Sistemas Operativos

Curso

2014-2015

Práctica 4

Implementación de un driver sencillo en Linux

J.C. Sáez

Contenido

Contenido

Introducción

Objetivo

- Familiarizarse con los conceptos básicos de la programación de módulos del kernel Linux

Requisitos

1. Es obligatorio usar la máquina virtual de Debian
   - Versión específica del kernel Linux (2.6.32-5)
   - Acceso como administrador (*root*)
     - Ejecutar comando como root vía sudo + password "usuarioso":

       `$ sudo <comando>`
     - Iniciar `shell` de root vía sudo:

       `$ sudo -i`
     - Iniciar `shell` de root vía su + password "r00tSO":

       `$ su -`
2. Necesario usar un PC con teclado estándar para hacer la práctica

Contenido

Módulos cargables del kernel Linux (I)

¿Qué es un módulo cargable?

- Un "fragmento de código" que puede cargarse/descargarse en el mapa de memoria del SO (kernel) bajo demanda
- Sus funciones se ejecutan en modo kernel (privilegiado)
  - Cualquier error fatal en el código "cuelga" el SO
  - Herramientas de depuración menos elaboradas
    - `printk()`: Imprimir mensajes en fichero de log del kernel
    - `dmesg` : Muestra contenido del fichero de log del kernel

También existe soporte para módulos cargables en otros sistemas tipo UNIX (BSD, Solaris) y en MS Windows

Módulos cargables del kernel Linux (II)

Ventajas de los módulos del kernel

1. Reducen el *footprint* del kernel del SO
   - Cargamos únicamente los componentes SW (módulos) necesarios
2. Permiten extender la funcionalidad del kernel en caliente (sin tener que reiniciar el sistema)
   - Mecanismo para implementar/desplegar *drivers*
3. Permiten un diseño más *modular* del sistema

Módulos cargables del kernel Linux (III)

- Los módulos disponibles para nuestro kernel se encuentran en un directorio predefinido
  - `/lib/modules/$KERNEL_VERSION`
    - `$KERNEL_VERSION` para el kernel en ejecución puede obtenerse con `uname -r`
- Podemos saber qué módulos que están cargados con `lsmod`

Terminal

```
usuarioso@debian:~$ lsmod
Module                  Size  Used by
mperf                    935  0
cpufreq_stats           2139  0
bluetooth              55448  2
cpufreq_powersave        650  0
cpufreq_userspace       1464  0
cpufreq_conservative     3791  0
binfmt_misc             4994  1
uinput                  5172  1
fuse                   49890  3
acpiphp                12757  0
loop                   10809  0
tpm_tis                 5725  0
...
```

Anatomía de un módulo cargable

- En lugar de una función `main()`, el código de un módulo tiene funciones `init_module()` y `cleanup_module()`
  - `init_module()` se invoca cuando se carga el módulo del kernel
  - `cleanup_module()` se invoca al eliminar/descargar el módulo del kernel
- Al compilar el módulo se genera un fichero `.ko` que es un fichero objeto ELF (*Executable and Linkable Format*) especial

Carga y descarga de módulos

- Para cargar el módulo se usa el comando `insmod`

```
$ insmod mi_modulo.ko
```

- Un módulo puede descargarse con `rmmod`

```
$ rmmod mi_modulo
```

- Solo el administrador (*root*) puede ejecutar ambos comandos
  - Aconsejable utilizar `sudo <comando>`

Módulo del kernel ("Hello World")

hello.c

```c
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */

int init_module(void)
{
    printk(KERN_INFO "Hello world.\n");

    /*
     * A non 0 return means init_module failed; module can't be loaded.
     */
    return 0;
}

void cleanup_module(void)
{
    printk(KERN_INFO "Goodbye world.\n");
}
```

Compilación de Módulos

- Compilación gestionada mediante un fichero *Makefile*
  - Es necesario tener instalados los ficheros de cabecera (*headers*) del kernel en ejecución
    - Ya instalados en la máquina virtual
  - Compilar con "`make`" y borrar los restos de compilación con "`make clean`"

Makefile (módulo de un solo fichero .c)

```
obj-m =  hello.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
```

Ejemplo: compilación, carga y descarga (I)

Terminal

```
usuarioso@debian6:~/FicherosP4/Hello$ ls
hello.c  Makefile
usuarioso@debian6:~/FicherosP4/Hello$ make
make -C /lib/modules/2.6.32-5-686/build M=/home/usuarioso/FicherosP4/Hello modules
make[1]: se ingresa al directorio `/usr/src/linux-headers-2.6.32-5-686'
  CC [M]  /home/usuarioso/FicherosP4/Hello/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/usuarioso/FicherosP4/Hello/hello.mod.o
  LD [M]  /home/usuarioso/FicherosP4/Hello/hello.ko
make[1]: se sale del directorio `/usr/src/linux-headers-2.6.32-5-686'
usuarioso@debian6:~/FicherosP4/Hello$ sudo insmod hello.ko
usuarioso@debian6:~/FicherosP4/Hello$ lsmod | head
Module                  Size  Used by
hello                    528  0
ppdev                   4058  0
lp                      5570  0
sco                     5885  2
bridge                 32943  0
stp                      996  1 bridge
bnep                    7408  2
rfcomm                 25147  8
l2cap                  21745  16 bnep,rfcomm
```

Ejemplo: compilación, carga y descarga (II)

Terminal

```
usuarioso@debian6:~/FicherosP4/Hello$ dmesg | tail
[ 7123.146443] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 7123.146445] usb 2-2.1: Product: Virtual Bluetooth Adapter
[ 7123.146447] usb 2-2.1: Manufacturer: VMware
[ 7123.146449] usb 2-2.1: SerialNumber: 000650268328
[ 7123.146608] usb 2-2.1: configuration #1 chosen from 1 choice
[ 7322.054847] hello: module license 'unspecified' taints kernel.
[ 7322.054851] Disabling lock debugging due to kernel taint
[ 7322.067488] Hello world
usuarioso@debian6:~/FicherosP4/Hello$ sudo rmmod hello
usuarioso@debian6:~/FicherosP4/Hello$ dmesg | tail
[ 7123.146445] usb 2-2.1: Product: Virtual Bluetooth Adapter
[ 7123.146447] usb 2-2.1: Manufacturer: VMware
[ 7123.146449] usb 2-2.1: SerialNumber: 000650268328
[ 7123.146608] usb 2-2.1: configuration #1 chosen from 1 choice
[ 7322.054847] hello: module license 'unspecified' taints kernel.
[ 7322.054851] Disabling lock debugging due to kernel taint
[ 7322.067488] Hello world.
[ 7369.231062] Goodbye world.
usuarioso@debian6:~/FicherosP4/Hello$ lsmod | head
Module                  Size  Used by
ppdev                   4058  0
lp                      5570  0
sco                     5885  2
bridge                 32943  0
```

Contenido

Dispositivos de caracteres

- Un dispositivo de caracteres es un tipo de fichero especial en UNIX
  - Abstracción software que proporciona el SO
  - El SO expone algunos dispositivos HW a los programas mediante dispositivos de caracteres
    - Ejemplo: terminales, puertos serie, ...
    - *No necesariamente han de estar asociados a dispositivos HW*
  - Los programas de usuario pueden acceder a ellos como si fueran ficheros convencionales
    - Tienen presencia en el sistema de ficheros (ej: `/dev/ttyS0`)
    - `open()`, `read()`, `write()`, `close()`

Dispositivos de caracteres

- El par (*major number*, *minor number*) identifica de forma únivoca a cada dispositivo de caracteres del sistema
  - *major number* → identificador del driver que lo gestiona
  - *minor number* → identificador secundario que permite al driver distinguir entre múltiples dispositivos que gestione
- En Linux, la mayor parte de los *drivers* de dispositivo se implementan como módulos cargables del kernel:
  1. Implementan una interfaz especial
  2. Se registran como *driver* de dispositivo de caracteres

Interfaz *Driver* de disp. de caracteres

Interfaz de Operaciones

```
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 (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, 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 (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
....
};
```

Implementación de un driver

- Crear un módulo del kernel con funciones `init_module()` y `cleanup_module()`
- Definir variable global de tipo `struct file_operations`
  - Especifica qué operaciones de dispositivo de caracteres se implementan y su asociación con las funciones del módulo

```
struct file_operations fops = {
    .read = device_read,    //read()
    .write = device_write,  //write()
    .open = device_open,    //open()
    .release = device_release  //close()
};
```

- Implementar las operaciones de la interfaz para conseguir la funcionalidad deseada

Implementación de un driver (cont.)

- En la función de inicialización, registrar el módulo como *driver* de dispositivo de caracteres mediante `register_chrdev()`
  - Si pasamos un 0 como primer parámetro, nos devuelve el *major number* asignado automáticamente

```
int register_chrdev(unsigned int major, const char *name,
                                struct file_operations *fops);
```

- En la función *cleanup* del módulo, desregistrar el módulo como *driver* de dispositivo de caracteres mediante `unregister_chrdev()`
  - Pasar el *major number* asignado por `register_chrdev()` como primer parámetro

```
int unregister_chrdev(unsigned int major, const char *name);
```

Ejemplo: chardev.c

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

...

int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

#define DEVICE_NAME "chardev" /* Dev name as it appears in /proc/devices */

...

static int Major;    /* Major number assigned to our device driver */

...

static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release
};
```

Ejemplo: chardev.c (cont.)

```
/* This function is called when the module is loaded */
int init_module(void) {
    Major = register_chrdev(0, DEVICE_NAME, &fops);

    if (Major < 0) {
      printk(KERN_ALERT "Registering char device failed with %d\n", Major);
      return Major;
    }

    printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
    printk(KERN_INFO "the driver, create a dev file with\n");
    printk(KERN_INFO "'sudo mknod -m 666 /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
    printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
    printk(KERN_INFO "the device file.\n");
    printk(KERN_INFO "Remove the device file and module when done.\n");

    return SUCCESS;
}

/* This function is called when the module is unloaded */
void cleanup_module(void) {
    unregister_chrdev(Major, DEVICE_NAME);
}

...
```

Operaciones `read()`, `write()`

```
static ssize_t device_read(struct file *file, char *buff, size_t len, loff_t *off);
static ssize_t device_write(struct file *file, const char *buff, size_t len, loff_t
      *off);
```

Parámetros relevantes

- `buff`: buffer de bytes o caracteres donde el usuario nos pasa los datos (`write()`) o donde debemos devolverle los datos (`read()`)
- `len`
  - Lectura → número máximo de bytes/caracteres que podemos escribir en `buff`
  - Escritura → número máximo de bytes/caracteres que el usuario escribió y se almacenan en `buff`

Valor de retorno

- Devuelve número de bytes que el kernel lee de `buff` (en `write()`) o escribe en `buff` (`read()`)
  - 0 →fin de fichero en `read()` (no hay nada más que leer)
- $< 0 \rightarrow$ error

Copia espacio usuario $\Longleftrightarrow$ espacio kernel (I)

- Las operaciones `read()` y `write()` de una entrada `/proc` aceptan como parámetro un puntero al buffer del proceso de usuario (espacio de usuario)
  - Puntero pasado en llamada al sistema `read()` o `write()`
- No debemos confiar en los punteros al espacio de usuario
  - Puntero NULL
  - Región de memoria a la que el proceso no tiene acceso

Copia espacio usuario $\Longleftrightarrow$ espacio kernel (II)

- Siempre se ha de trabajar con una copia privada de los datos en espacio de kernel
  - Por ejemplo, declarar array `char kbuf[MAX_CHARS]` local a las funciones read() y write()
  - En `read()`: trabajar sobre `kbuf` + copiar contenido de `kbuf` a buffer de usuario con `copy_to_user()`
  - En `write()`: copiar datos de buffer de usuario a `kbuf` (`copy_from_user()` + realizar procesamiento sobre `kbuf`

<asm/uaccess.h>

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

- Semántica de copia similar a `memcpy()`
- Ambas funciones devuelven el número de bytes que NO pudieron copiarse

Ejemplo: chardev (Compilación y Carga)

Terminal

```
usuarioso@debian6:~/FicherosP4/Chardev$ ls
chardev.c  Makefile
usuarioso@debian6:~/FicherosP4/Chardev$ make
make -C /lib/modules/2.6.32-5-686/build M=/home/usuarioso/FicherosP4/Chardev modules
make[1]: se ingresa al directorio `/usr/src/linux-headers-2.6.32-5-686'
  CC [M]  /home/usuarioso/FicherosP4/Chardev/chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/usuarioso/FicherosP4/Chardev/chardev.mod.o
  LD [M]  /home/usuarioso/FicherosP4/Chardev/chardev.ko
make[1]: se sale del directorio `/usr/src/linux-headers-2.6.32-5-686'
usuarioso@debian6:~/FicherosP4/Chardev$ sudo insmod chardev.ko
usuarioso@debian6:~/FicherosP4/Chardev$ lsmod | head
Module                  Size  Used by
chardev                 1604  0
ppdev                   4058  0
lp                      5570  0
sco                     5885  2
bridge                 32943  0
stp                      996  1 bridge
bnep                    7408  2
rfcomm                 25147  8
l2cap                  21745  16 bnep,rfcomm
```

Ejemplo: chardev (Listado drivers)

Terminal

```
usuarioso@debian6:~/FicherosP4/Chardev$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 99 ppdev
128 ptm
136 pts
180 usb
189 usb_device
216 rfcomm
251 chardev
252 hidraw
253 bsg
254 rtc
```

Invocar funciones del driver

Para usar las funciones del driver:

1. Crear un fichero especial de caracteres con `mknod` (como root) que tenga el mismo *major number* con el que se registró el driver
   - `sudo mknod <ruta_fich_especial> c <major> <minor>`
     - Recomendable usar mknod con la opcion `-m 666` (todos tendrán permiso de lectura/escritura)
   - El *major number* se puede consultar en el fichero especial `/proc/devices`
2. Acceder al dispositivo creado:
   - Desde un programa de usuario: `open()`, `read()`, `write()`, `close()`
   - Desde el shell: `cat`, `echo`
     - `cat <ruta_dispositivo>` → Lee del dispositivo hasta EOF (potencialmente varios read()') y muestra contenido recibido por pantalla
     - `echo "hola" > <ruta_dispositivo>` → Escribe la cadena "hola\n" (sin el '\0' al final) en el dispositivo

Ejemplo: chardev (Creación del dispositivo)

Terminal

```
usuarioso@debian6:~/FicherosP4/Chardev$ dmesg | tail
...
[ 9903.544133] I was assigned major number 251. To talk to
[ 9903.544139] the driver, create a dev file with
[ 9903.544144] 'sudo mknod -m 666 /dev/chardev c 251 0'.
[ 9903.544147] Try various minor numbers. Try to cat and echo to
[ 9903.544151] the device file.
[ 9903.544154] Remove the device file and module when done.
usuarioso@debian6:~/FicherosP4/Chardev$ sudo mknod -m 666 /dev/chardev c 251 0
usuarioso@debian6:~/FicherosP4/Chardev$ stat /dev/chardev
  File: «/dev/chardev»
  Size: 0           Blocks: 0          IO Block: 4096   fichero especial de caracteres
Device: 5h/5d   Inode: 23760       Links: 1     Device type: fb,0
Access: (0666/crw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2015-01-08 17:32:43.663863059 +0100
Modify: 2015-01-08 17:32:43.663863059 +0100
Change: 2015-01-08 17:32:43.663863059 +0100
usuarioso@debian6:~/FicherosP4/Chardev$ ls -l /dev/chardev
crw-rw-rw- 1 root root 251, 0 ene  8 17:32 /dev/chardev
usuarioso@debian6:~/FicherosP4/Chardev$
```

Ejemplo: chardev (Manipulación del dispositivo)

Terminal

```
usuarioso@debian6:~/FicherosP4/Chardev$ cat /dev/chardev
I already told you 0 times Hello world!
usuarioso@debian6:~/FicherosP4/Chardev$ cat /dev/chardev
I already told you 1 times Hello world!
usuarioso@debian6:~/FicherosP4/Chardev$ cat /dev/chardev
I already told you 2 times Hello world!
usuarioso@debian6:~/FicherosP4/Chardev$ echo Hola > /dev/chardev
bash: echo: error de escritura: Operación no permitida
usuarioso@debian6:~/FicherosP4/Chardev$
```

API módulos del kernel

API módulos

- En el kernel no hay libc, sólo existen algunas funciones implementadas
- Los módulos del kernel pueden invocar funciones del kernel para asignar memoria, administrar temporizadores, etc.
- Uso avanzado API módulos: **Optativa "Arquitectura Interna de Linux y Android"**

Manejo de Cadenas

- `sprintf`, `strcmp`, `strncmp`, `sscanf`, `strcat`, `memset`, `memcpy`, `strtok`, ...

Reservar y liberar memoria dinámica `<linux/vmalloc.h>`

```
void *vmalloc( unsigned long size );
void vfree( void *addr );
```

Otras funciones útiles

Incrementar/decrementar contador de referencia del módulo

- `try_module_get(THIS_MODULE);`
- `module_put(THIS_MODULE);`

Copia espacio usuario $\Longleftrightarrow$ kernel `<asm/uaccess.h>`

- `copy_from_user(), copy_to_user(), get_user(), put_user()`

Contenido

Antes de empezar...

1. Leer detenidamente el guión
   - Más información en Cap. 1-4 "The Linux Kernel Module Programming Guide"
     - `http://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html`
     - `http://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf`
2. Realizar los ejercicios del guión
   - `hello.c`: "Hola mundo" de los módulos del kernel
   - `chardev.c`: *driver* que gestiona dispositivos de caracteres ficticios

Desarrollo de la práctica

La práctica consta de dos partes

- (**Parte A**) Crear un driver (`chardev_leds.c`) que controle los *leds* del teclado de un PC
  - Código de partida: ejemplo chardev.c
  - El nuevo módulo gestionará el dispositivo de caracteres `/dev/leds`
    - Permitirá al usuario modificar el estado de los LEDs escribiendo en `/dev/leds`
    - Más información en el guión
- (**Parte B**) Escribir un programa de usuario (`leds_user.c`) que modifique el estado de los leds del teclado usando el driver de la parte A
  - El programa debe acceder a `/dev/leds` mediante las llamadas al sistema `open()`, `write()` y `close()`
  - Queda a elección del alumno el tipo de interacción que el programa realice con el dispositivo
    - Ejemplo: Implementar un contador binario con los LEDs
    - Aquellas soluciones más ingeniosas obtendrán mayor calificación

Entrega de la práctica

- Durante la sesión de la P4 (26 de enero)
  - El profesor de laboratorio realizará preguntas a cada grupo sobre el desarrollo de la práctica
  - Obligatorio mostrar el funcionamiento de la práctica durante la sesión
  - En esta práctica no habrá parte extra
- Entrega en fichero comprimido
  - `L<num_laboratorio>_P<num_puesto>_Pr4.tar.gz`
  - Debe incluir un *Makefile* para compilar el módulo
    - El programa `leds_user.c` puede compilarse manualmente:

```
$ gcc -Wall -g leds_user.c -o leds_user
```

Estructura entrega (en fichero .zip o .tar.gz)