Lezione 11: Segnali(2)

11.1 Segnali affidabili

Uno dei problemi più insidiosi che può verificarsi quando si gestisce un segnale è l'occorrenza di un secondo segnale mentre è in esecuzione la funzione di gestione del segnale, il secondo segnale può essere di un tipo diverso rispetto a quello correntemente in gestione, od anche dello stesso tipo.

è necessario prendere delle precauzioni all'interno della funzione di gestione del segnale per evitare race, unix contiene alcune caratteristiche che consentono di bloccare l'eventuale elaborazione dei segnali.

11.2 Maschera dei segnali

La chiamata di sistema posix usata per mascherare i segnali sigprocmask()

Consente di specifiacare un insieme di sengali da blcocare e restituisce la lista dei segnali che erano bloccati precedentemente, utile per ripristinare lo stato della maschera precedente, una volta completata la nostra sezione critica.

11.2.1 sigprocmask()

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

int how: definisce se aggiungere segnali alla maschera corrente del processo SIG_BLOCK rimuoverli dalla maschera corrente SIG_NBLOCK o sostituire completamente la maschera corrente con una nuoca SIG_SETMASK

const sigset_t *set l'insieme dei segnali da bloccare, aggiungere o rimuovere dalla maschera corrente sull abase del parametro how.

sigset_t*oset se non è null conterrà la maschera precedente, dopo aver invocato sigprocmasck, se qualche segnale non bloccato è pendente, almeno uno di tali segnali sarà consegnato al processo prima che sigprocmask ritorni.

11.3 Il tipo di dato sigset_t

Tipicamente i segnali possono essere messi in corrispondenza con ciascun bit di un intero, il numero di segnali differenti che possono occorrere può superare il numero di bit in un intero. In generale potremmo non usare un intero per rappresentare l'insieme con un bit per sengale. Posix definisce il tipo di dato sigset_t per contenere un insieme di segnali ed un insieme di funzioni per manipolarlo.

11.4 Insieme di segnali

```
#include<signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
//Ritornano 0 se OK, -1 in caso di errore
int sigismember(const sigset_t *set, int signo);
//Ritorna 1 se vero, O se falso, -1 in caso si errore
```

11.4.1 Esempio 1

```
/* definisce una nuova maschera */
sigset_t mask_set;

/* svuota la maschera */
sigemptyset(&mask_set);

/* aggiunge i segnali TSTP e INT alla maschera */
sigaddset(&mask_set, SIGTSTP);
sigaddset(&mask_set, SIGINT);

/* rimuove il segnale TSTP dall’insieme */
sigdelset(&mask_set, SIGTSTP);

/* controlla se il segnale INT è definito nell’insieme */
if (sigismember(&mask_set, SIGINT)
    printf(“segnale INT nell’insieme\n");
else
    printf("signale INT non nell’insieme\n");

/* pone tutti i segnali del sistema nell’insieme */
sigfillset(&mask_set)
```

11.4.2 Esempio 2

Vediamo un breve esempio di codice che conta il numero di segnali ctrl c digitati dall’utente

```
/* Definiamo il contatore di Ctrl-C counter inizializzato a 0 */
int ctrl_c_count = 0;
#define CTRL_C_THRESHOLD 5

void catch_int(int sig_num) { /* gestore segnale Ctrl-C */
    sigset_t mask_set; /* per impostare la maschera dei segnali
    sigset_t old_set; /* memorizza la vecchia maschera */

    /* maschera gli altri segnali mentre siamo nel gestore*/
    sigfillset(&mask_set);
    sigprocmask(SIG_SETMASK, &mask_set, &old_set);


ctrl_c_count++; /* incrementa count, e controlla la soglia */
if (ctrl_c_count >= CTRL_C_THRESHOLD) {
    char answer[30];
    printf("\nVuoi terminare? [s/N]: ");
    fflush(stdout);
    gets(answer);
    if (answer[0] == ‘s' || answer[0] == ‘S') {
        printf("\nTermino...\n");
        fflush(stdout);
        exit(0); }
    else {
        printf("\nContinuo\n");
        fflush(stdout);
        /* reset del contatore per Ctrl-C */
        ctrl_c_count = 0; }
```

```
    }
}

void catch_suspend(int sig_num) {
    sigset_t mask_set; /* usato per mascherare i segnali */
    sigset_t old_set; /* memorizza la vecchia maschera */
    sigfillset(&mask_set);
    sigprocmask(SIG_SETMASK, &mask_set, &old_set);
    /* stampa il contatore corrente per Ctrl-C */
    printf("\n\n'%d' Ctrl-C digitazioni\n", ctrl_c_count);
    fflush(stdout);
    }
    /* da qualche parte nel main... */ . .
    /* imposta i gestori per the Ctrl-C e Ctrl-Z */
    signal(SIGINT, catch_int);
    signal(SIGTSTP, catch_suspend);
    . . /* ed il resto del programma */ . .
```

11.5 Evitare la race condition

L'uso di `sigprocmask()` nell'esempio non risolve tute le possibili race condition ad esempio, dopo l'entrata nel gestore del segnale, ma prima di invocare `sigprocmask()` è possibile ricevere un altro segnale, che sarà gestito. Pertanto se l'utente è molto veloce, si crea una race.

Il modo per evitare completamente race, è consentire al sistema di impostare la maschera dei segnali prima che sia chiamato il gestore del segnale. Ciò può essere invocando la SC `sigaction()` per definire sia la funzione di gestione del segnale che la maschera dei segnali da usare quando è eseguito il gestore.

11.6 Sigaction

Consente di esaminare o modificare l'azione associata con un particolare segnale, questa dunzione sostituisce la funzione signal nelle versioni più recenti di unix.

```
#include<signal.h>
int sigaction(int signo, const struct sigaction *act, struct sig
```

signo è il numero del segnale la cui azione stiamo esaminando o modificando.

Se il puntatore **act** è non nullo, stiamo modificando l'azione, se **oact** è non nullo, il sistema ritorna l'azione precedente per il segnale attraverso il puntatore oact

11.6.1 La struttura sigaction

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_sigaction)(int, siginfo_t *,void *);
};
```

Quando si cambia l'azione per un segnale, se il campo **sa_handler** contiene l'indirizzo di una funzione per la sua gestione, il campo sa_mask specifica l'insieme dei segnali che sono aggiunti alla maschera dei segnali prima che la funzione di gestione sia chiamata.

Se e quando la funzione di gestione del segnale ritorna, la maschera dei segnali del processo è resettata al suo valore precedente.

Il SO include il segnale attualmente in consegna nella maschera dei segnali quando il gestore è invocato, in questo modo siamo sicuri che mentre stiamo elaborando un dato segnale, un'altra occorrenza di quel segnale sarà bloccata fino a che non abbiamo completato la gestione del precedente.

11.6.2 Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void signal_handler(int signum) { // Gestore dei segnali
    printf("Ricevuto il segnale %d\n", signum);
}

int main() {
    struct sigaction sa;
    sa.sa_handler = signal_handler; // Impostazione del gestore
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGINT, &sa, NULL) == -1) {// Installazione de
        perror("Errore nell'installazione del gestore del segnal
        exit(EXIT_FAILURE);
}
…
```

11.7 Interruzione e riavvio di SC

Consiferiamo il segujente scenario:

- Definiamo un handler per un dato segnale
- Facciamo una cniamata di sistema bloccante, ad esempio, una read da un terminale che si blocca fino a che non è fornito un input
- Mentre la SC è bloccata, il segnale per cui abbiamo definito il gestore è consegnato ed è invocato il suo gestore

Dopo che il gestore ritorna:

- Di default, la SC fallisce con un errore EINTR, questa caratteristica è utile.

- Spesso però preferiamo continuare l'esecuzione di una SC interrotta. Per fafre ciò possiamo usare del codice come il seguente per riavviare normalmente una SC nel caso interrotta da un gestore di segnale

11.7.1 Flag SA_RESTART

```
while (( cnt = read(fd,buf, BUF_SIZE)) ==-1 && errno == EINTR)
    Continue;
    if (cnt == -1)
        errExit("read");
```

11.8 Segnali per il controllo dei job

11.8.1 Gruppi di processi

Ogni processo è membro di un gruppo di processi, un gruppo di processi ha associato un identificatore pgid - process group id da non confondere con il grid.

Un processo figlio eredita il gruppo di appartenenza del padre.

La SC `setpgid()` permette al processo invocante di cambiare il gruppo di appartenenza a se stesso o ad altri.

11.8.2 Gruppo di processi e terminale di controllo

Ad ogni processo può essere associato un terminale di controllo: è tipicamente il terminale da cui il processo è lanciato; i figli ereditano il terminale di controllo del padre, se un processo esegue una `exec()` il terminale di controllo non cambia.

Ad ogni terminale è associato un processo di controllo, se il terminale individua un metacarattere come `<ctrl C>` spedisce il segnale appropriato a tutti i processi nel gruppo del processo di controllo.

Se un processo tenta di leggere dal suo terminale, il processo riceve un segneale SIGTTIN che normalmente lo sospende.

11.8.3 Gruppi e terminale: uso nella shell

All'avvio di una shell interattiva, se la shell esegue un comando in foregrounf, la shell figlio si mette in un diverso grupp, assume il controllo del terminale, esegue il comando così ogni segnale generato dal terminale viene indirizzato al comando e non alla shell originaria, quando il comando termina, la shell originaria riprende il controllo del terminale.

Se invece la shell esegue un comando in background, la shell figlio mette in un diverso gruppo ed esegue il comando, ma non assume il controllo del terminale. così ogni segnale del terminale continua ad essere indirizato alla shell originaria, se il comando in background tenta di leggere dal suo terminale di controllo, viene sospeso da un segnale SIGTTIN

11.8.4 Cambiare il gruppo: setpgid()

Quando un processo vuole creare un proprio gruppo di processi, distinto dagli altri gruppi del sistema tipicamente passa il proprio pid come argomento per `setgid()`

```
setpgid(0,getpid())
```

11.8.5 ottenere il gruppo: getpgid()

```
pid_t getpgid(pid_t pid)
```

11.9 pgrp1.c

```
#include <signal.h>
#include <stdio.h>

void sigHandler (int sig) {
    printf ("Process %d got a %d signal\n", getpid(), sig);
}

int main (void) {
    signal (SIGINT, sigHandler); /* Gestisce Control-C */
    if (fork () == 0)
        printf ("Child PID %d PGRP %d waits\n",getpid(),getpgid(
    else
        printf ("Parent PID %d PGRP %d waits\n",getpid(),getpgid(
        pause (); /* Aspetta un segnale */
    }
```

Il programma pgrp1.c mostra come un terminale invia i segnali ad ogni processo appartenente al gruppo di processi del suo processo di controllo

Poiché un figlio eredita il gruppo di processi del padre, padre e figlio intercettano il segnale SIGINT

11.10 pgrp2.c

Il programma pgrp2.c mostra che se un processo lascia il gruppo del processo di controllo del terminale, non riceve più segnali dal terminale

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void sigHandler (int sig) {
```

```
    printf ("Process %d got a SIGINT\n", getpid());
    exit (1);
}

int main (void) {
    int i;
    signal (SIGINT, sigHandler); /* gestore di segnali */
    if (fork () == 0)
        setpgid (0, getpid ());/* il figlio nel suo gruppo */
        printf("Process PID %d PGRP %d waits\n",getpid(),getpgi
        for (i = 1; i <= 3; i++) { /* Cicla 3 volte */
            printf ("Process %d is alive\n", getpid());
            sleep(2);
    }
    return 0;
}
```

Esercizi per casa