Linux Day 2006

GNU/Linux Kernel Module ...Programming

Fabio Rubino
frubino@cs.unibo.it

Moduli

- Cosa sono
- Perche' scriverli
- Quando è utile scriverli
- Come scriverli

Moduli – Cosa sono

Codice che può essere caricato e rimosso dinamicamente

Si possono pensare come librerie, plug-in per vari programmi, in questo caso il Kernel

Moduli – Peche' scriverli

- Supporto di nuove funzionalitàa a run-time
  - driver
  - file system
  - protocolli
  - ....

Moduli – Quando è utile scriverli

- Aggiunta di nuove funzionalità solo quando servono
- Risparmio di memoria
- Velocizzare la fase testing

Moduli – Come scriverli

E' bene tener presente alcune differenze tra la scrittura di un semplice programma e la scrittura di un modulo del kernel

- **Ciclo di vita**
- **Contesto di esecuzione**
- **Funzioni esterne**
- **Protezione memoria**
- **Memoria disponibile**
- **Concorrenza**
- **Floating point**

Ciclo di vita

Programmi

- Iniziano, fanno ciò che devono e terminano

Moduli

- Caricati, rimangono in attesa di richieste
- Rimossi, le funzionalità non sono più disponibili

Contesto di esecuzione

Programmi

- User space
  - non hanno accesso diretto al risorse

Moduli

- Kernel space
  - accesso a tutto le risorse

Funzioni esterne

Programmi

- Possibile utilizzo di funzioni di libreria
  - printf()
  - strcopy()
  - ...

Moduli

- Solo funzioni e macro esportate dal kernel
  - printk()
  - put_user()
  - ...

Protezione della memoria

Programmi

- Fornita dal S.O.
  - dereferenzianzione di un puntatore nullo invio del segnale SIGSEGV da parte del kernel e uccisione del processo

Moduli

- PANIC!

Memoria disponibile

Programmi

- Rappresentati in memoria virtuale
  - ampio stack

Moduli

- Stack molto piccolo 4/8KB
  - non passare grandi strutture
  - usare l'allocazione dinamica
- Rischio di sovrascrivere la memoria

Concorrenza

Programmi

- Generalmente single-threaded
  - sequenziale (no race-condition)

Moduli

- Multi-threaded
  - accesso a strutture dati condivise in maniera casuale a secondo della schedulzione possiblie perdita di consistenza (rece-condition)
- Proteggere le strutture dati
  - semafori, spinlock...

Floating point

Programmi

- Hanno a disposizioni i registri per il calcolo del FP

Moduli

- Non è ammesso l'uso del FP per evitare di dover salvare i registri FP ad ogni entrata e uscita dal kernel space

Iniziamo

Primo esempio il classico hello world che illustra lo scheletro che sta dietro ogni modulo

Il classico Hello world

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static *char name
module_param (name,  charp, S_IRUGO);

static int __init hello_init(void)
{
        printk(KERN_ALERT "Hello, %s\n", name);
        return 0;
}

static void __exit hello_exit(void)
{
        printk(KERN_ALERT "Goodbye, %s\n", name);
}

module_init(hello_init);
module_exit(hello_exit);
```

Analisi - Struttura

Solo due funzioni di base:

- Una funzione hello_init() eseguita al momento del caricamento
- Una funzione hello_exit() eseguita la momento della rimozione dal modulo

Analisi – hello_init()

```
static int __init hello_init(void)
{
    printk(KERN_ALERT "Hello world %s\n", name);
    return 0;
}
```

Invocata al momento del caricamento da module_init() stampa nel file di log “Hello, <name>”

- Inizializza strutture, device

Analisi – hello_exit()

```
static void __exit hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, %s\n", name);
}
```

Invocata al momento della rimozione da module_exit() stampa nel file di log “Goodbye, <name>”

- Dealloca tutto ciò che è stato inizializzato dalla init

Analisi – Stadi di log

- I possibili stadi di log

KERN_EMERG	0	sistema instabile
KERN_ALERT	1	stato di allerta
KERN_CRIT	2	stato critico
KERN_ERR	3	stato di errore
KERN_WARNING	4	stato in pericolo
KERN_NOTICE	5	
KERN_INFO	6	informazioni
KERN_DEBUG	7	messaggi di debug

Analisi – Passaggio dei parametri

- Tramite la macro module_param(name, type, perm)
  - name nome variabile
  - type tipo passato
  - perm permessi di accesso

Analisi - Headers

- **<linux/init.h>**
  macro (__init e __exit)
- **<linux/module.h>**
  strtture descrittive(nome, autore, licenza)
  funzioni (module_init/exit())
- **<linux/kernel.h>**
  prototipi di funzioni (printk())
  stadi di log (KERN_ALERT)

Compilazione

- Creare un Makefile contenente
  obj-y := hello.o (statica)
  obj-m := hello.o (come moulo)

- Compilare con il comando
  make -C <KernelDir> M=$(pwd) modules

  - KernelDir Directory del kernel
  - M=$(pwd) Directory contenente i moduli
  - modules Crea i moduli

Caricamento e rimozione

- Per caricare un modulo

  insmod <nome modulo>

  insmod hello.ko

- Per rimuoverlo

  rmmod <nome modulo>

  rmmod hello

Device driver

- Character device
- Block device
- Si trovano in /dev
  - /proc/devices tutti i device caricati
- Major number e minor number

Major e minor number

$ ls -l dev
brw-rw---- 1 root disk 3, 0 2006-10-26 14:03 /dev/hda

- Major number
  numero del device drivere utilizzato

- Minor number
  numero distintivo per ogni componente hardware che usa il device driver

registrare un character device

- Reggistriamo il nostro device

  register_chrdev(0, "cdev", &fops);

  - primo parametro il major number
  - secondo parametro il nome del device
  - terzo parametro struttura file_operations
  - restituisce il major number assegnato

- Per rilasciare un dispositivo

  unregister_chrdev("cdev");

file_operation struct

- Definire le operazioni open, read, write...

Struttura file operations di cdev

```
static struct file_operations fops = {
        .read = cdev_read,
        .write = cdev_write,
        .open = cdev_open,
        .release = cdev_release
};
```

cedv_open()

```
static int cdev_open(struct inode *inode, struct file *file)
{
    if (Busy)
        return -EBUSY;
    Busy++;
    string_p = string;
    try_module_get(THIS_MODULE);
    return 0;
}
```

- Controlla possibili errori del device
- Inizializa il device

cdev_read()

```
static ssize_t cdev_read(struct file filp,  /* file pointer */
                         char __user *buffer, /* user-data pointer */
                         size_t l_data,  /* length data read */
                         loff_t * offs) /* offset posizione corrente */
{
    if (*string_p == 0)
        return 0;

    while (length && *string_p) {
            put_user(*(string_p++), buffer++);
        l_data--;
    }
    return 0;
}
```

- Non si puo dereferenziare buffer
  - si usa put_user()

cdev_write()

```
static ssize_t cdev_write(struct file filp,  /* file pointer */
                          char __user *buffer, /* user-data pointer */
                          size_t l_data,  /* length data write */
                          loff_t * offs) /* offset posizione corrente */
{
    int i;
    for (i = 0; i < l_data && i < MAXLENGTH; i++)
        get_user(string[i], buffer + i);
    return i;
}
```

- Non si puo dereferenziare buffer
  - si usa get_user()

cdev_release()

```
static int cdev_release(struct inode *inode, struct file *file)
{
      Busy--;
      module_put(THIS_MODULE);
      return 0;
}
```

- Chiude il device

Dopo tanto....

- Compiliamo
- Carichiamo il modulo
- Creiamo il dispositivo

  mknod /dev/<nome device> c <Maj> <min>

- Per scopire il Major number assegnato

  cat /dev/devices

- Proviamolo

  echo “Linux Day 2006” > /dev/cdev0

  cat < /dev/cdev0

Per finire...

NON AVVELENATE IL KERNEL!!!

- **Rilasciate il vostri moduli sotto copyleft GPL**
- Basta aggiungere
  MODULE_LICENSE(“GPL”);

Grazie..

Happy Hacking!