Módulos del Kernel

Roberto Muñoz Gómez
roberto@gul.es

Grupo de Usuarios de Linux
Universidad Carlos III de Madrid

2 de marzo de 2010

Contenido

Contenido

www.GUL.es

¿Qué es un módulo del Kernel?

- Trozo de código que se carga en el núcleo bajo demanda.
- Amplian sus capacidades sin necesidad de reiniciar.
- Un driver es un modulo del kernel, por ejemplo ETTY.

¿Por qué escribir módulos?

- Por diversión.
- Para ganar dinero
- Para aprender cómo funcionan los entresijos.
- Por que puedes.

¿Vale pero por qué de Linux?

- Porque es libre y el código fuente está disponible
- Por la inmensa comunidad que tiene alrededor
- ¿He dicho ya que es divertido?

¿Cómo se meten los módulos en el kernel?

- ¿Qué modulos estoy usando ahora mismo? `lsmod`
- ¿Cómo cargo uno? `modprobe`
- ¿Y para descargarlo? `rmmod`

Contenido

www.GUL.es

Hola Mundo

```
#include <linux/module.h>
#include <linux/kernel.h>
int init_module(void){
  printk(KERN_INFO 'Hola Mundo.\n');
  return 0;
}
void cleanup_module(void){
  printk(KERN_INFO 'Adios mundo cruel.\n');
}
```

Cómo se compila

```
obj-m += holamundo.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
```

Ver ejemplo 1

Cómo se le pasan argumentos

- `argc`/`argv` han pasado a mejor vida
- `module_param`(nombre,tipo,permisos)
  - Nombre de la variable
  - Tipo de variable

bool/invbool	un bit verdadero o falso.
charp	cadena
int	entero
long	entero largo
short	entero corto
uint...	sin signo

  - Permisos `sysfs` (ver `man 2 open`)

```
insmod mimodulo.ko miparametro=5
```

Ver ejemplo 2

Contenido

www.GUL.es

Cómo se ejecutan los módulos

- No hay `main()`
- No tienen un flujo determinado.

Funciones disponibles

- No hay `printf`, `malloc`, `strcmp`, `atoi`...
- No hay `libc`!!!!
- ¿Entonces me tengo que picar toooodo lo que haga?Sí.Bueno, casi todo.
- Se pueden usar todas las funciones que exporta el kernel.
- `/proc/kallsyms`

Memoria dinámica en el kernel

- `kmalloc`
  - `#include <linux/slab.h>`
  - `void *kmalloc(size_t size, int flags)`
  - flags: `GFP_KERNEL`
  - flags: `GFP_ATOMIC`
  - flags: ...
- `kfree(void *ptr)`

Espacio de Kernel vs Espacio de Usuario

- Objetivo del kernel:gestión de los recursos.
  - Tarjeta de video
  - Disco duro
  - Memoria
  - ...
- Competición por los recursos
- Modos de ejecución
  - Modo supervisor
  - Modo usuario

Espacio de nombres

- Variables locales
- Variables globales
  - Corrupción del espacio de nombres (*namespace pollution*)
- El kernel es un único espacio de nombres
- `/proc/kallsyms`

Espacio de código

- Gestión de memoria uffff
- Violación de Segmento
- Punteros = Direcciones de memoria físicas?NO
- Punteros = Direcciones de memoria lógicas.
- Espacio de código del kernel = módulos
- *Segfault* del módulo $\Rightarrow$ *Segfault* del kernel

Tienes un problema

Contenido

www.GUL.es

Nodos de dispositivo

- Qué son? Ficheros
- *major*
  - `/proc/devices`
- *minor*
- Tipos de dispositivos
  - Caracteres
  - Bloques
- Para crear uno
  - mknod <nombre> bc<b|c> *major minor*
- `/usr/src/linux/Documentation/devices.txt`

Estructura `file_operations`

```
struct file_operations {
 struct module *owner;
 loff_t (*llseek) (struct file *,loff_t, int);
 ssize_t (*read) (struct file *,char __user *,size_t,loff_t *);
 ssize_t (*write) (struct file *,const char __user *,size_t, loff_t *);
 int (*readdir) (struct file *,void *,filldir_t);
 unsigned int (*poll) (struct file *,struct poll_table_struct *);
 int (*ioctl) (struct inode *,struct file *,unsigned int,unsigned long);
 int (*open) (struct inode *,struct file *);
 int (*flush) (struct file *,fl_owner_t id);
 int (*fsync) (struct file *,struct dentry *,int datasync);
};
```

Uso de `file_operations`

```
struct file_operations fops = {
   .owner = THIS_MODULE,
   .read = midispositivo_read,
   .write = midispositivo_write,
   .open = midispositivo_open,
   .release = midispositivo_release
};
```

Otras estructuras importantes

- Estructura `file`
- Estructura `cdev`

Registrar un dispositivo

- cdev_alloc
- cdev_add
  - int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
- cdev_del

```
struct cdev *my_cdev = cdev_alloc( );
my_cdev->owner = THIS_MODULE;
my_cdev->ops = &my_fops;
ret=cdev_add(my_cdev, my_major, my_count);
...
cdev_del(my_cdev);
```

www.GUL.es

Ver ejemplo 3

Contenido

Pasar datos del usuario al kernel

- `ssize_t (*write) (struct file *,const char __user*,size_t, loff_t *);`
- `unsigned long copy_from_user(void *to,const void __user *from,unsigned long count);`

Pasar datos del kernel al usuario

- `ssize_t (*read) (struct file *,char __user*,size_t,loff_t *);`
- `unsigned long copy_to_user(void __user *to,const void *from,unsigned long count);`

www.GUL.es

Ver ejemplo 4

Contenido

www.GUL.es

Cómo crear un fichero en /proc

- <linux/proc_fs.h>
- int (*read_proc)(char *page, char **start, off_t offset, int count,int *eof, void *data);
- struct proc_dir_entry *create_proc_read_entry(const char *name, mode_t mode, struct proc_dir_entry *base, read_proc_t *read_proc, void *data);
- remove_proc_entry

Ver ejemplo 5

Contenido

www.GUL.es

`Read` y `Write`

- Dispositivos lógicos $\Leftrightarrow$ Dispositivos físicos
- Para leer: `read` ó `midispositivo_read`
- Para escribir: `write` ó `midispositivo_write`
- ¿Y si sólo quiero hablar con el dispositivo lógico?
  - Configurar baudios, paridad, bloquear una puerta, autodestrucción, ...

Ioctl

- IOCTL:Input Output ConTroL
- Usuario: `int ioctl(int fd, unsigned long cmd, ...????);`
- Cada `ioctl` es distinta, desestructurado, no documentado, no portable, ...
- En resumen: no usar
- Mejor otros métodos; sobre todo `sysfs`

Ver ejemplo 6

Contenido

www.GUL.es

¿Qué es una TTY?

- Originalmente
  - TTY: *Teletypewriter* teletipo
  - Conexión física o virtual
- Actualmente
  - Cualquier puerto serie
  - Terminales
  - USB-to-serial
  - Modems
  - Bluetooth

Cómo se hace un driver `tty`

- `<linux/tty_driver.h>`
- `struct tty_driver`
- `alloc_tty_driver(tty_minors)`
- `struct tty_operations`
- `tty_set_operations`
- `tty_registrer_driver`
- `tty_registrer_device`

Ejemplo de `tty_operations`

```
static struct tty_operations serial_ops = {
   .open = tiny_open,
   .close = tiny_close,
   .write = tiny_write,
   .write_room = tiny_write_room,
   .set_termios = tiny_set_termios,
};
```

Ejemplo de inicialización de driver

```
tiny_tty_driver->owner = THIS_MODULE;
tiny_tty_driver->driver_name = "tiny_tty";
tiny_tty_driver->name = "ttty";
tiny_tty_driver->major = TINY_TTY_MAJOR,
tiny_tty_driver->type = TTY_DRIVER_TYPE_SERIAL,
tiny_tty_driver->subtype = SERIAL_TYPE_NORMAL,
tiny_tty_driver->flags = TTY_DRIVER_REAL_RAW | TTY_DRIVER_NO_DEVFS,
tiny_tty_driver->init_termios = tty_std_termios;
tiny_tty_driver->init_termios.c_cflag = B9600 | CS8;
tty_set_operations(tiny_tty_driver, &serial_ops);
```

struct termios

```
struct termios tty_std_termios = {
  .c_iflag =ICRNL | IXON,
  .c_oflag =OPOST | ONLCR,
  .c_cflag =B38400 | CS8 | CREAD | HUPCL,
  .c_lflag =ISIG | ICANON | ECHO | ECHOE
        | ECHOK |ECHOCTL | ECHOKE | IEXTEN,
  .c_cc = INIT_C_CC
};
```

- Configuración inicial de la tty
- `man termios`

¿No hay función `read`?

- No es necesaria
- Drivers tty envían datos al hardware
- Datos recibidos $\rightsquigarrow$ `tty_flip_buffer`
- `tty_flip_buffer_push`

Ejemplo

```
for (i = 0; i < data_size; ++i) {
  if (tty->flip.count >= TTY_FLIPBUF_SIZE)
    tty_flip_buffer_push(tty);

  tty_insert_flip_char(tty, data[i], TTY_NORMAL);
}
tty_flip_buffer_push(tty);
```

Más sobre una `tty`

- Tan compleja como se desee
- Funciones `tiocmget` y `tiocmset`
- `ioctl`s propias de una `tty`.

Ver ejemplo etty

¿Alguna pregunta?

Bibliografía


[Peter Jay Salzman, 2005] Linux Kernel Module Programming Guide.
The Linux Documentation Project (www.tldp.org).


[O'Reilly, 2005] Linux Device Drivers Third Edition


[Imágenes]
Tux: tux.crystalxp.net
Logo: www.gul.es

Agradecimientos

- Gracias al gul por organizar estas jornadas
- Gracias a David por convencerme de dar esta charla
- Gracias a todos vosotros por venir


www.GUL.es