Práctica 7: Implementación de un driver sencillo en Linux

Índice

1 Introducción y Objetivos

En esta práctica nos familiarizaremos con los drivers y cómo se organizan y se implementan en Linux.

Los ficheros para esta práctica están disponibles en el siguiente enlace: ficheros_p7s.tar.gz.

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

Desarrollar un nuevo driver `buffermod.c` que gestione un buffer de caracteres con capacidad de 512 bytes, y que permita al usuario tanto consultar, como modificar el contenido de este buffer. 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.

La interacción con el driver desde espacio de usuario se llevará a cabo mediante un fichero de dispositivo llamado `/dev/buffermod`. Al escribir una cadena de caracteres en ese fichero especial, la cadena escrita reemplazará el contenido antiguo del buffer de caracteres gestionado por el driver. Asimismo, la lectura de ese fichero especial de caracteres devolverá al usuario el contenido de la última cadena escrita en dicho buffer. Este tipo de interacción se refleja en el siguiente ejemplo de ejecución:

```
## Cargamos módulo (que suponemos ya compilado).
$ sudo insmod buffermod.ko

# A partir de aquí suponemos que se ha creado el fichero especial de caracteres con mknod
## El buffer está inicialmente vacío (cadena ""), por lo que la lectura inicial no devuelve ningun byte
$ cat /dev/buffermod
$

## Alteración del contenido del buffer
$ echo cool_string > /dev/buffermod

## Múltiples lecturas devuelven el mismo valor
$ cat /dev/buffermod
cool_string
$ cat /dev/buffermod
cool_string

## Alteración del contenido del buffer y consulta del mismo
$ echo cooler string > /dev/buffermod
```

```
$ cat /dev/buffermod
cooler string
```

Para crear el driver se ha de reutilizar el código del ejemplo `chardev.c`. También se han de tener en cuenta las siguientes consideraciones:

- En este driver solo es preciso implementar las operaciones de lectura y escritura (`read` y `write`) sobre el fichero de dispositivo. La implementación de estas operaciones difiere considerablemente a la del driver `chardev.c`. En cualquier caso, al reusar el código de `chardev.c` se puede descartar la implementación de las operaciones de apertura (`open`) y cierre (`release`) .
- A la hora de implementar la operación de escritura (`write`) sobre el dispositivo de caracteres, se debe prestar especial atención al parámetro `buff` de la llamada, es decir, al array que almacena los caracteres que escribe el usuario. Los aspectos más críticos a considerar al gestionar este array son los siguientes
  - El caracter terminador ('\0') NO está incluido al final del array, ya que su contenido no es un *string* de C, sino una secuencia de bytes escrita por el usuario mediante la llamada al sistema `write()`.
  - El array almacena tantos caracteres como se especifican en el parámetro `len` de la operación `write`. Si la longitud de este array supera la capacidad del buffer (512, incluyendo el caracter terminador), la operación `write` devolverá un error.
  - 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` al buffer de caracteres global del módulo (variable global) usando la función`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 se deberá cerrar manualmente la cadena copiada a buffer de caracteres global del módulo, insertando el caracter terminador al final.
- Se ha de tener en cuenta que la operación `read` debe retornar el valor 0 –fin de fichero– en algún momento. De no ser así, la lectura realizada sobre el fichero de dispositivo con el comando `cat` no terminará. Se aconseja tomar la decisión sobre el valor a retornar en base a la ubicación actual del puntero de posición del fichero (parámetro `off` de la operación `read`). Una posible estrategia de implementación sería devolver todo el contenido del buffer de caracteres en la primera lectura (cuando `*off==0`) y retornar fin de fichero en sucesivas lecturas (si `*off>0`).

- 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.