I Moduli in Linux

Struttura dei *Loadable Kernel Modules* e loro utilizzo.

Luigi Mormile Antonio Ospite

Università degli Studi di Napoli
Federico II

Corso di Sistemi Operativi II
prof. Maurizio Giordano
Gennaio 2005

Sommario Parte I

Sommario Parte II

Parte I

LKMs

Cosa sono i moduli Linux?

Un modulo Linux è:

- Una porzione di software che può essere aggiunta al Kernel quando questo è già in esercizio
- Una parte del kernel che può essere caricata in memoria quando serve e scaricata se è opportuno

Vantaggio principale

Non serve ricompilare il Kernel per includere un modulo

Struttura di base di un kernel

Due grandi famiglie di kernel:

- Kernel monolitici
- Microkernel

In sintesi:

- La differenza tra Kernel monolitico e microkernel risiede nella comunicazione tra i sottosistemi
- La struttura di base di Linux è monolitica

Kernel Monolitico

Un kernel monolitico:

- È strutturato in un unico grande programma
- È potenzialmente veloce
- Rende tutte le parti del kernel visibili fra loro, a meno di scelte precise
- Può comunque seguire un approccio modulare

Linux:

Anche se il Kernel fa un massiccio uso dei Moduli, questi non sono entità autonome: il kernel rimane monolitico.

Microkernel

Un microkernel:

- Utilizzo di server diversi per compiti diversi
- I server vengono usati per gestire compiti del kernel

Impostazione filosofica:

Un OS basato su Microkernel utilizza processi separati
non solo per i programmi in User Mode,
ma anche per gestire i vari sottosistemi del Kernel.

Pro e Contro dei microkernel

Pro:

Lo sviluppo del kernel è più semplice da gestire perché i vari sottosistemi hanno un elevatissimo grado di indipendenza fra loro.

Contro:

Le prestazioni sono peggiori rispetto ai kernel Monolitici: ovunque vi sia comunicazione per scambio di messaggi c'è sempre un overhead tutt'altro che trascurabile.

Considerazioni

Alcune considerazioni.

- I Microkernel sono ritenuti spesso un buon esempio didattico (come Minix)
- Non tutti credono che l'idea del microkernel sia valida per sistemi operativi Genereal Purpose
- Si possono utilizzare alcune idee dell'approccio a microkernel nel progetto di un kernel monolitico

Cosa può fare un LKM?

LKM: Loadable Kernel Module

Servizi implementabili con un LKM:

Driver dei dispositivi:
Per comunicare con i dispositivi hardware senza mostrare i dettagli sul loro funzionamento.

Driver per i filesystem:
Per implementare l'accesso ad uno specifico filesystem implementando le procedure specifiche alla memorizzazione dei dati.

Driver di rete:
Per l'implementazione di protocolli per le reti a diversi livelli.

Cosa può fare un LKM? 2

Servizi implementabili con un LKM:

Chiamate a sistema:

Per specificare nuove *system call* o per ridefinire l'implementazione di una *syscall* esistente nel kernel.

Interpreti per eseguibili:

Per caricare e lanciare eseguibili in diversi formati. Il kernel Linux può eseguire programmi con formati diversi, purché vi siano moduli che ne conoscano la struttura.

Cosa NON può fare un LKM?

I moduli in linux condividono molte proprietà con i programmi utente ma NON devono essere confusi con essi.

I moduli in Linux NON possono:

- usare funzioni della libreria standard C (per esempio quelle definite in math.h)
- usare l'aritmetica floating point
- avere un utente proprietario

Come sono fatti i moduli

Per ogni modulo il Kernel alloca un'area di memoria contenente:

Dati di un modulo:

- Un oggetto module
- Una stringa che determina il nome del modulo
- Il codice oggetto del modulo

Nota:

Il kernel vede anche se stesso come un modulo!!
(mette se stesso nella lista dei moduli del sistema)

Struttura struct module

struct module ../include/linux/module.h

```
struct module
{
  unsigned long size_of_struct; /* == sizeof(module) */
  struct module *next;          /*+ Elemento successivo nella lista +*/
  const char *name;             /*+ Puntatore al nome del modulo +*/
  unsigned long size;           /*+ Dimensione modulo +*/

  union
  {
    atomic_t usecount;          /*+ Usage Counter +*/
    long pad;
  } uc;        /* Needs to keep its size - so says rth */
```

Struttura struct module 2

struct module ../include/linux/module.h

```
14  unsigned long flags;   /* AUTOCLEAN et al */      /*+ Indicatori di stato +*/

16  unsigned nsyms;                                   /*+ Numero simboli esportati +*/
    unsigned ndeps;                                   /*+ Numero moduli dipendenti +*/
18
    struct module_symbol *syms;                       /*+ Tabella simboli  esportati  +*/
20  struct module_ref *deps;                          /*+ Lista  dei  moduli dipendenti +*/
    struct module_ref *refs;                          /*+ Moduli da cui si dipende +*/
22  int  (* init )(void);                             /*+ Routine di  inizializzazione  +*/
    void (*cleanup)(void);                            /*+ Routine di cancellazione +*/
24  const struct exception_table_entry *ex_table_start; /*+ Inizio  tabella  eccezioni +*/
    const struct exception_table_entry *ex_table_end;  /*+ Fine tabella  eccezioni +*/
26 #ifdef __alpha__
    unsigned long gp;
28 #endif
```

Struttura struct module 3

struct module ../include/linux/module.h

```
    /* Members past this point are extensions to the basic
30     module support and are optional. Use mod_member_present()
       to examine them. */

/*+ ... campi opzionali +*/
41  };
```

La tabella delle eccezioni

Tabella delle eccezioni.

- I simboli di inizio e temine sono identificati con `__start___ex_table` e `__stop___ex_table`
- Linux definisce diverse *exception tables*
- Ogni modulo del kernel caricato dinamicamente possiede la propria exception table.
- Ogni entry di una exception table è di tipo `exception_table_entry`

Esportazione dei simboli

Esportazione dei simboli

- I simboli di un modulo possono essere esportati ed usati da altri moduli
- Usando la macro EXPORT_NO_SYMBOLS i simboli del modulo non sono aggiunti alla tabella globale dei simboli
- Per esportare dei simboli si deve definire la macro EXPORT_SYMTAB prima di includere linux/module.h .
  A questo punto si può usare la macro EXPORT_SYMBOL per esportare un simbolo specifico.
- Se né EXPORT_NO_SYMBOLS né EXPORT_SYMTAB sono definite nel sorgente del modulo, allora tutti i simboli globali non-statici del modulo vengono esportati.

Processo di inizializzazione

Inizializzazione con il programma insmod.

Fasi principali:

1. Calcolo dello spazio richiesto dal modulo
2. Chiamata a create_module()
   - Chiamata a sys_create_module()
3. Copia del modulo nello spazio User Mode
4. Chiamata ad init_module ()
   - Chiamata a sys_init_module ()
5. Rilascio della memoria in User Mode

Creazione di un modulo

sys_create_module() ../kernel/module.c

```
/*
 * Allocate space for a module.
 */

asmlinkage unsigned long
sys_create_module(const char *name_user, size_t size)
{
    char *name;
    long namelen, error;
    struct module *mod;
    unsigned long flags;
```

Descrizione

- Alcune variabili locali

Creazione di un modulo 2

sys_create_module() ../kernel/module.c

```
    if (!capable(CAP_SYS_MODULE))    /*+ Verifica se il kernel gestisce i moduli +*/
14      return -EPERM;
    lock_kernel ();
16  if ((namelen = get_mod_name(name_user, &name)) < 0) {
        error = namelen;
18      goto err0;
    }
```

Descrizione

- Controllo sul supporto ai moduli

Creazione di un modulo 3

sys_create_module() ../kernel/module.c

```
20      if (size < sizeof(struct module)+namelen+1) {
            error = -EINVAL;
22          goto err1;
        }
24      if (find_module(name) != NULL) { /*+ Cerca il modulo nella lista dei moduli +*/
            error = -EEXIST;
26          goto err1;
        }
```

Descrizione

- Controllo sull'esistenza del modulo

Creazione di un modulo 4

sys_create_module() ../kernel/module.c

```
28      if ((mod = (struct module *)module_map(size)) == NULL) {
            error = -ENOMEM;
30          goto err1;
        }
32
        memset(mod, 0, sizeof(*mod));     /*+ Una prima inizializzazione +*/
34      mod->size_of_struct = sizeof(*mod);
        mod->name = (char *)(mod + 1);
36      mod->size = size;
        memcpy((char*)(mod+1), name, namelen+1);
```

Descrizione

- Allocazione spazio
- Prima inizializzazione della struttura

Creazione di un modulo 5

sys_create_module() ../kernel/module.c

```
put_mod_name(name);

spin_lock_irqsave(&modlist_lock, flags);
mod->next = module_list;          /*+ Aggiunta alla lista dei moduli +*/
module_list = mod; /* link it in */
spin_unlock_irqrestore(&modlist_lock, flags);
```

Descrizione

- Salvataggio irq
- Inserimento nella lista dei moduli
- Ripristino irq

Creazione di un modulo 6

sys_create_module() ../kernel/module.c

```
46      error = (long) mod;
        goto err0;
48  err1:
        put_mod_name(name);
50  err0:
        unlock_kernel ();
52      return error;
    }
```

Descrizione

- Gestione errori e ritorno

Inizializzazione di un modulo

sys_init_module() ../kernel/module.c

```
/*
 * Initialize a module.
 */

asmlinkage long
sys_init_module(const char *name_user, struct module *mod_user)
{
    struct module mod_tmp, *mod, *mod2 = NULL;
    char *name, *n_name, *name_tmp = NULL;
    long namelen, n_namelen, i, error;
    unsigned long mod_user_size, flags;
    struct module_ref *dep;
```

Descrizione

- Variabili locali

Inizializzazione di un modulo 2

sys_init_module() ../kernel/module.c

```
14      if (!capable(CAP_SYS_MODULE))    /*+ Verifica se il kernel gestisce i moduli +*/
            return -EPERM;
16      lock_kernel ();
        if ((namelen = get_mod_name(name_user, &name)) < 0) {
18          error = namelen;
            goto err0;
20      }
        if ((mod = find_module(name)) == NULL) {
22          error = -ENOENT;
            goto err1;
24      }
```

Descrizione

- Controllo gestione moduli
- Controllo esistenza del modulo

Inizializzazione di un modulo 3

sys_init_module() ../kernel/module.c

```
/* ... */
    /* Hold the current contents while we play with the user's idea
       of righteousness. */
    mod_tmp = *mod;
    name_tmp = kmalloc(strlen(mod->name) + 1, GFP_KERNEL); /* Where's kstrdup()? */
    if (name_tmp == NULL) {
        error = -ENOMEM;
        goto err1;
    }
    strcpy(name_tmp, mod->name);     /*+ Copia del nome del modulo +*/
```

Descrizione

- Copia del modulo e del suo nome

Inizializzazione di un modulo 4

sys_init_module() ../kernel/module.c

```
50      /* Copying mod_user directly over mod breaks the module_list chain and
         * races against search_exception_table.  copy_from_user may sleep so it
52       * cannot be under modlist_lock, do the copy in two stages.
         */
54      if (!(mod2 = vmalloc(mod_user_size))) {
            error = -ENOMEM;
56          goto err2;
        }
58    /*+ Copia dallo spazio utente +*/
        error = copy_from_user(mod2, mod_user, mod_user_size);
60      if (error) {
            error = -EFAULT;
62          goto err2;
        }
```

Descrizione

- Inserimento nella lista dei moduli: Primo Passo

Inizializzazione di un modulo 5

sys_init_module() ../kernel/module.c

```
64      spin_lock_irqsave(&modlist_lock, flags);
        memcpy(mod, mod2, mod_user_size); /*+ Copia effettiva del modulo +*/
66      mod->next = mod_tmp.next;
        spin_unlock_irqrestore(&modlist_lock, flags);
```

Descrizione

- Inserimento nella lista dei moduli: Secondo Passo

Inizializzazione di un modulo 6

sys_init_module() ../kernel/module.c

```
69      /* Sanity check the size of the module. */

...     /*+ controlli vari sui campi della struct module +*/
164
        /* Ok, that's about all the sanity we can stomach; copy the rest. */
```

Descrizione

- Controlli sui tutti i puntatori della **struct** module

Inizializzazione di un modulo 7

sys_init_module() ../kernel/module.c

```
/* Ok, that's about all the sanity we can stomach; copy the rest. */

if (copy_from_user((char *)mod+mod_user_size,
                   (char *)mod_user+mod_user_size,
                   mod->size-mod_user_size)) {
    error = -EFAULT;
    goto err3;
}

if (module_arch_init(mod))
    goto err3;
```

Descrizione

- Copia della restante parte del modulo

Inizializzazione di un modulo 8

sys_init_module() ../kernel/module.c

```
    /* On some machines it is necessary to do something here
178     to make the I and D caches consistent.  */
    flush_icache_range((unsigned long)mod, (unsigned long)mod + mod->size);
180
    mod->refs = NULL;
182
    /* Sanity check the module's dependents */
184 for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {

... /*+ Controllo sui moduli dipendenti +*/
203 }
```

Descrizione

- Controllo sui moduli che dipendono dal modulo che si sta inizializzando

Inizializzazione di un modulo 9

sys_init_module() ../kernel/module.c

```
    /* Update module references. */
206 for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {

... /*+ Controllo sui riferimenti al modulo +*/
    }
216
    /* Free our temporary memory. */
218 put_mod_name(n_name);
    put_mod_name(name);
```

Descrizione

- Aggiornamento ai riferimenti al modulo corrente

Inizializzazione di un modulo 10

sys_init_module() ../kernel/module.c

```
    /* Initialize the module. */
222 /*+ Settaggio di stato e lancio routine inizializzazione +*/
    atomic_set(&mod->uc.usecount,1);
224 mod->flags |= MOD_INITIALIZING;
    if (mod->init && (error = mod->init()) != 0) {
226     atomic_set(&mod->uc.usecount,0);
        mod->flags &= ~MOD_INITIALIZING;
228     if (error > 0)  /* Buggy module */
            error = -EBUSY;
230     goto err0;
    }
232 atomic_dec(&mod->uc.usecount);
```

Descrizione

- Chiamata alla routine di inizializzazione di quel modulo

Inizializzazione di un modulo 11

sys_init_module() ../kernel/module.c

```
234     /* And set it running. */
        mod->flags = (mod->flags | MOD_RUNNING) & ~MOD_INITIALIZING;
236     error = 0;
        goto err0;
```

Descrizione

- Aggiornamento stato del modulo

Inizializzazione di un modulo 12

sys_init_module() ../kernel/module.c

```
err3:
240     put_mod_name(n_name);
err2:
242     *mod = mod_tmp;
        strcpy ((char *)mod->name, name_tmp);    /* We know there is room for this */
244 err1:
        put_mod_name(name);
246 err0:
        if (mod2)
248         vfree(mod2);
        unlock_kernel();
250     kfree(name_tmp);
        return error;
252 }
```

Descrizione

- Fasi di uscita

Processo di scaricamento.

Rimozione con il programma `rmmod`.

Fasi principali:

1. Ricerca dei moduli collegati a quello da rimuovere
2. Ricerca delle dipendenze dei moduli collegati (se usato con -r)
3. Costruzione di una lista dei moduli da rimuovere (se usato con -r)
4. Chiamata a delete_module()
   - Chiamata a sys_delete_module()
5. Rimozione eventuale dei moduli collegati

Scaricamento di un modulo

sys_delete_module() ../kernel/module.c

```
asmlinkage long
sys_delete_module(const char *name_user)
{
  struct module *mod, *next;
  char *name;
  long error;
  int something_changed;

  if (!capable(CAP_SYS_MODULE))   /*+ Verifica se il kernel gestisce i moduli +*/
    return -EPERM;
```

Descrizione

- Controllo sulla gestione dei moduli

Scaricamento di un modulo 2

sys_delete_module() ../kernel/module.c

```
12    lock_kernel ();
      if (name_user) {
14      /*+ Copia il nome del modulo dallo spazio utente +*/
        if ((error = get_mod_name(name_user, &name)) < 0)
16        goto out;
        error = -ENOENT;
18      /*+ Cerca l'oggetto module nella lista "module_list" +*/
        if ((mod = find_module(name)) == NULL) {
20        put_mod_name(name);
          goto out;
22      }
        put_mod_name(name);
24      error = -EBUSY;

/* ... */
```

Descrizione

- Controllo sull'esistenza del modulo da rimuovere

Scaricamento di un modulo 3

sys_delete_module() ../kernel/module.c

```
28
    spin_lock(&unload_lock);
30  if (!__MOD_IN_USE(mod)) {
      mod->flags |= MOD_DELETED;
32    spin_unlock(&unload_lock);
      /*+ Chiama la cleanup,
34        rimuove il modulo dalla lista delle dipendenze e dalla lista dei moduli
          e libera la memoria +*/
36    free_module(mod, 0);
      error = 0;
```

Descrizione

- Rimozione del modulo

Scaricamento di un modulo 4

sys_delete_module() ../kernel/module.c

```
44    /* Do automatic reaping */
      restart :
46    something_changed = 0;

48    /*+ Scandisce la lista dei moduli
          e rimuove il modulo +*/
50    for (mod = module_list; mod != &kernel_module; mod = next) {

      /*+ ... eventualmente pone something_changed ad 1 +*/
71        }
```

Descrizione

- Rimozione moduli dipendenti
- Rimozione moduli con AUTOCLEAN

Scaricamento di un modulo 5

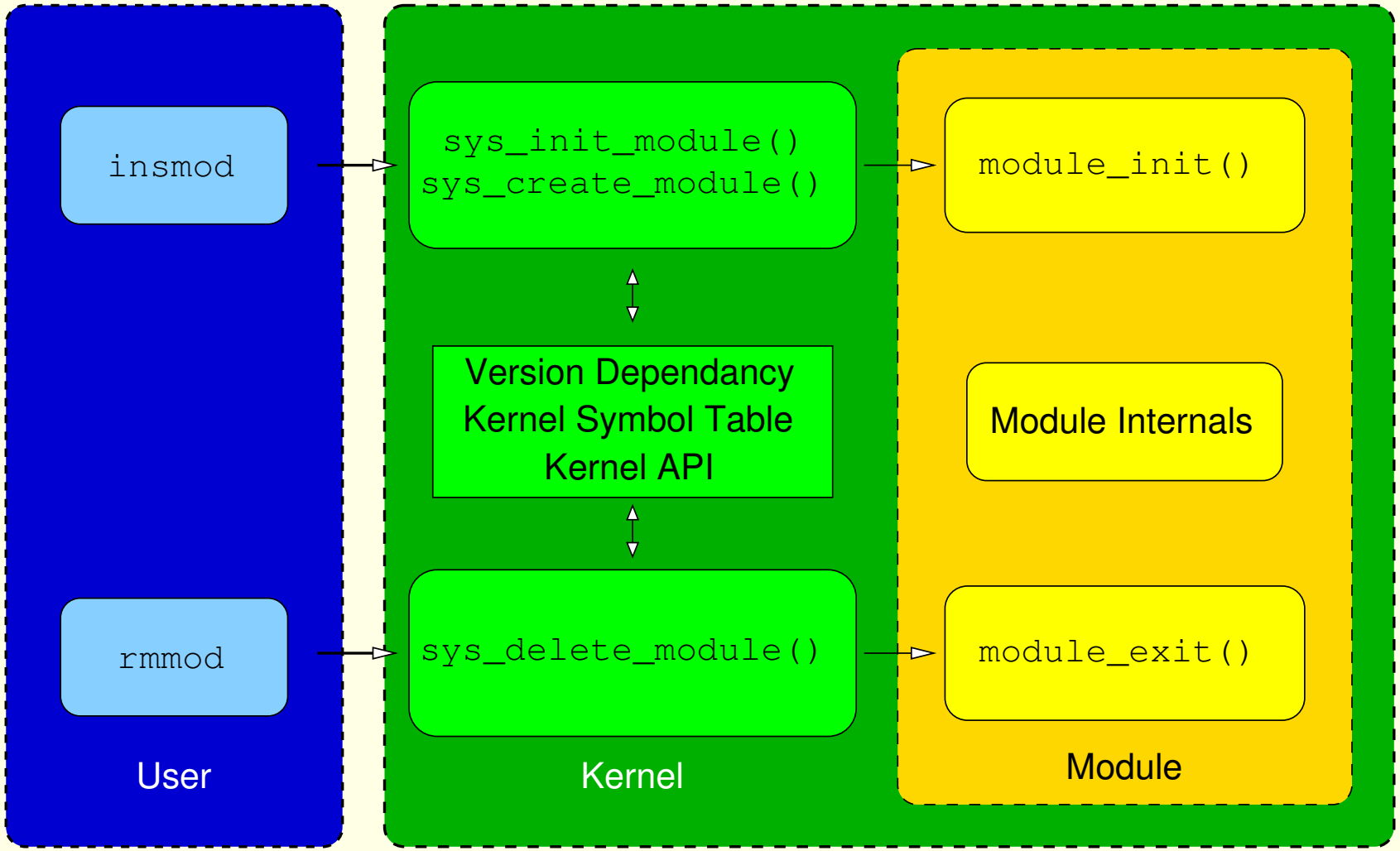
sys_delete_module() ../kernel/module.c

```
  }

  if (something_changed)
    goto restart;

  for (mod = module_list; mod != &kernel_module; mod = mod->next)
    mod->flags &= ~MOD_JUST_FREED;

  error = 0;
out:
  unlock_kernel();
  return error;
}
```

Descrizione

- Aggiornamento variabili di stato e gestione errori

Schema riassuntivo



Parte II

Un esempio

Problema

Quale è il problema:

Monitorare l'invocazione di una system call per un utente

Perche?

Per esempio per controllare processi sospetti

Soluzione

Mediante il descrittore di processo puntato dalla macro `current` possiamo conoscere l'uid del processo che ha richiesto la system call.

```
current->uid
```

Possiamo intercettare la *system call* grazie alla tabella:

```
extern void *sys_call_table[];
```

Kernel recenti

Sui kernel 2.6 il simbolo sys_call_table non è esportato.

Come fare?

Usiamo l'indirizzo direttamente!

```
$ cat /boot/System.map-`uname -r` | grep -C 2 sys_call_table
...
c03fc500 D interrupt
c03fc540 D sys_call_table
c03fc9c4 d kstack_depth_to_print
...
$ _
```

Makefile

Makefile

```
# A makefile for linux 2.6 AND 2.4

MODULENAME=uid_mon

BUILDVER=$(shell if uname -r | grep -q ^2.6; then echo "linux26"; \
                                 else echo "linux24"; fi)

ifeq (${BUILDVER},linux24)
TARGET  := $(MODULENAME)
INCLUDE := -I/lib/modules/$(shell uname -r)/build/include
CFLAGS  := -O2 -Wall
DEFINES := -D_LINUX24xx_ -DMODULE -D__KERNEL__ -DLINUX
endif

obj-m   := $(MODULENAME).o

KDIR    := /lib/modules/$(shell uname -r)/build
PWD     := $(shell pwd)
```

Makefile 2

Makefile

```
20 default: $(BUILDVER)

22 linux26:
       $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
24
   linux24:
26     $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) $(TARGET).o

28 clean:
       rm -f *.o *.ko *.mod.c .*.cmd
30     rm -r .tmp_versions
```

Codice del modulo

uid_mon.c

```
...
14
/*+ THIS_MODULE si riferira' all'oggetto module corrente +*/
16 #define MODULE_NAME THIS_MODULE->name

18 /*+ kernel headers +*/
#include <linux/module.h>
20 #include <linux/moduleparam.h>
#include <linux/init.h>
22
#include <linux/unistd.h>
24
/*+ Per la macro 'current' +*/
26 #include <linux/sched.h>
```

Codice del modulo 2

uid_mon.c

```
28  /*+ Per la tabella delle sysem call +*/
    #ifdef _LINUX24xx_
30  extern void *sys_call_table[];
    #else
32  /*+ Il simbolo 'sys_call_table' non e' esportato nei kernel 2.6 +*/
    #define SYS_CALL_TABLE_ADDRESS 0xc03eb540
34  void **sys_call_table=(void **) SYS_CALL_TABLE_ADDRESS;
    #endif
36
    /*+ L'uid che si desidera controllare (passato sulla linea di comando) +*/
38  static int uid=-1;
    module_param(uid, int, 0);
40  MODULE_PARM_DESC(uid, "The UID we want to spy on.");
```

Codice del modulo 3

uid_mon.c

```
42  /*+ Un puntatore alla vecchia syscall +*/
    int (*old_mkdir)(char *path) = NULL;
44
    /*+ La nostra nuova syscall +*/
46  int new_mkdir(char *path)
    {
48    if (current->uid==uid)
        printk (KERN_ALERT
50          "%s: ---> Called mkdir by uid=%d.\n", MODULE_NAME, uid);
      return old_mkdir(path);
52  }
```

Codice del modulo 4

uid_mon.c

```
54  /*+ Initializzazione LKM +*/
    int __init uid_mon_init()
56  {
      printk (KERN_ALERT "%s: Loading Module...\n", MODULE_NAME);
58    if (uid==-1)
        return -1;
60
      old_mkdir = sys_call_table [__NR_mkdir];
62    sys_call_table [__NR_mkdir] = new_mkdir;

64    printk (KERN_ALERT "%s: Module initialized.\n", MODULE_NAME);
      printk (KERN_ALERT "%s: Spying on uid: %d.\n", MODULE_NAME, uid);
66
      /*+ ritorniamo 0 in caso di successo, un altro valore altrimenti +*/
68    return 0;
    }
```

Codice del modulo 5

uid_mon.c

```
/*+ Metodo per il Cleanup +*/
void __exit uid_mon_cleanup()
{
  printk("%s: Cleaning up Module.\n", MODULE_NAME);
  sys_call_table[__NR_mkdir] = old_mkdir;
  printk("%s: Module unloaded.\n", MODULE_NAME);
}

/*+ Impostazione metodi di 'init' e 'cleanup' +*/
module_init(uid_mon_init);
module_exit(uid_mon_cleanup);
```

Codice del modulo 6

uid_mon.c

```
/*+ Alcune informazioni +*/
84 MODULE_LICENSE("GPL");
MODULE_AUTHOR("Mormile e Ospite");
86 MODULE_DESCRIPTION("Esempio di LKM per il corso di SOII");
```

Riferimenti bibliografici

linuxdevices.com.
Howto compile kernel modules for the kernel 2.6.
http://linuxdevices.com/articles/AT4389927951.html.

J. Newbigin.
Patching a running linux kernel.
http://uranus.it.swin.edu.au/~jn/linux/kernel.htm.

W. R. Stevens.
Advanced Programming in the UNIX Environment.
Addison Wesley, 1993.
URL http://www.kohala.com/start/apue.html

Riferimenti bibliografici 2

D. Bovet, M. Cesati et A. Oram.
Understanding the Linux Kernel, Second Edition.
O'Reilly & Associates, Inc., 2002.
URL
http://www.oreilly.com/catalog/linuxkernel2

L. Colombo.
Writing a malicious Linux Kernel Module.
http://openit.disco.unimib.it/docs/colombo.pdf, 2003.

B. Henderson.
Linux loadable kernel module howto.
http://www.tldp.org/HOWTO/Module-HOWTO/, 2005.