

Intercettazione dei segnali – assembly, Linux & fun

Pubblicato da syscalo il 08/12/2000

Livello *intermedio*

Introduzione

Intercettare i segnali inviati al nostro programma e' una delle cose da imparare se vogliamo gestire in modo intelligente le varie problematiche che si potrebbero verificare durante l'esecuzione.

Per non farne una trattazione noiosa, ho pensato di creare un piccolo programmino che potreste usare per fare uno scherzo ad un vostro amico poco esperto di Linux, molto poco esperto (capirete poi perche' specifico questo).

Come tutti sapete per inviare segnali ad un programma in esecuzione sul nostro pc si possono usare le combinazioni di tasti ^C (segnale di interrupt), ^Z (segnale di stop) e ^\ (segnale di uscita).

Lo scherzo consiste nell'intercettare questi segnali e far rispondere con frasi simpatiche (forse) invece che eseguire le operazioni di default assegnate ad ogni segnale.

Programmi usati

- *gcc*: compilatore C.

Iniziamo

Un po' di teoria:

Per intercettare i segnali e' necessario effettuare le seguenti operazioni:

- scrivere la nostra funzione che andra' a gestire il segnale.
- registrare la funzione in modo che venga chiamata al posto di quella di default quando arriva il segnale.

Le nuove funzioni che andremo a scrivere, generalmente una per ogni tipo di segnale, hanno questo prototipo:

```
void nome(void);
```

Per registrare la funzione dobbiamo chiamare la funzione `signal` che ha il seguente prototipo:

```
void (*signal(int signum, void (*handler)(int)))(int);
```

in cui i parametri rappresentano:

- `signum`: numero del segnale.
- `(* handler)(int)`: puntatore alla funzione che dovra' essere chiamata.

E' necessario prestare attenzione ad una particolarita': in Linux dopo che viene chiamata la funzione da noi scritta assegnata al segnale, viene ripristinata la funzione di default.

Per continuare ad eseguire la nostra funzione e' sufficiente richiamare opportunamente la funzione `signal` all'interno della funzione stessa.

Questa nota e' presente nella pagina di manuale "man 2 signal"; in realta' io non ho riefettuato l'assegnamento, e sono comunque rimaste settate le mie funzioni e non ripristinate quelle di default!

Provate, e se ottenete risultati differenti, come sempre fatemelo sapere.

Passiamo al codice:

Pur essendo semplice, ho aggiunto qualche commento al programma.

```
##
## file funsig.S
## gcc -o funsig funsig.S
## ./funsig
##

##
## sezione dati
##
.data

##
## valore numerico dei segnali
```

```

## reperibile in "man 7 signal"
##
.equ SIGINT, 2
.equ SIGQUIT, 3
.equ SIGTSTP, 20

## valore usato per il tloop
## ritardo di 3 secondi
.equ ritardo, 3

## le stringhe sono create per una lunghezza di 80 caratteri
presentazione:
.asciz "\n\033[36m\033[1m\033[33GFun & other...\033[m\n"

m_int:
.asciz "\033[44m\033[1m      Lasciami girare ancora un po' ;-D \
      \033[m"

m_tstp:
.asciz "\033[47m\033[1m      Non ho voglia di dormire!! ;-P \
      \033[m"

m_quit:
.asciz "\033[42m\033[1m      Ok, ok, adesso termino... che palle!! :-\\ \
      \\n      Terminazione processo in corso... \
      \033[m"

m_fun:
.asciz "\033[41m\033[1m      hehe scherzetto ;-pp \
      \033[m"

##
## sezione codice
##
.text

.global main

## implementazione delle funzioni
sig_int:
    pushl $m_int
    call puts
    addl $4, %esp
    ret

sig_tstp:
    pushl $m_tstp
    call puts
    addl $4, %esp
    ret

sig_quit:
    pushl $m_quit
    call puts
    addl $4, %esp

## ritardo di 3 secondi
    pushl $ritardo
    call sleep
    addl $4, %esp

    pushl $m_fun
    call puts
    addl $4, %esp
    ret

##
## funzione main
##
## setta la corrispondenza segnale <--> funzione
## visualizza la frase di intestazione del programma
## entra in un ciclo infinito
##

```

```

main:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx
    pushl %esi
    pushl %edi

    pushl $sig_int
    pushl $SIGINT
    call signal
    addl $8, %esp

    pushl $sig_tstp
    pushl $SIGTSTP
    call signal
    addl $8, %esp

    pushl $sig_quit
    pushl $SIGQUIT
    call signal
    addl $8, %esp

    pushl $presentazione
    call puts
    addl $4, %esp

## ciclo infinito
loop:
    jmp loop

    popl %edi
    popl %esi
    popl %ebx
    movl %ebp, %esp
    popl %ebp

    xorl %eax, %eax
    ret

```

Ed ora ecco la spiegazione del perché ho sottolineato che questo "scherzetto" può essere fatto solo ad un amico che ne sa molto poco di Linux: per terminare il programma è sufficiente loggarsi su un'altra console e dare il comando "killall funsig". Ovviamente sono molti i segnali che possono essere inviati al nostro programma, qui ne ho scelti 3 inviabili tramite combinazioni di tasti della tastiera.

Inoltre non tutti i segnali sono bloccabili o ignorabili, questo per ovvie ragioni di sicurezza; immaginate un programma che non può essere terminato in alcun modo, obbligherebbe allo spegnimento fisico della macchina!

A voi sperimentare gli altri segnali e a prendere la buona abitudine di gestirli nei vostri programmi futuri.

Conclusioni

Per la funzione signal fate riferimento a "man 2 signal".

Per una lista dei segnali e delle loro caratteristiche "man 7 signal".