# Práctica 2

# Creación de drivers para dispositivos hardware



## 2.1. Objetivos

Poner en práctica el conocimiento adquirido en clase implementando un driver que controle los leds del teclado.

## 2.2. Ejercicios

**Ejercicio 1:** Compilar el ejemplo `hello.c` cuyo código ha sido proporcionado en teoría. Insertar el módulo en el kernel con el comando `insmod hello.ko` (recuerda que debes insertar el módulo como root). Para verificar que el módulo se insertó correctamente, chequear el `log` del sistema en `/var/log/messages`. En este fichero de `log` se puede encontrar el mensaje imprimido por el módulo por su función de inicialización.

**Ejercicio 2:** Compilar el ejemplo `chardev.c` con el código proporcionado en teoría. Después de compilar, insertar el módulo en el kernel con el comando `insmod chardev.ko` (recuerda que debes insertar el módulo como root). Para verificar que el módulo se insertó correctamente, chequear el `log` del sistema en `/var/log/messages`. En este fichero de `log` se puede encontrar el número major asignado a este módulo y crear un fichero de dispositivo para este módulo:

```
$ mknod /dev/chardev 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
$ rmmod /dev/chardev
```

¿Por qué aparece este error? A pesar del error, enhorabuena, ¡acabamos de crear nuestro primer driver!

## 2.3. 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 lo que la UCP 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 `/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 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 pueden ser usadas únicamente en modo kernel, es decir, un programa normal no puede acceder a los puertos de E/S (aunque lo ejecute el usuario `root`). Para leer y escribir en un puerto de E/S se deben usar las siguientes funciones:

- `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 lo siguiente:

```
#include <asm/uaccess.h>    // Para put_user
#include <asm/io.h>         // Para outb e inb
// ...
        retries = 5;
        timeout = 1000;
        // Almacenamos la configuración de los leds (0x04 por ejemplo)
        state = ...;
        // Le decimos al teclado que queremos modificar los leds
        outb(0xed,0x60);
        udelay(timeout);
        // Esperamos al controlador
        while(retries!=0 && inb(0x60)!=0xfa) {
                retries--;
                udelay(timeout);
        }
        // comprobamos que el teclado está listo
        if (retries!=0) {
                outb(state,0x60);
        }
// ...
```

### 2.3.1. Parte A

Escribir un nuevo `driver` 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 se registre a sí mismo como un dispositivo tipo carácter cuando se cargue. 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 |

### 2.3.2. Parte B

Escribir un programa en el espacio de usuario que controle los leds del teclado usando el `driver` desarrollado en el apartado anterior. El programa deberá escribir en el fichero de dispositivo `/dev/leds` 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.