Práctica 4: Implementación de un driver sencillo en Linux

Índice

1 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.

2 Bibliografía

- The Linux Kernel Module Programming Guide (Cap. 1-3):
  - https://sysprog21.github.io/lkmpg/
- Robert Love; *Linux Kernel Development*. Addison Wesley, 3rd Edition. 2010
  - Cap. 17 *"Devices and Modules"*
- Kaiwan N. Billimoria. Linux Kernel Programming. Packt Publishing. 1st edition. 2021
  - Caps. 4 y 5: "Writing Your First Kernel Modules"
- Kaiwan N. Billimoria. Linux Kernel Programming. Part 2 - Char Device Drivers and Kernel Synchronization. Packt Publishing. 1st edition. 2021

3 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 `sudo dmesg` o `sudo 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 `sudo dmesg` o `sudo dmesg | tail`. En este fichero de "log" se puede encontrar el *major number* asignado a este *driver* y el comando para crear un fichero de dispositivo gestionado por el *driver*:

```
$ sudo mknod /dev/chardev -m 666 c 248 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!. El comportamiento del driver es el esperado.

Ejercicio 3

Estudiar el módulo del kernel `modleds.c` que interactúa con el driver de teclado de un PC para encender/apagar los leds num-lock, caps-lock y scroll-lock. Al cargar el módulo se encienden los tres leds del teclado y al descargarlo se apagan.

En el código del ejemplo se ha de prestar especial atención a las siguientes funciones:

- `get_kbd_driver_handler()`: Se invoca durante la carga del módulo para obtener un puntero al manejador del driver de teclado/terminal.
- `set_leds(handler,mask)`: Permite establecer el valor de los leds. Acepta como parámetro un puntero al manejador del driver y una máscara de bits que especifica el estado de cada LED (1 → ON; 0 → OFF). Cada bit controla un led específico del teclado:
  - bit 0: scroll lock ON/OFF
  - bit 1: num lock ON/OFF
  - bit 2: caps lock ON/OFF

- bits 3-31: se ignoran

Ejercicio 4

Escribir un nuevo driver `chardev_leds.c` que controle los leds de un teclado estándar. 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 en este fichero de dispositivo se debería encender el led num-lock, con 2 se debería encender el led caps-lock, con 3 se debería encender el led scroll-lock, 12 debería encender los leds num-lock y caps-lock, y así sucesivamente:

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

Para crear el driver se ha de reutilizar el código de los ejemplos `chardev.c` y `modleds.c`. Nótese que el driver de teclado sólo permite que el usuario *root* o un usuario con permisos de `sudo` modifique el estado de los leds del teclado (p.ej., mediante función `setleds()` definida en el módulo del Ejercicio 3). Por lo tanto, es preciso invocar la operación de escritura sobre `/dev/leds` desde un proceso del usuario *root* o invocando `echo` mediante el comando `sudo`, como se muestra en la tabla.

A la hora de implementar la operación de escritura (`device_write()`) sobre el dispositivo de caracteres no se debe trabajar directamente con el parámetro `buff` de la llamada (array que almacena los caracteres que escribe el usuario[1] con `echo`). No podemos confiar en la integridad de ese parámetro, 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 `-EFAULT`. 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.

Ejercicio 5

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 llamadas al sistema: `open()`, `write()`, `close()`, etc. 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. Es preciso ejecutar este programa mediante `sudo` para que las peticiones de modificación del estado de los leds que envía el módulo del kernel al driver de teclado se sirvan satisfactoriamente.

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.

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

[1] 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.

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