Laboratorio di sistemi operativi
A.A. 2010/2011
Gruppo 2
Gennaro Oliva
18
Segnali

Segnali

- I segnali sono interrupt software che forniscono una tecnica rudimentale di comunicazione asincrona tra processi
- I segnali ci consentono di gestire eventi asincroni quali ad esempio
  - un utente che interrompe l'esecuzione di un programma utilizzando Ctrl+c
  - la fine prematura di un processo in una pipeline

L'insieme di segnali

- I segnali costituiscono un insieme ben definito ed a ciascun di essi è associata una costante simbolica ed un intero positivo
- I sistemi Linux prevedono 64 segnali diversi:
- 1-31: segnali standard
- 32-63: segnali “real-time”
- `0`: utilizzato per controllare se si hanno I permessi sufficienti per inviare un segnale ad un processo
- Le costanti simboliche hanno tutte il prefisso SIG, sono definite nell'header signal.h e descritte nella pagina di manuale signal(7)

Condizioni che generano segnali

- Distinguiamo le seguenti categorie di segnali in base alla loro origine
  - Segnali provenienti dal terminale: generati quando l'utente utilizza alcune combinazioni di tasti quali Control-C (SIGINT) o Control-Z (SIGSTOP)
  - Segnali derivanti da eccezioni hardware: generati dall'hardware quando si verificano eventi quali la divisione per `0` o il tentativo di accesso ad un'area di memoria non valida; gli eventi vengono rilevati dal kernel che invia il segnale corrispondente al processo che ha causato le eccezioni (ad esempio SIGSEGV)
  - Segnali provenienti dalla funzione kill(2) (e dal comando kill(1)): generati dalla system call kill verso un determinato PID
  - Segnali collegati ad eventi software: generati quando accade un evento di cui un processo deve essere informato; è il caso, ad esempio, di un processo che scrive in una pipeline mentre il processo che legge è già terminato (ps -e | head); in tale ipotesi il primo riceve un segnale SIGPIPE quando il secondo termina

Gestione dei segnali

- La generazione di un segnale è la sua notifica passano attraverso il kernel che li gestisce utilizzando una coda di segnali pendenti
- La notifica è un evento asincrono rispetto alla generazione e dipende dal kernel
- Sui sistemi Linux un segnale standard (non real time) viene inserito nella coda dei segnali pendenti solo se non è già presente
- In pratica inviando n volte lo stesso segnale standard ad un processo: la prima istanza viene inserita in coda, mentre quelle successive vengono scartate fintanto che la prima è in coda

Gestione di segnali

- Il processo che riceve il segnale può decidere di gestirlo in 3 modi:
  - ignorare il segnale
  - catturare il segnale specificando una funzione detta signal handler da eseguire
  - eseguire l'azione di default associata al segnale (molto spesso la terminazione del processo)
- I segnali SIGKILL e SIGSTOP non possono essere ne catturati ne ignorati ma eseguono sempre l'azione di default
- Le azioni di default sono specificate nella pagina di manuale signal(7)

Alcuni Segnali

Nome	Significato	Azione di default
SIGINT	Interruzione da tastiera CTRL+c	Terminazione processo
SIGQUIT	Quit da tastiera (produce core dump)	Terminazione processo
SIGTERM	Terminazione da tastiera	Terminazione processo
SIGKILL	Terminazione forzata	Terminazione processo
SIGSTOP	Interruzione del processo	Interruzione processo
SIGCONT	Continua processo interrotto	Continuazione processo
SIGCHLD	Processo figlio terminato	Segnale ignorato
SIGALRM	Intervallo di tempo passato	Terminazione processo
SIGSEGV	Segmentation fault	Terminazione processo
SIGUSR1	A disposizione dell'utente	Terminazione processo
SIGUSR2	A disposizione dell'utente	Terminazione processov

Permessi per l'invio di segnali

- Dal momento che l'effetto che otteniamo inviando un segnale molto spesso è la terminazione del processo ed essendo UNIX un ambiente multi utente è necessario stabilire regole di protezione
- Per inviare un segnale utilizzando la system call kill (ed il comando kill) è necessario che l'uid (real o effective) del processo mittente sia uguale all'uid (real o effective) del processo destinatario
- L'utente root può inviare segnali a tutti i processi

Il comando kill

- Il comando:

  ```
  $ kill -signal n
  ```

- Invia il segnale signal al processo con pid n
- Se non si specifica nessun segnale viene inviato SIGTERM
- Il comando:

  ```
  $ kill -l
  ```

- elenca tutti i segnali ed i loro valori numerici
- Esempio d'uso:

  ```
  $ kill -SIGINT 1722
  $ kill -INT 1722
  $ kill -2 1722
  ```

Il comando pkill

- Ci consente di inviare un segnale ad un processo specificando il nome del programma piuttosto che il pid

```
$ pkill -9 a.out
```

- Invia il segnale SIGTERM a tutte le istanze di a.out a cui ho il permesso di inviare segnali

La system call kill

- Il comando kill utilizzanda la system call:

  int kill(pid_t pidn, int sig);

- che consente di inviare il segnale sig al processo con pid pidn
- Specificando -1 come pidn, il segnale viene inviato a tutti i processi per i quali si ha il permesso
- La system call restituisce `0` in caso di successo e -1 in caso di errore

Catturare un segnale

- Nella realizzazione di un programma è possibile definire, per tutti i segnali catturabili, un comportamento diverso da quello previsto di default
- Quest'operazione viene effettuata associando ad un segnale una funzione speciale chiamata handler (gestore) che ha tipo void ed accetta come unico parametro un intero

  Esempio di signal handler:

```
void funzione(int num_segnale) {
    printf("Segnale %d ricevuto \n", num_segnale);
}
```

- All'atto della notifica del segnale il sistema operativo invoca l'handler passandogli come argomento il valore numerico del segnale catturato
- In questo modo uno stesso signal handler può essere utilizzato per gestire più segnali utilizzando il parametro passato per differenziare il comportamento all'interno della funzione
- Una volta eseguito l'handler, il processo riprende l'esecuzione dal punto in cui era stato interrotto

signal

- Per associare un handler ad un segnale è possibile utilizzare la system call:

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

sighandler_t signal(int signo, sighandler_t hand);
```

  Dove signo è il segnale da associare; mentre hand è un puntatore alla funzione che funge da handler oppure una delle seguenti costanti:

  - SIG_DFL per impostare come handler l'azione di default
  - SIG_IGN per ignorare il segnale

- La signal restituisce il gestore precedentemente associato al segnale: ovvero l'indirizzo della funzione handler, SIG_DFL, SIG_IGN oppure SIG_ERR in caso di errore

alarm

- La system call:

  unsigned int alarm(unsigned int sec);
- richiedere al sistema operativo di inviare il segnale SIGALRM al processo dopo sec secondi, oppure di cancellare una richiesta pendente se sec=0
- alarm restituisce 0 se non c'è una precedente sveglia pendente, oppure il numero di secondi che restavano alla sveglia precedentemente impostata in caso contrario
- Non è possibile impostare più di una sveglia per processo
- La generazione del segnale non implica l'esecuzione immediata dell'handler, ma può trascorrere un intervallo di tempo "indefinito" tra i due eventi
- L'azione di default di SIGALRM è la terminazione quindi è opportuno definire un handler prima di eseguire l'alarm

pause

- La system call

  int pause(void);

- sospende l'esecuzione di un processo ponendolo in attesa di un segnale
- pause() ritorna solo quando la funzione associaa all’handler di un segnale termina
- pause() non ritorna se il segnale viene ignorato, ma soltanto se il segnale è gestito o causa la terminazione del processo

Esempio d'uso

```
#include<stdio.h>
#include <unistd.h>
#include<signal.h>
void gestisci (int s) {
  switch (s) {
    case SIGINT:
      printf("SIGINT intercettato\n");
      break;
    case SIGUSR1:
      printf("SIGUSR1 intercettato\n");
      break;
    default:
      printf("Non gestisco questo segnale\n");
  }
}
int main(void){
  signal(SIGUSR1, gestisci);
  signal(SIGINT, gestisci);
  while (1) pause();
}
```