5. SEGNALI

Un segnale è un interrupt software e può essere generato da un processo utente o dal kernel a seguito di un errore software o hardware.
Ogni segnale ha un nome che comincia con SIG (ex. SIGABRT, SIGALARM) a cui viene associato una costante intera (≠0) positiva definita in **signal.h**.
Il segnale è un evento asincrono, esso può arrivare in un momento qualunque ad un processo ed il processo può limitarsi a verificare, per esempio, il valore di una variabile, o può fare cose più specifiche.
Quando arriva un segnale da un processo possono accadere 3 cose:

1. **Ignorare il segnale** (tranne che per SIGKILL e SIGSTOP);
2. **Catturare il segnale** (equivale ad associare una funzione utente quando il segnale occorre; ex. se il segnale SIGTERM è catturato possiamo voler ripulire tutti i file temporanei generati dal processo);
3. **Eseguire l'azione di default associata** (terminazione del processo per la maggior parte dei segnali).

5.1 SIGNAL

```
#include <signal.h>
void (*signal(int signo, void (*func)(int)))(int);
```

Restituisce SIG_ERR in caso di errore e il puntatore al precedente gestore del segnale se OK.
Prende due argomenti: il nome del segnale ***signo*** ed il puntatore alla funzione ***func*** da eseguire all'arrivo di signo (signal handler) e restituisce il puntatore ad una funzione che prende un intero e non restituisce niente, che rappresenta il puntatore al precedente signal handler.
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'esecuzione di un programma tramite **fork+exec** ha le seguenti caratteristiche

- Se un segnale è ignorato nel processo padre viene ignorato anche nel processo figlio;
- Se un segnale è catturato nel processo padre viene assegnata l'azione di default nel processo figlio.

5.2 KILL E RAISE

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int signo);
int raise (int signo);
```

Mandano il segnale signo specificato come argomento e restituiscono 0 se OK, -1 in caso di errore.
Raise: consente ad un processo di mandare un segnale a sé stesso.
Kill: manda un segnale ad un processo o ad un gruppo di processi specificato da pid, quest'ultimo può avere valori:

- pid > 0 invia al processo pid;
- pid == 0 invia ai processi con lo stesso gid del processo sender;
- pid < 0 invia ai processi con gid uguale a |pid|.

5.3 ALARM

```
#include <unistd.h>
unsigned int alarm (unsigned int secs);
```

Invia al processo corrente il segnale SIGALRM (uccide il processo) dopo che siano trascorsi secs secondi e restituisce 0 o il numero di secondi rimasti da una precedente alarm.

5.4 PAUSE

```
#include <unistd.h>
int pause(void);
```

Sospende il processo finché non arriva un segnale, il corrispondente signal handler viene eseguito ed esce, e restituisce –1.

5.5 SLEEP

```
#include <unistd.h>
unsigned int sleep (unsigned int secs);
```

Restituisce 0 oppure, nel caso la funzione sia risvegliata da un segnale, il numero di secondi rimasti.

Come prima cosa nel main è la chiamata alla signal, proprio perché se si vuole gestire l'arrivo di un segnale con una funzione bisogna farlo il prima possibile perché non si sa quando arriverà tale signale.
Ora se all'interno di tale processo arriverà uno dei due segnali esplicitati, allora verrà eseguita la funzione specificata.

```
[geronimo@fujitsu SO]$ ./sign1 &
[1] 73043
[geronimo@fujitsu SO]$ ps
    PID TTY          TIME CMD
  71359 pts/0    00:00:00 bash
  73043 pts/0    00:00:00 sign1
  73066 pts/0    00:00:00 ps
[geronimo@fujitsu SO]$ kill -USR1 73043
SIGUSR1
[geronimo@fujitsu SO]$ kill -USR2 73043
SIGUSR2
[geronimo@fujitsu SO]$ kill -2 73043
[geronimo@fujitsu SO]$ ps
    PID TTY          TIME CMD
  71359 pts/0    00:00:00 bash
  73225 pts/0    00:00:00 ps
[1]+  Interruzione              ./sign1
```

```
#include <signal.h>
void sig_usr(int);
int main (void) {
  signal(SIGUSR1, sig_usr);
  signal(SIGUSR2, sig_usr);
  for(; ;)  pause();

 }
void sig_usr(int signo) {
  if(signo == SIGUSR1) printf("SIGUSR1\n");
  else if (signo == SIGUSR2) printf("SIGUSR2\n");
    else printf("Segnale %d\n",signo);
}
```