Práctica III

Platform Device Drivers & Device Tree OF Matching Style

Implementación de Manejadores de Dispositivos

Maestría en Sistemas Embebidos

Año 2024

Autor

Mg. Ing. Pablo Slavkin

Mg. Ing. Hanes Nahuel Sciarrone

Tabla de contenido

Registro de cambios

Revisión	Cambios realizados	Fecha
1.0	Creación del documento	12/11/2021
1.1	Corrección de titulo a “Práctica III”	15/11/2021
1.2	Actualización de año de dictado	14/11/2022
1.3	Modificación de encabezado	16/02/2024

Platform drivers

En esta práctica se pretende escribir y testear un módulo sencillo del tipo platform basando la estructura en el driver del tipo **misc** (utilizando el *misc framework*). El lector debe recordar que:

- Un platform device se denomina así porque no existe forma de descubrir el dispositivo presente a priori si no se declara explícitamente.
- Recibe su nombre porque se conecta a un platform bus.
- De ninguna manera un platform device deja de ser un char device, porque debe recordarse que la descripción de character device proviene de cómo se modela la comunicación, como un stream de datos. Nada tiene que ver con las características que hacen que un dispositivo sea del tipo *platform*.

Según los puntos anteriores, lo que se busca en esta práctica es aumentar la especificidad de nuestro driver char genérico, que registra el dispositivo (*major & minor*) y sus *file operations* mediante el *framework misc*, agregando una conexión al platform bus. Este es el pseudo-bus al que se conectan, entre otros, los dispositivos i2c.

Cada platform driver es responsable de instanciar y registrar una instancia de una estructura **platform_driver** dentro del núcleo del LDM. Estos drivers siguen la convención estándar donde el descubrimiento y enumeración se hace por fuera de los drivers (responsabilidad del kernel) y los drivers proveen los métodos **probe()** y **remove()**. Asimismo los drivers soportan notificaciones de manejo de energía y shutdown mediante convenciones estándar.

Los miembros más importantes de la estructura **platform_drive** son los siguientes:

```
struct platform_driver {
   int (*probe)(struct platform_device *);
   int (*remove)(struct platform_device *);
   void (*shutdown)(struct platform_device *);
   int (*suspend)(struct platform_device *, pm_message_t state);
   int (*suspend_late)(struct platform_device *, pm_message_t state);
   int (*resume_early)(struct platform_device *);
   int (*resume)(struct platform_device *);
```

```
    struct device_driver driver;
};
```

En la anterior estructura se puede ver uno de los campos como un puntero a función que apunta a **probe()**. Esta función es invocada cuando el *bus driver* efectúa el apareamiento entre el dispositivo y el driver correspondiente. Así, la función probe() es responsable de inicializar el dispositivo y registrarlo en el framework del kernel correspondiente. Los pasos que se efectúan son los listados a continuación:

1. La función probe() obtiene un puntero a una estructura de dispositivo como argumento. Ejemplos: *struct pci_dev *, struct usb_dev *, struct platform_device *, struct i2c_client **.
2. Luego inicializa el dispositivo, mapea la memoria de entrada/salida, asigna buffers dinámicamente, registra handlers para interrupciones, etc.
3. Registra el dispositivo a un framework específico, como por ejemplo el **misc**.

Las funciones **suspend()** y **resume()** son utilizadas por dispositivos que soportan administración para modos de bajo consumo.

El platform driver responsable del platform device que se haya registrado, debe estar registrado en el **platform core** utilizando la función *platform_driver_register(struct platform_driver *drv)*. Esto usualmente se hace en la función init() y luego se da de baja en la función exit().

Escribiendo un módulo “platform device”

La funcionalidad de este platform driver es la misma que la que tenía el misc char driver implementado en la guía de práctica II. Pero a diferencia de ese módulo, el presente registra el char device en la función probe() en vez de hacerlo dentro de init().

Cuando el módulo es cargado, el platform driver se registra a sí mismo en el platform bus driver utilizando la función **platform_driver_register()**. La función probe() entonces es llamada cuando el platform driver hace un match del valor de uno de sus strings *compatible*, las cuales están declaradas en la estructura **of_device_id** que contiene, con

la propiedad *compatible* declarada en el nodo correspondiente del device tree. Este proceso de asociar un dispositivo con su respectivo driver se denomina **binding**.

Se invita al lector a descargar el archivo **hello_platform_driver.c** y luego a modificar el device tree correspondiente a la SBC utilizada, agregando el siguiente nodo:

```
nodo_imd {
    compatible = "imd,mse_driver";
};
```

Luego de compilar tanto el módulo como el DT, se procede a insertar el módulo:

```
$ insmod hello_platform_driver.ko
```

Al momento de ser insertado, la función **probe()** debería ser llamada, dado que el kernel posee un mecanismo que trata de obtener un match apropiado entre el módulo (driver) cargado y algún dispositivo declarado, en este caso, en el DT. En caso exitoso, dentro del pseudo filesystem sysfs debería existir una entrada correspondiente:

```
$ find /sys -name "*hello_platform_driver*"
```

Así también debería existir una entrada descrita por el nodo del device tree dentro de la jerarquía donde fue declarado. Esto depende de cómo está organizado el DT y es usual encontrar esta entrada en */sys/devices* y algún subdirectorio.

También se debe observar que el driver se registró a si mismo dentro del platform bus:

```
$ ls -l /sys/bus/platform/drivers/

$ ls -l /sys/bus/platform/devices

$ ls /sys/bus/platform/drivers/hello_platform_driver
```

De la misma forma debe poder visualizarse el módulo dentro de /sys y el dispositivo registrado dentro de la clase correspondiente al framework utilizado:

```
$ ls -l /sys/module/hello_platform_driver/drivers
```

```
$ ls /sys/class/misc
```

Al haberse creado la entrada en el sysfs, udev se encarga de crear el file device en devtmpfs

```
$ ls -l /dev
```

Verificado todo esto, se puede correr la aplicación de test que se utilizó en la guía de práctica II:

```
$ ./ioctl_test
```

```
$ rmmod hello_platform_driver
```