

segnali

- I segnali rappresentano un metodo per sincronizzare i processi e per gestire eventi asincroni
- un processo può collegare (catch) una funzione alla presentazione di un segnale
- un processo può eseguire una segnalazione ad un altro processo o a se' stesso
- esistono segnali particolari come, ad esempio, quello generato dalla pressione di ctrl-C
- un segnale è identificato da un numero intero; i segnali sono codificati con nomi mnemonici in `signal.h`
- più segnalazioni dello stesso segnale possono non corrispondere ad altrettante invocazioni della funzione collegata

catch

- il collegamento di una funzione action ad un segnale signum si ottiene con la primitiva:

```
sighandler_t signal(int signum, sighandler_t  
    action)
```

- action è un puntatore a funzione:
- typedef void (*sighandler_t) (int);
- che viene invocata passando come parametro intero il numero corrispondente al segnale; in questo modo è possibile scrivere un unico handler per più segnali.
- **ATTENZIONE:** gli handler non sono protetti da rientranza, quindi è possibile ricevere un segnale mentre se ne sta processando un altro.

```

/*****
 * $Id: es50.c,v 1.2 2002/03/08 09:04:59 valealex Exp $
 * Esempio di signal
 * *****/
#include <signal.h>
#include <stdio.h>

#define MAXSIGNALS 10
#define BSD_SIGNALS

int main(int argc, char **argv);
void sighandler(int sigid);
int main(int argc, char **argv) {
    /* Collega la funzione sighandler() al segnale SIGINT.
     * Questo segnale si genera premendo CTRL-C dalla console
     * alla quale e' collegato il processo (controlling tty)
     * ovvero, normalmente, da tastiera */
    signal(SIGINT,sighandler);
    printf("Handler set\n");
    for (;;) {
        /* forever loop */
        ;
    }
}

```

```

void sighandler(int sigid) {
    static int cnt=0;
    cnt++;
    printf("signal received: %d, %d times\n",sigid,cnt);
#ifdef BSD_SIGNALS
    /* comportamento: System V
     * quando arriva il segnale, si rimuove il collegamento
     * con l'handler -> bisogna ripristinarlo */
    signal(SIGINT,sighandler);
#endif
    if(cnt>=MAXSIGNALS) {
        /* uso l'handler di default: terminazione */
        signal(SIGINT,SIG_DFL);
    } else {
        return;
    }
}

```

generazione di segnali temporizzati

- Uno degli usi ‘tradizionali’ dei segnali è quello di gestire le condizioni di allarme o timeout.
- per questo scopo è definito un segnale (SIGALRM) ed una funzione di schedulazione (alarm(int secondi))

```
int main(int argc, char **argv) {
    signal(SIGALRM, sighandler);
    alarm(10);
    printf("alarm set\n");
    for (;;) {
        ;
    }
}

void sighandler(int sigid) {
    printf("signal received: %d\n", sigid);
    exit(0);
}
```

non local (long) jump

- Spesso può essere conveniente assegnare ad un signal handler il compito di interrompere un loop del processo.
- In questi casi:
 - o si utilizza una flag di uscita da verificare continuamente nel loop
 - o si utilizza un meccanismo di salto
- Nel secondo caso si ricorre a due funzioni della libc:
 - `setjmp()` ‘marca’ lo stato corrente
 - `longjmp()` ‘salta’ alla posizione della `setjmp`
- il valore di ritorno della `setjmp` indica se si tratta dell’invocazione di ‘set’ (zero) o del ritorno da `longjmp` (non zero)
- queste funzioni necessitano di una variabile di tipo `jmp_buf` per memorizzare e ripristinare lo stato del processo.

```

/*****
 * $Id: es52.c,v 1.1 2002/03/05 21:29:10 alex Exp $
 * Esempio di alarm con setjmp/longjmp
 * *****/
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <setjmp.h>
static jmp_buf env;
int main(int argc, char **argv);
void sighandler(int sigid);
int main(int argc, char **argv) {
    signal(SIGALRM, sighandler);
    if (setjmp(env)) {
        /* NZ -> jump */
        printf("Here by jump\n");
        exit(0);
    }
    /* Zero -> no jump */
    alarm(10);
    printf("alarm set\n");
    for (;;) {
        ;
    }
}

```

```
void sighandler(int sigid) {  
    printf("signal received: %d\n",sigid);  
    longjmp(env,0);  
}
```

generazione di segnali

- La generazione di un segnale da parte di un processo a se' stesso si ottiene con la primitiva:

```
int raise(int signum);
```

- Questa primitiva fallisce solamente se signum non è un numero valido per un semaforo
- La segnalazione di un semaforo ad un altro processo di cui si conosce il pid e del quale si condivide lo user id si ottiene con:

```
int kill(pid_t pid, int signum)
```
- Con kill è anche possibile eseguire un broadcast del semaforo a *gruppi di processo*, ad esempio a tutti i processi che condividono una sessione di terminale.

```

/*****
 * $Id: es53.c,v 1.2 2002/03/08 13:07:35 valealex Exp $
 * Esempio di kill a se stesso
 * *****/
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <setjmp.h>
#include <sys/types.h>

static jmp_buf env;

int main(int argc, char **argv);
void sighandler(int sigid);
int main(int argc, char **argv) {
    /* colleghiamo il segnale SIGUSR1 */
    signal(SIGUSR1,sighandler);
    if (setjmp(env)) {
        /* NZ -> jump */
        printf("Here by jump\n");
        exit(0);
    }
    /* Zero -> no jump */
    printf("send SIGUSR1\n");
    kill(getpid(),SIGUSR1);
}

```

```
    /* equivalente a: */  
    /* raise(SIGUSR1); */  
    printf("SIGUSR1 sent\n"); /* never here */  
    pause();  
    exit(0);  
}  
void sighandler(int sigid) {  
    printf("signal received: %d\n",sigid);  
    longjmp(env,0);  
}
```

mascheramento di segnali

- In certe condizioni è opportuno proteggere il programma da eventi asincroni, ad esempio durante la modifica di dati condivisi o durante il processamento di altri segnali
- in queste condizioni è possibile richiedere al sistema operativo l'oscuramento dei segnali con la primitiva:

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

- dove *how* indica l'operazione richiesta:
 - SIG_BLOCK aggiunge i segnali del set fra quelli bloccati
 - SIG_UNBLOCK rimuove l'eventuale blocco per i segnali del set
 - SIG_SETMASK maschera solo ed esclusivamente i segnali del set
- set e oldset sono puntatori a strutture sigset_t

signal set

- Le operazioni di mascheramento si appoggiano a strutture di tipo `sigset_t`
- La libc mette a disposizione delle funzioni per riempire queste strutture con i vari segnali:
- `sigemptyset(&set)` inizializza il set vuoto
- `sigfillset(&set)` inizializza il set con tutti i segnali
- `sigaddset(&set,signal)` aggiunge il segnale al set
- `sigdelset(&set,signal)` rimuove il segnale dal set
- `sigismember(&set,signal)` verifica se il segnale è nel set

```

/*****
 * $Id: es55.c,v 1.1 2002/03/08 13:07:53 valealex Exp $
 * Blocco di segnali
 * *****/
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv);

static sigset_t set,old;

int main(int argc, char **argv) {
    sigemptyset(&set);
    sigaddset(&set,SIGINT);
    sigprocmask(SIG_BLOCK,&set,&old);
    /* il segnale CTRL-C e' bloccato */
    printf("Esco fra 20 secondi\n");
    sleep(20);
    exit(0);
}

```

Esercizio in Laboratorio

- Progettare e realizzare un programma concorrente (es54.c) in C composto da due processi.
- I processi si devono sincronizzare fra loro in modo esclusivo mediante lo scambio di segnali
- I due processi, alternativamente, scrivono sullo standard output due stringhe: “padre\n” per il processo padre e “figlio\n” per il processo figlio.
- Il primo processo a scrivere è il figlio.
- Facoltativo: estendere a ‘n’ (passato sulla command line) il numero di processi che si alternano il controllo, ognuno dei quali provvede a scrivere il proprio pid su standard output.