Creare un modulo per il kernel

Il mio primo modulo

Nerio Da Canal
nerio@nerio.it

BLUG - Belluno Linux User Group
http://belluno.linux.it

LINUX DAY 2006 ITALIA
28 ottobre 2006
Feltre
ITI Negrelli





Prologo

"So, you want to write a kernel module. You know C, you've written a few normal programs to run as processes, and now you want to get to where the real action is, to where a single wild pointer can wipe out your file system and a core dump means a reboot." [*Peter J. Salzman*]







Moduli del Kernel perchè

- Come e' possibile realizzare un proprio modulo?
- Perche' ?
  - Per interfacciarsi ad una periferica.
- Perche' scrivere un modulo per il kernel
  - Per permettere all'utente di interagire direttamente con la periferica senza dover assegnargli privilegi di amministratore.

Da dove partire: http://www.tldp.org

IL KERNEL

Che cos'e' il Kernel? (1)

Il kernel e' il cuore del sistema operativo. E' quel codice che viene eseguito subito dopo la fase di boot. Il kernel fornisce i servizi base per tutte le altre parti del sistema operativo come la gestione della memoria, la gestione dei processi, la gestione dell'I/O sui file e su dispositivi esterni. Questi servizi sono utilizzati da altre parti del sistema operativo o da altri programmi attraverso un set specifico di program interfaces indicate come system calls.

IL KERNEL - continua

Che cos'e' il Kernel? (2)

Il contenuto di un kernel varia abbstanza da sistema operativo all'altro, ma tipicamente consiste di uno scheduler che stabilisce come i vari processi si dividono il tempo di utilizzo della CPU e in quale ordine, un interrupt handler che gestisce le richieste dalle varie periferiche (quali harddisk,tastiere,.....) e un memory manager che distribuisce la memoria fra i vari utenti del sistema.

MODULI

Cosa sono e cosa permettono di fare

I moduli sono delle parti di codice che viene caricato e scaricato al volo nel kernel su richiesta. I moduli possono estendere le funzionalita' del kernel senza bisogno di un reboot del sistema. Un tipo di modulo e' il device driver che mi permette di accedere a un dispositivo esterno.

Library Functions e System Calls

Library functions

- printf(...)
- fopen(...)
- fclose(...)
- memcpy(...)
- ...

Sys calls

- read(...)
- write(...)
- close(...)
- ...







I device file

COSA SONO E DOVE SI TROVANO

I device file o special file sono dei file che rappresentano dei dispositivi fisici o virtuali con cui interagire. Si trovano tutti generalmente nella directory `/dev/` e nelle sue sottodirectory.

Esempio:

```
brw-rw---- 1 root disk 3,8 2006-10-24 17:41 hda8
```

Sono contraddistinti da un major number che rappresenta il modulo che li gestisce, mentre il minor number rappresenta il dispositivo fisico a cui sono associati.

GLI STRUMENTI

- strumenti di compilazione
- un compilatore
- make
- gli header del kernel
- pacchetto `module-init-tools` (insmod - modprobe)
- pacchetto coreutils (mknod)







File e comandi

- Makefile
- mychardev.c
- mychardev.h
- comando "make -C /usr/src/linux M=‘pwd‘ modules"
- `/dev/mychardev`







Makefile

ESEMPIO DI MAKEFILE

```
obj-m += mychardev.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```








Le funzioni principali

- `int init_module(...)`
- `void cleanup_module(...)`
- `printk(...)`
- `write_module(...)`
- `read_module(...)`
- `struct file_operations`







Inizializzazione del modulo

```
int init_module()
{
     int ret_val;
     // Register the character device (atleast try)
     ret_val = register_chrdev(MAJOR_NUM, DEVICE_NAME, &Fops);
     if (ret_val < 0) {
     printk(KERN_ALERT "%s failed with %d\n",
            "Sorry, registering the character device ", ret_val);
     return ret_val;
     }
     printk(KERN_INFO "%s The major device number is %d.\n",
                        "Registeration is a success", MAJOR_NUM);
     printk(KERN_INFO "If you want to talk to the device driver,\n");
     printk(KERN_INFO "you'll have to create a device file. \n");
     printk(KERN_INFO "We suggest you use:\n");
     printk(KERN_INFO "mknod %s c %d 0\n", DEVICE_FILE_NAME, MAJOR_NUM);
     printk(KERN_INFO "The device file name is important\n");
     return 0;
}
```








Scaricare il modulo

```
void cleanup_module()
{
        int ret;
        /*
         * Unregister the device
         */
        ret = unregister_chrdev(MAJOR_NUM, DEVICE_NAME);
        /*
         * If there's an error, report it
         */
        if (ret < 0)
                printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret);
}
```








File operations

```
struct file_operations Fops = {
        .read = device_read,
        .write = device_write,
        .ioctl = device_ioctl,
        .open = device_open,
        .release = device_release,
};
```








Gli include

```
#include <linux/kernel.h>       /* We're doing kernel work */
#include <linux/module.h>       /* Specifically, a module      */
#include <linux/fs.h>           /* For struct file_operations  */
#include <asm/uaccess.h>        /* For get_user and put_user   */
#include "mychardev.h" /* Personalization                 */

#include <linux/config.h>
#include <linux/parport.h>
#include <linux/parport_pc.h>/* For using parallel port
   function inb outb         */
```








Leggere dal device

```
static ssize_t device_read(struct file *file,
                           char __user * buffer,

                            size_t length,
                            loff_t * offset)
  {
 int bytes_read = 0;
 if (*Message_Ptr == 0)
                return 0;
 while (length && *Message_Ptr) {
   put_user(*(Message_Ptr++), buffer++);
           length--;
           bytes_read++;
        }
        return bytes_read;
  }
```








Scrivere sul device

```
static ssize_t
device_write(struct file *file,
        const char __user * buffer, size_t length, loff_t * offset){
       int number=0; //variabile che contiene il numero
                     //da scrivere sulla parallela
       int i;
       for (i = 0; i < length && i < BUF_LEN; i++){
              get_user(Message[i], buffer + i);
       }
       Message_Ptr = Message;
       /*  buffer e' l'array che si trova in user space
       Message si trova in kernel space                  */
       for (i = 0; i < length && i < BUF_LEN; i++){
       number=number*10;
       number= number + ((int)Message[i] - 48);
       outb((unsigned char)number, base);
       /*scrive sulla parallela il numero convertito*/
       /*ritorna il numero di caratteri in input */
       return i;
}
```








Risultato Finale

Posso scrivere sulla porta parallela come utente non privilegiato

Linguaggio C

```
int dato=255;
File* miofile =
fopen("/dev/mychardev",'a+');
fprintf(miofile,"%i",dato );
```

Shell script

```
echo 255 > /dev/mychardev
```








Riferimenti

WWW.TLDP.ORG
linux-kernel-2.6-module-programming.pdf
by Peter Jay Salzman, Michael Burian, Ori Pomerantz

WWW.DTI.SUPSI.CH
driver_linux_2.6.pdf
by Roberto Bucher
Scuola universitaria professionale della Svizzera Italiana

LaTeX