

Modulos

AISO - Introducción



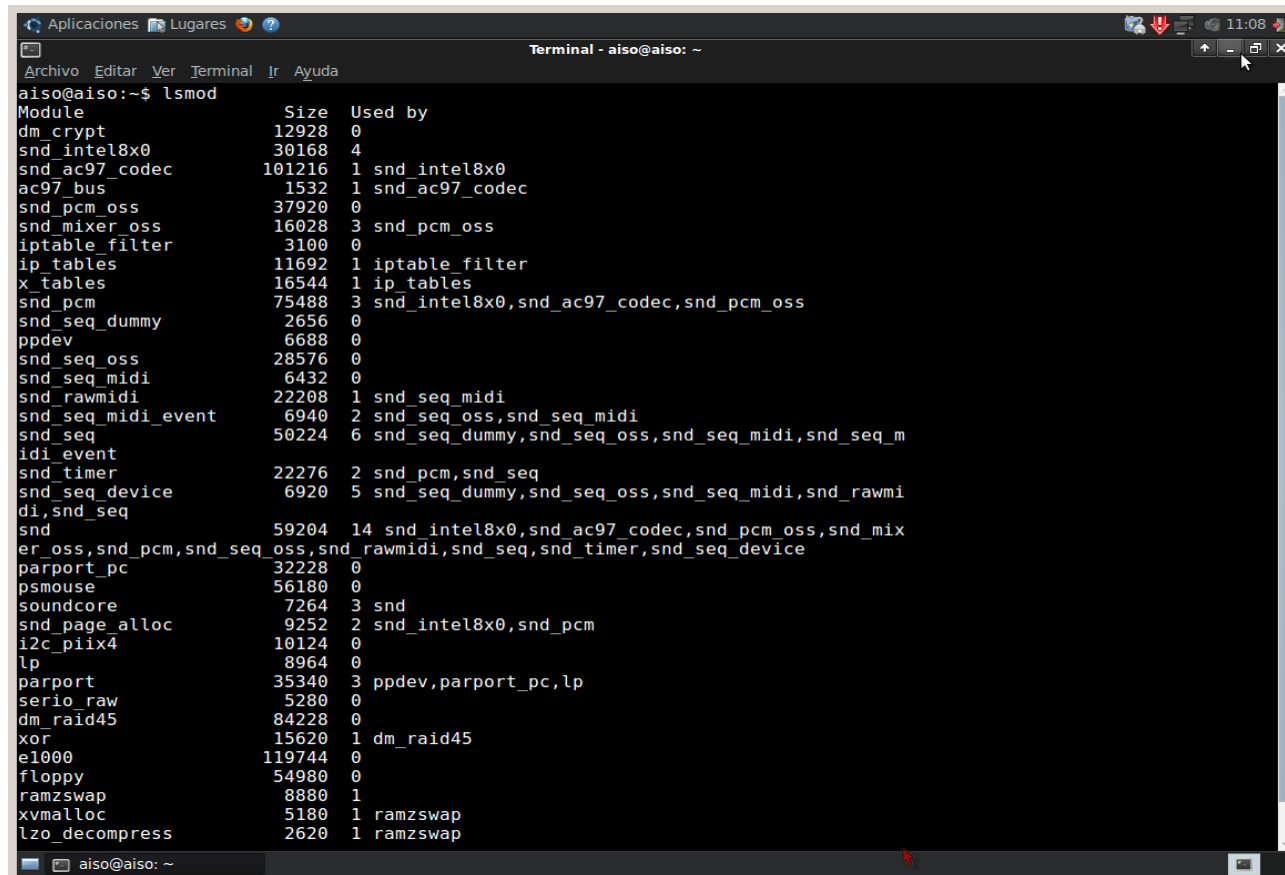
Modulos Cargables(I)

- Qué es un módulo del kernel Linux?
 - Un “fragmento de código” que puede cargarse/descargarse en el kernel bajo demanda
- Objetivos:
 - Reducir el *footprint* del kernel
 - Permitir extender la funcionalidad del kernel en caliente – sin necesidad de hacer un “reboot” del sistema –
- Disponibles desde la versión Linux 1.2
 - También existe soporte para módulos cargables en las variantes BSD, Sun Solaris, MS Windows...



Modulos Cargables (II)

- Los módulos disponibles para nuestro kernel se encuentran en `/lib/modules/version`
- Podemos conocer los modulos que estan cargados con `lsmod` (`/proc/modules`)



```
Terminal - aiso@aiso: ~
Archivo Editar Ver Terminal Ir Ayuda
aiso@aiso:~$ lsmod
Module              Size  Used by
dm_crypt            12928  0
snd_intel8x0        30168  4
snd_ac97_codec     101216  1 snd_intel8x0
ac97_bus            1532   1 snd_ac97_codec
snd_pcm_oss         37920  0
snd_mixer_oss       16028  3 snd_pcm_oss
iptables_filter     3100   0
ip_tables           11692  1 iptables_filter
x_tables            16544  1 ip_tables
snd_pcm             75488  3 snd_intel8x0,snd_ac97_codec,snd_pcm_oss
snd_seq_dummy       2656   0
ppdev              6688   0
snd_seq_oss         28576  0
snd_seq_midi        6432   0
snd_rawmidi         22208  1 snd_seq_midi
snd_seq_midi_event   6940   2 snd_seq_oss,snd_seq_midi
snd_seq             50224  6 snd_seq_dummy,snd_seq_oss,snd_seq_midi,snd_seq_m
idi_event
snd_timer           22276  2 snd_pcm,snd_seq
snd_seq_device       6920   5 snd_seq_dummy,snd_seq_oss,snd_seq_midi,snd_rawmi
di,snd_seq
snd                 59204  14 snd_intel8x0,snd_ac97_codec,snd_pcm_oss,snd_mix
er_oss,snd_pcm,snd_seq_oss,snd_rawmidi,snd_seq,snd_timer,snd_seq_device
parport_pc          32228  0
psmouse             56180  0
soundcore            7264   3 snd
snd_page_alloc      9252   2 snd_intel8x0,snd_pcm
i2c_piix4            10124  0
lp                  8964   0
parport             35340  3 ppdev,parport_pc,lp
serio_raw            5280   0
dm_raid45           84228  0
xor                 15620  1 dm_raid45
e1000               119744  0
floppy              54980  0
ramzswap             8880   1
xvmalloc            5180   1 ramzswap
lzo_decompress       2620   1 ramzswap
```



Anatomía de un modulo cargable (I)

- Fichero objeto ELF (Executable and Linkable Format) especial (**.ko**)
 - Por qué objeto?
 - Tipicamente los objetos se linkan (otros objetos, bibliotecas) para generar un ejecutable.
 - Un modulo no puede resolver todos sus símbolos hasta su carga
 - Se puede inspeccionar su contenido con herramientas habituales (**file**, **objdump**, ...)

```
Terminal - aiso@aiso: /lib/modules/2.6.31-14-generic-pae/kernel/sound
Archivo  Editar  Ver  Terminal  Ir  Ayuda

aiso@aiso:/lib/modules/2.6.31-14-generic-pae/kernel/sound$ ls
ac97_bus.ko  core  drivers  i2c  isa  oss  pci  pcmcia  soc  soundcore.ko  sound_firmware.ko  synth  usb
aiso@aiso:/lib/modules/2.6.31-14-generic-pae/kernel/sound$ file soundcore.ko
soundcore.ko: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
aiso@aiso:/lib/modules/2.6.31-14-generic-pae/kernel/sound$
```



Anatomia de un modulo cargable (II)

```
manuel@ubuntuserver: /lib/modules/2.6.31-19-generic-pae/kernel/sound
File Edit View Terminal Help
manuel@ubuntuserver:/lib/modules/2.6.31-19-generic-pae/kernel/sound$ objdump -h soundcore.ko

soundcore.ko:      file format elf32-i386

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .note.gnu.build-id 00000024 00000000 00000000 00000000 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .text           0000081c 00000000 00000000 00000060 2**4
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  2 .exit.text      0000001e 00000000 00000000 0000087c 2**0
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  3 .init.text      00000073 00000000 00000000 0000089a 2**0
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  4 .rodata         000000a8 00000000 00000000 00000920 2**5
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
  5 .rodata.str1.1  00000098 00000000 00000000 000009c8 2**0
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .parainstructions 00000028 00000000 00000000 00000a60 2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
  7 .rodata.str1.4  0000004e 00000000 00000000 00000a88 2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .modinfo        000000df 00000000 00000000 00000ae0 2**5
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 __ksymtab       00000050 00000000 00000000 00000bc0 2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
10 __kcrctab       00000028 00000000 00000000 00000c10 2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
11 __ksymtab_strings 000000d8 00000000 00000000 00000c38 2**0
    CONTENTS, ALLOC, LOAD, READONLY, DATA
12 __versions      00000600 00000000 00000000 00000d20 2**5
    CONTENTS, ALLOC, LOAD, READONLY, DATA
13 .data           00000000 00000000 00000000 00001320 2**2
```



Anatomía de un modulo cargable (III)

- En lugar de una función `main`, un modulo tiene funciones `entry` y `exit`
 - `entry`: cuando se carga el módulo en el kernel (macro `module_init`)
 - `exit`: Cuando se elimina el módulo del kernel (macro `module_exit`)



Gestión de Módulos (I)

- Insercción/enlace (Carga en el Kernel) y Eliminación (Descarga del Kernel) Básicos
 - **Insmmod / rmmmod**
 - Cómo es el enlace?
 - El kernel mantiene tablas de símbolos (variables, funciones)
 - Secciones `__ksymtab`, `__ksymtab_gpl` . . .
- Qué símbolos pueden utilizarse? Los símbolos del kernel tienen tres niveles de visibilidad
 - **static**: visibles solo dentro del mismo fichero fuente
 - **extern**: visibles desde cualquier fichero fuente utilizando en la construcción del kernel
 - **exported**: visible/disponible para cualquier modulo cargable – *exported kernel interface* / **API del kernel** – . Macros:
 - **EXPORT_SYMBOL** (cualquier módulo)
 - **EXPORT_SYMBOL_GPL** (solo con licencia compatible GPL)



Gestión de Módulos (II)

- Inventario de dependencias
 - Los módulos también pueden exportar símbolos (en sus secciones `__ksymtab ...`) para ser utilizados por otros.
 - Para gestionar dichos símbolos es conveniente generar un inventario de dependencias
 - **depmod** (`/lib/modules/version/modules.dep`)
- Instalación o Eliminación “inteligente”
 - **modprobe** (wrapper de `insmod` y `rmmod`)
 - Busca por defecto en `/lib/modules/version`
 - Tiene en cuenta dependencias
- Conocer información (macros)
 - **modinfo**



Ejemplo Simple

```
*simple-lkm.c
```

```
1  #include <linux/module.h>
2  /* Licencia del modulo */
3  MODULE_LICENSE("GPL");
4
5  /* Función que se invoca cuando se carga el modulo en el kernel */
6  int modulo_aiso_init( void )
7  {
8      printk(KERN_INFO "Modulo aiso cargado. Hola Kernel.\n");
9      return 0;
10 }
11
12 /* Función que se invoca cuando se descarga el modulo del kernel */
13 void modulo_aiso_clean(void)
14 {
15     printk(KERN_INFO "Modulo aiso descargado. Adios Kernel.\n");
16     return;
17 }
18
19 /* Declaración funciones init y exit */
20 module_init(modulo_aiso_init);
21 module_exit(modulo_aiso_clean);
```

- Macros
- Constructor del Módulo
- Destructor del Módulo
- Macros init y exit



Interfaz /proc (I)

- Manipulación entradas /proc <linux/proc_fs.h>

- `struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode, struct proc_dir_entry *parent);`
- `void remove_proc_entry(const char *name, struct proc_dir_entry *parent);`
- `struct proc_dir_entry{`
 - `unsigned int low_ino;`
 - `unsigned short namelen;`
 - `const char *name;`
 - `mode_t mode;`
 - `nlink_t nlink;`
 - `uid_t uid;`
 - `gid_t gid;`
 - `...`
 - `struct proc_dir_entry *next, *parent, *subdir;`
 - `void *data;`
 - `read_proc_t read_proc; /* Read Callback */`
 - `write_proc_t *write_proc; /* Write Callback */`
 - `...``};`



Interfaz /proc (II)

- Manipulación entradas /proc <linux/proc_fs.h>

- **Read Callback: Datos de salida – se lee la entrada (ej. cat)**

- `int read_proc_proto (char *buffer, char **buffer_location, off_t offset, int buffer_length, int *eof, void *data)`

- **buffer:** puntero a la cadena que devolvemos (en el espacio del kernel)

- **offset:** posición actual en el fichero

- **return int :**

- **0:** end of file (no tienes mas información que devolver)

- **Negativo:** error

- **Positivo:** se vuelve a llamar de nuevo a la función (advertencia!)



Interfaz /proc (III)

- Manipulación entradas /proc <linux/proc_fs.h>

- **Write Callback: Datos de entrada – se escribe la entrada (ej. echo)**

- `int write_proc_proto(struct file *filepointer, const char __user *buffer, unsigned long len, void *data)`

- `buffer`: puntero a la cadena que nos pasan (en espacio usuario: `copy_from_user`)

- `len`: longitud de la cadena

-



Otras funciones utiles

■ Gestion de memoria <linux/vmalloc.h>

■ Reservar memoria

```
■ void *vmalloc( unsigned long size );
```

■ Liberar memoria

```
■ void vfree( void *addr );
```

■ Copiar buffer entre espacio usuario ← → espacio kernel <asm-generic/uaccess.h>

```
■ unsigned long copy_to_user(void __user *to,const void  
*from,unsigned long n );
```

```
■ unsigned long copy_from_user(void *to,const void __user *from,  
unsigned long n );
```

■ Manejo de Cadenas

```
■ sprintf
```

```
■ memset
```



Referencias

- The Linux Kernel Programming Guide:
 - <http://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html>



Ejercicios (I)

■ Ejercicio 1

- `printk()` es un mecanismo de logging. 8 niveles de prioridad (`<linux/kernel.h>`)

- Que diferencia encuentras entre `KERN_INFO` y `KERN_ALERT`

■ Ejercicio 2

- La función de carga devuelve 0, que ocurre si devuelve un número negativo

■ Ejercicio 3

- Macros `__init` y `__exit` `<linux/init.h>`

- Que diferencias en el módulo generado?



Ejercicios (II)

■ Ejercicio 4

- En `<linux/module.h>` se describen los distintos tipos de licencias

- Que ocurre si no incluimos licencia en nuestro módulo?

■ Ejercicio 5

- Paso de parametros `<linux/moduleparam.h>`

- Ejemplo 2.7 de Linux Kernel Module Programming (modulo `hello-5`)



Ejercicios (III)

- Ejercicio 6
 - Exportar los parametros utilizados en `hello-5` y crear un modulo `dephello-5` que vuelque su contenido con `printk()`
 - Hay algun variación en el modulo entre exportar los parametros para todos los modulos o solo para los GPL?
 - Qué ocurre si intentamos descargar `hello-5` mientras `dephello-5` esta cargado?



Ejercicios (IV)

- Ejercicio 7
 - Crear un módulo que interaccione con `/proc`
 - Cuando el modulo se cargue/descargue se creará/eliminará una entrada **aiso** en el sistema de ficheros virtual `/proc`
 - Utilizar **aiso** como un clipboard del sistema
 - Consultar ejemplos 5.1 y 5.2 de Linux Kernel Module Programming – Advertencia!: necesario adaptarlos a la versión del kernel



AISO Introducción
Versión 0.2

© Manuel Prieto Matias

*This work is licensed under the Creative Commons **Attribution-Share Alike 3.0** Spain License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/es/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.*

*Esta obra está bajo una licencia **Reconocimiento-Compartir Bajo La Misma Licencia 3.0 España** de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.*

Este documento (o uno muy similar) esta disponible en <https://cv2.sim.ucm.es/moodle/course/view.php?id=3235>

