "Catch" del segnale: Indicare una procedura da eseguire (signal handler). Ad esempio:
    - SIGCHLD: esegui le operazioni associate alla termiazione di un figlio
    - SIGINT: (CTRL-C) "cancella file temporanei"...
    - **Non e' possibile intercettare** SIGKILL o SIGSTOP.
  - Default: Eseguire l'azione di default.

Azioni associate a segnali (I)

Ignorare il segnale

- Alcuni segnali che non possono essere ignorati: **`SIGKILL`** e **`SIGSTOP`**
  - Motivo: permettere al superutente di terminare processi
  - Segnali hardware: comportamento non definito in POSIX se ignorati

Esecuzione dell'azione di default

- Per molti segnali "critici", l'azione di default consiste nel terminare il processo
- Può essere generato un file di core (eccetto quando bit set-user-id e set-group-id settati e uid/gid sono diversi da owner/group o mancano di permessi in scrittura per la directory il core file e' troppo grande)

Azioni associate a segnali (II)

Catturare ("catch") il segnale:

- Il kernel informa il processo chiamando una funzione specificata dal processo stesso (*signal handler*)
- Il signal handler gestisce il problema nel modo più opportuno

Esempio:

- nel caso del segnale **`SIGCHLD`** (terminazione di un child)
  → possibile azione: eseguire **`waitpid`**
- nel caso del segnale **`SIGTERM`** (terminazione standard)
  → possibili azioni: rimuovere file temporanei, salvare file

Alcuni segnali

nome	*significato*	*default*
SIGINT	interruzione da tastiera (Ctrl-c)	terminare
SIGSTOP*	stop al processo	*fermare*
SIGKILL*	terminazione forzata	terminare
SIGQUIT	quit da tastiera (Ctrl-y)	terminare
SIGTERM	terminazione da tastiera (Ctrl-\)	terminare
SIGCHLD	figlio terminato o fermato	*ignorare*
SIGALRM	suona la sveglia!	terminare
SIGSEGV	segmentation fault	terminare
SIGUSR1	a disposizione dell'utente	terminare

Alcuni segnali

`SIGABRT` (Terminazione, core)

- Generato da syscall `abort()`; terminazione anormale

`SIGALRM` (Terminazione)

- Generato da un timer settato con la syscall `alarm()` o la funzione `setitimer()`

`SIGBUS` (Non POSIX; terminazione, core)

- Indica un hardware fault (definito dal s.o.)

- **`SIGCHLD` (Default: ignore)**
  - Quando un processo termina, `SIGCHLD` viene spedito al processo parent
  - Il processo parent deve definire un signal handler che chiami `wait()` o `waitpid()`
- **`SIGFPE` (Terminazione, core)**
  - Eccezione aritmetica, come divisioni per 0
- **`SIGHUP` (Terminazione)**
  - Inviato ad un processo se il terminale viene disconnesso

Alcuni Segnali

`SIGILL` (Terminazione, core)

- Generato quando un processo ha eseguito un'azione illegale

`SIGINT` (Terminazione)

- Generato quando un processo riceve un carattere di interruzione (**`ctrl-c`**) dal terminale

`SIGIO` (Non POSIX; default: terminazione, ignore)

- Evento I/O asincrono

- **`SIGKILL` (Terminazione)**
  - Maniera sicura per uccidere un processo
- **`SIGPIPE` (Terminazione)**
  - Scrittura su pipe/socket in cui il lettore ha terminato/chiuso
- **`SIGSEGV` (Terminazione, core)**
  - Generato quando un processo esegue un riferimento di memoria non valido

Alcuni Segnali

`SIGUSR1`, `SIGUSR2` (Terminazione)

- Segnali non definiti utilizzabili a livello utente

`SIGSTP` (Default: stop process)

- Generato quando un processo riceve un carattere di suspend (**`ctrl-z`**) dal terminale

- **`SIGSYS` (Terminazione, core)**
  - Invocazione non valida di system call
  - Esempio: parametro errato
- **`SIGTERM` (Terminazione)**
  - Segnale di terminazione normalmente generato dal comando **`kill`**
- **`SIGURG` (Non POSIX; ignora)**
  - Segnala il processo che una condizione urgente è avvenuta (dati out-of-bound ricevuti da una connessione di rete)

Catturare i segnali

- Un handler (gestore) è una funzione del tipo:

```
void funzione(int num_segnale) {
    printf(“%d”, num_segnale);
}
```

Una volta che l'handler termina, si torna al punto in cui il programma era stato interrotto.

Catturare un segnale

```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

- signal(SIGINT, foo) imposta la funzione foo come handler del segnale SIGINT
- si puo' anche richiedere di ignorare il segnale
  - signal(SIGINT, SIG_IGN)
- oppure ritornare alla reazione di default
  - signal(SIGINT, SIG_DFL)

Catturare un segnale

```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

- restituisce l'impostazione precedente, cioe' uno dei seguenti valori:
  - l'indirizzo dell'handler precedente
  - SIG_DFL: reazione di default
  - SIG_IGN: ignorare il segnale
  - SIG_ERR: errore

Catturare un segnale

```
typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

- Lo stesso signal handler puo' gestire piu' segnali.
  - Questo e' il motivo per cui prende in input un intero, la codifica del segnale.
- E' sufficiente utilizzare ogni volta la signal indicando uno per volta tutti i segnali da gestire:

```
signal(SIGUSR1, foo);

signal(SIGUSR2, foo);
```

Esempio

```
void foo(int num_segnale);
int main(void){

    signal(SIGUSR1, foo);
    signal(SIGUSR2, foo);
    signal(SIGINT, foo);
    if (signal(SIGKILL, foo)==SIG_ERR) // Errore Certo!
        perror("Impossibile intercettare SIGKILL");
    for (;;){pause();}

}
void foo(int num_segnale) {
    if (num_segnale==SIGINT) // Impossibile bloccare con “CTRL-C”
      printf("Segnale INT %d\n", num_segnale);
    if (num_segnale==SIGUSR1)
        printf("Segnale USR1 %d\n", num_segnale);
    if (num_segnale==SIGUSR2)
        printf("Segnale USR2 %d\n", num_segnale);
}
```

Esempio

```
lso:>./a.out &
[3] 2043
Impossibile intercettare SIGKILL: Invalid argument
lso:> kill -SIGUSR1 2043
Segnale USR1 10
lso:> kill -SIGUSR1 2043
Segnale USR1 10
lso:> kill -SIGUSR2 2043
Segnale USR2 12
lso:> fg
./a.out
(CTRL-C)
Segnale INT 2
(CTRL-C)
Segnale INT 2
(CTRL-Z)
[3]+  Stopped                 ./a.out
lso:> kill -SIGKILL 2043
[3]+  Killed                  ./a.out
```

Inviare segnali

- I segnali si inviano ai processi
  - oppure a gruppi di processi identificati da un process group.
- Un processo si individua usando il suo *process id* (pid), che e' un intero non negativo
  - i pid 0 ed 1 sono riservati al sistema
- Per vedere i pid dei vostri processi correnti, digitare (ad esempio) “ps -u”, e leggere la seconda colonna

Inviare segnali

- Per inviare un segnale, bisogna averne il permesso
- In pratica, si possono inviare segnali solo ai propri processi

Inviare segnali dalla shell

comando: **kill** [-<segnale>] <pid>
oppure: **kill** -l

- kill -INT 127 invia il segnale SIGINT al processo il cui pid e' 127
- kill -l elenca tutti i segnali ed i loro valori numerici
- kill 127 equivale a kill -TERM 127

Inviare segnali in C

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

- kill(127, SIGINT) invia il segnale SIGINT al processo il cui pid e' 127
- restituisce 0 in caso di successo e -1 in caso di errore

Inviare segnali in C

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

- Il parametro pid puo assumere i seguenti valori:
  - pid>0: Identifica il processo con PID=pid
  - pid=0: Tutti i processi con group ID pari al group ID del process che esegue la kill.
  - pid<0: Tutti i processi con group id pari al valore assoluto di pid.
  - pid=-1 Inviato a tutti i processi del sistema per cui il processo che esegue la kill ha il permesso di inviare un segnale.

Inviare segnali in C

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

- Il parametro sig puo assumere i seguenti valori:
  - sig>0: E' un intero specificato in signal.h.
  - sig=0: E' utilizzato per verificare se il processo ha i permessi per inviare un segnale al/i processo/i specificati da pid.
    - Nessun segnale viene inviato
    - Utile per verificare l'esistenza di un processo.
    - Attenzione: UNIX ricicla i pid!

Impostare una sveglia

unsigned int **alarm**(unsigned int seconds);

- alarm(30) prenota un segnale SIGALRM, che sara' inviato tra 30 secondi
- restituisce 0 se non c'era nessuna sveglia gia' prenotata
- altrimenti, restituisce il tempo rimanente perche' la vecchia sveglia suonasse
- alarm(0) cancella la prenotazione precedente.

Impostare una sveglia

unsigned int **alarm**(unsigned int seconds);

- Esiste un'unica “sveglia” per processo
- Puo' trascorrere un tempo “indefinito” da quando il kernel genera il segnale fino all'esecuzione del handler.
- Attenzione: L'azione di default di SIGALRM e' la terminazione.
  - Definire l'handler prima di eseguire l'alarm

Impostare una sveglia

unsigned int **alarm**(unsigned int seconds);

- Esiste un'unica “sveglia” per processo
- Puo' trascorrere un tempo “indefinito” da quando il kernel genera il segnale fino all'esecuzione del handler.
- alarm(0) cancella la prenotazione precedente.

Esempio:

```
void foo(int num_segnale);
int main(void){

   int n=0;    int buf[100];
   alarm(5);
   signal(SIGALRM,foo);

   while (n<=0){
      printf("Digitare qualcosa:\n");
      alarm(1);
      if ((n=read(STDIN_FILENO,buf,10))<0)
         perror("Read error");
      alarm(0);
   }
}
void foo(int num_segnale) {
   alarm(1);
   printf("Vuoi muoverti a digitare qualcosa ???\n");
}
```

Insiemi di segnali

```
#include <signal.h>
 int sigemptyset(sigset_t *set);
 int sigfillset(sigset_t *set);
 int sigaddset(sigset_t *set, int signum);
 int sigdelset(sigset_t *set, int signum);
```

Ritornano: O se OK, -1 su errore

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

Ritorna 1 se vero, 0 se falso, -1 su errore

- In alcuni casi e' necessario definire un insieme di segnali
  - Per indicare al kernel quali segnali “bloccare”
- Non e' possibile rappresentare tutti i segnali in un unico intero
  - Troppi segnali disponibili.
- Per questo motivo POSIX definisce il tipo sigset_t

Insiemi di segnali

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
```

Ritornano: O se OK, -1 su errore

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

Ritorna 1 se vero, 0 se falso, -1 su errore

- sigemptyset: Tutti i segnali sono esclusi da *set
- sigfillset: Tutti i segnali sono includi in *set
- sigaddset: Aggiunge il segnale signum a *set
- sigdelset: Rimuove signum da *set
- sigismember: Verifica se signum appartiene a *set.

sigprocmask

#include <signal.h>

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

Ritorna: 0 se Ok; -1 su errore

- La “signal mask” identifica un insieme di segnali bloccati dal processo.
- sigprocmask consente di leggere, modificare od eseguire entrambe le operazioni sulla maschera dei segnali.
- Se oldset e' non nullo, conterra' la “vecchia” maschera
- Se set e' non nullo, la nuova maschera viene calcolata in base i parametri set e how

sigprocmask

- how puo' assumere i valori:
  - SIG_BLOCK: La nuova maschera e' l'unione (in senso “algebrico”) tra (l'insieme rappresentato dal)la vecchia maschera e la nuova
  - SIG_UNBLOCK: La nuova maschera e' l'intersezione (in senso algebrico) tra la vecchia maschera e la nuova.
  - SIG_SETMASK: La nuova maschera e' uguale alla maschera definita dal parametro sig.

Esercizio

- Usando il comando kill della shell, inviare il segnale SIGINT ai processi con pid 1 e 2.

- Lanciare un editor di testi (pico), metterlo in background (CTRL-Z) e poi terminarlo inviandogli il segnale SIGTERM.

Esercizio

- Scrivere un programma C “aspetta.c”, che scrive un messaggio su standard output ogni volta che riceve i segnali SIGINT o SIGUSR1.
- Il programma non deve mai terminare spontaneamente.
- Lanciare il programma. Usando il comando kill della shell, provare ad inviargli quei due segnali, ed infine terminarlo.