Práctica 4

Implementación de un driver sencillo en Linux

Introducción y Objetivos

El sistema operativo se debe comunicar con múltiples componentes hardware y periféricos. Para ello se deben emplear una gran variedad de protocolos de comunicación diferentes, que permiten la interacción del SO con estos componentes (teclado, ratón, disco, puertos serie, puertos paralelos, tarjetas de red,...). En todos los sistemas operativos modernos esta interacción la lleva a cabo una parte del sistema operativo llamada driver o controlador software (se debe distinguir del controlador hardware). El driver encapsula las particularidades de un dispositivo hardware específico y expone un conjunto de operaciones ejecutadas sobre una interfaz estándar.

En esta práctica nos familiarizaremos con los drivers y cómo se organizan y se implementan bajo Linux. Al final del guión, pondremos en práctica el conocimiento adquirido implementando un driver que controle los leds del teclado.

Bibliografía

- P.J. Salzman [et al.], The Linux Kernel Module Programming Guide. 2007-05-18 ver 2.6.4.
  - `http://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf`
- Kernel Documentation: `http://www.kernel.org/doc/Documentation/`

Ejercicios

Ejercicio 1

Compilar el módulo de ejemplo "hello.c". Insertar el módulo en el kernel con el comando `sudo insmod hello.ko`. Para verificar que el módulo se insertó correctamente, ejecutar el comando `lsmod` y chequear el "log" del sistema con `dmesg` o `dmesg | tail`. En este fichero de "log" se puede encontrar el mensaje que el módulo imprimió con `printk()` en su función de inicialización. Finalmente descargar el módulo usando `sudo rmmod hello`.

Ejercicio 2

Compilar el módulo chardev.c, que implementa un driver que gestiona dispositivos de caracteres ficticios. Después de compilarlo, insertar el módulo en el kernel con el comando `sudo insmod chardev.ko`. Para verificar que el módulo se insertó correctamente, chequear el "log" del sistema con `dmesg` o `dmesg | tail`. En este fichero de "log" se puede encontrar el *major number* asignado a este módulo y el comando para crear un fichero de dispositivo para este módulo:

```
$ sudo mknod /dev/chardev -m 666 c 251 0
```

Una vez que el fichero de dispositivo fue creado, podemos leer del driver del dispositivo como sigue:

```
$ cat /dev/chardev
I already told you 0 times Hello world!
$ cat /dev/chardev
I already told you 1 times Hello world!
$ cat /dev/chardev
I already told you 2 times Hello world!
```

Si intentamos escribir en el dispositivo obtendremos un mensaje de error, en el terminal o en el fichero de "log":

```
$ echo "Hello" > /dev/chardev
bash: echo: error de escritura: Operación no permitida
```

¿Por qué aparece este error?
No obstante, enhorabuena, ¡acabamos de crear nuestro primer driver!.

Desarrollo de la práctica - Control de los leds del teclado

Los leds del teclado se pueden controlar directamente usando los puertos de E/S. Los puertos de E/S son uno de los mecanismos por los que la CPU se comunica con todos los controladores y periféricos de un computador. Cada puerto tiene una dirección mediante la cual puede leer o escribir. Cada dispositivo tiene asignado un rango de puertos de E/S que pueden usarse para comunicarse con él. Para saber qué puertos se asignan a los dispositivos se puede consultar el fichero especial `/proc/ioports`. Ahí podemos ver cómo el teclado usa el rango 0x60-0x6F. En esta práctica, para controlar los leds del teclado, sólo necesitaremos el puerto 0x60.

Para controlar los leds debemos ejecutar las siguientes acciones:

- Escribir el comando 0xED en el puerto 0x60. Esto le dice al controlador del teclado que le vamos a enviar un comando para modificar los leds
- El controlador de teclado devuelve al puerto 0x60 el ACK (del inglés acknowledgement) 0xFA cuando esté preparado para recibir la señal.
- El controlador espera un valor (byte) en el puerto 0x60 para configurar los leds:
    - bit 0: scroll lock
    - bit 1: num lock
    - bit 2: caps lock
    - bits 3-7: se ignoran

Hay instrucciones especiales para acceder a puertos de E/S. Estas instrucciones privilegiadas, no obstante, pueden ser usadas únicamente en modo kernel, es decir, un programa de usuario normal no puede acceder a los puertos de E/S. El acceso por tanto debe hacerse desde el código del kernel o desde un módulo. Para leer y escribir en un puerto de E/S se deben usar las siguientes funciones[1]:

- `outb(valor, puerto)` → Escribe un byte en un puerto
- `inb(puerto)` → Lee un valor de un puerto

Hay también funciones especiales para leer y escribir valores de tipo palabra y palabra larga (`outw`, `outl`, `inw`, `inl`). Usando estas funciones, el código que controla los leds del teclado podría ser el siguiente:

```
retries = 5;
timeout = 1000;
state = ...; // La configuracion de los leds (0x04 por ejemplo)
outb(0xed,0x60); // Le decimos al teclado que queremos modificar los leds
udelay(timeout);
while (retries!=0 && inb(0x60)!=0xfa) { // esperamos al controlador
  retries--;
  udelay(timeout);
}
if (retries!=0) { // comprobamos que el teclado esta listo
  outb(state,0x60);
}
```

Parte A

Escribir un nuevo driver `chardev_leds.c` que controle los leds de un teclado estándar: el num-lock, caps-lock y scroll-lock. El driver debe implementarse como un módulo del kernel que, en tiempo de carga, se registre a sí mismo como un driver de dispositivo de caracteres. Se debe poder interactuar con el driver mediante un nuevo fichero de dispositivo, `/dev/leds`, como sigue: escribiendo un 1 a este dispositivo se debería encender el led num-lock, con 2 se debería encender el caps-lock, con 3 se debería encender el led de scroll-lock, 12 debería encender num-lock y caps-lock leds, y así sucesivamente:

Comando	Num Lock	Caps Lock	Scroll Lock
`echo 1 > /dev/leds`	ON	OFF	OFF
`echo 123 > /dev/leds`	ON	ON	ON
`echo 32 > /dev/leds`	OFF	ON	ON
`echo > /dev/leds`	OFF	OFF	OFF
`echo 22 > /dev/leds`	OFF	ON	OFF

Para crear el driver se recomienda partir del código ejemplo `chardev.c`. A la hora de implementar la operación de escritura sobre el dispositivo de caracteres (`device_write()`) no se debe trabajar directamente con el parámetro `buff` de la llamada, array que almacena los caracteres que escribe el usuario[2] con `echo`. No podemos confiar en la integridad de ese parámetro,

[1] Para poder usar estas dos funciones desde un módulo del kernel es necesario incluir el fichero de cabecera `<asm/io.h>`.

[2] El array `buff` almacena tantos caracteres como se especifican en el parámetro 'len' de la llamada `device_write()`. Se ha de tener en cuenta que el caracter terminador ('\0') NO está incluido al final del array.

ya que es un puntero al espacio de usuario. Por ese motivo debemos copiar de forma segura los bytes de `buff` a un array auxiliar (variable local) usando `copy_from_user()`. En caso de que la copia falle, se ha de devolver un error `-EINVAL`. Por el contrario, si la copia es satisfactoria analizaremos los bytes copiados en el array auxiliar para determinar qué leds deben encenderse/apagarse y realizaremos las acciones necesarias para que esto ocurra.

Para poder compilar el driver se ha de construir un *Makefile* modificando el que se proporciona con el ejemplo `chardev`. Para ello, se debe actualizar el nombre del fichero objeto a generar, teniendo en cuenta el nombre del fichero .c que contiene el código de nuestro módulo.

Parte B

Escribir un programa de usuario `leds_user.c` que controle los leds del teclado usando el driver desarrollado en el apartado anterior. El programa debe acceder al fichero de dispositivo `/dev/leds` únicamente mediante las llamadas al sistema `open()`, `write()` y `close()`. El programa escribirá en el fichero de dispositivo las cadenas correspondientes de tal forma que los leds se apaguen y se enciendan en una secuencia predefinida.

Queda a elección del estudiante elegir la secuencia, que puede ser tan sencilla como un orden circular, un contador binario o algo tan fantástico como encender los leds cada vez que haya una operación de escritura en el disco duro (imitando el led del disco duro), o lo propio en la tarjeta de red. Se reservará nota de la práctica para premiar las soluciones más ingeniosas.

Este programa de usuario se puede compilar manualmente con el siguiente comando:

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