Práctica IV

I2C Driver

Implementación de Manejadores de Dispositivos

Maestría en Sistemas Embebidos

Año 202

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	22/11/2021
1.1	Modificación de encabezado	16/02/2024

I2C Subsystem e I2C device drivers

Arquitectura del driver

Cuando un dispositivo para el cual se escribe un driver i2c toma lugar en un bus físico, que en el sistema se representa como un controlador de bus, debe depender del driver que llama el controlador del bus, el cual es responsable de compartir el acceso al bus entre distintos dispositivos. El driver del controlador ofrece una capa de abstracción entre el dispositivo y el bus. Siempre que se efectúe una transacción (lectura, escritura) en un bus I2C, el controlador del bus se hace cargo de esta acción de forma transparente en segundo plano.

Cada controlador de bus exporta un set de instrucciones para simplificar el desarrollo de drivers para los dispositivos que se conectan a ese bus. Esto es verdad no solo para el bus I2C sino para todos los buses físicos en el sistema.

Un driver I2C se representa en el kernel como una instancia de la estructura **i2c_driver**. Un cliente I2C, el cual representa en resumidas cuentas al dispositivo mismo, se representa por una instancia de la estructura **i2c_client**.

Estructura de un driver I2C

La estructura i2c_driver que se instancia en espacio kernel tiene la siguiente declaración:

```
struct i2c_driver {
   /* Standard driver model interfaces */
   int (*probe)(struct i2c_client *, const struct i2c_device_id *);
   int (*remove)(struct i2c_client *);

   /* driver model interfaces that don't relate to enumeration */
   void (*shutdown)(struct i2c_client *);

   struct device_driver driver;
   const struct i2c_device_id *id_table;
};
```

La estructura **i2c_driver** contiene y caracteriza las funciones de acceso general necesarias para el manejo de los dispositivos que reclaman el driver para sí, donde por el contrario la estructura **i2c_client** representa y caracteriza un dispositivo I2C.

```
struct i2c_client {
   unsigned short flags;

   /* chip address - NOTE: 7 bit */
   /* addresses are stored in the */
   /* _LOWER_ 7 bits */

   unsigned short addr;


   char name[I2C_NAME_SIZE];
   struct i2c_adapter *adapter; /* the adapter we sit on */

   /* the device structure */
   struct device dev;

   /* IRQ issued by device */
   int irq;

   struct list_head detected;
};
```

Todos los campos listados son llenados por el kernel segun la informacion de registro que se provee para efectuar el mismo.

La función probe()

Como un dispositivo I2C sigue siendo un platform device, entonces la función **probe()** tiene el mismo significado y funcionamiento. Solamente cambia los argumentos que toma, siendo su prototipo el siguiente:

```
static int foo_probe(struct i2c_client *client, \
                     const struct i2c_device_id *id);
```

Datos específicos de cada dispositivo

El núcleo I2C ofrece la posibilidad de almacenar un puntero a cualquier estructura de datos que se desee, la cual es conocida como datos especificos de cada dispositivo. Para almacenar u obtener estos datos, se utilizan las siguientes funciones:

```
/* set the data */
void i2c_set_clientdata(struct i2c_client *client, void *data);

/* get the data */
void *i2c_get_clientdata(const struct i2c_client *client);
```

La función remove()

Así como el caso de la función **probe()**, la función remove() se utiliza de la misma forma que en un platform device, pero su prototipo cambia, tomando como argumento el cliente que se está removiendo.

```
static int foo_remove(struct i2c_client *client);
```

Funciones específicas del core I2C

Estas son las funciones que se utilizan para establecer una comunicacion I2C simple:

```
int i2c_master_send(struct i2c_client *client, const char *buf,
                    int count);

int i2c_master_recv(struct i2c_client *client, char *buf, int count);
```

Las funciones listadas leen y escriben algunos bytes desde y hacia un cliente. El cliente contiene la dirección I2C, por lo que no hace falta incluirla en las llamadas a estas funciones. El segundo parámetro contiene los bytes a ser leídos o escritos y el tercero la cantidad de bytes a leer o escribir, entendiendo que este número debe ser menor de la longitud del buffer, y asimismo menor a 64k dado que la longitud del mensaje está representada por una variable entera sin signo de 16 bits (u16). Las funciones retornan la cantidad de bytes leídos o escritos de manera satisfactoria.

Asimismo existe una función que envía una serie de mensajes, donde cada mensaje puede ser de lectura o escritura y estos pueden ser mezclados de cualquier manera. Esto es muy útil para completar una transacción en una única llamada.

```
int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msg,
                 int num);
```

Escribiendo un módulo “holamundo_i2c_driver”

Para poder escribir un driver holamundo haciendo uso de las API del núcleo i2c, se utiliza, aunque no es mandatorio, una estructura privada, la cual contenga no solo la estructura que caracteriza el dispositivo sino también la estructura **miscdevice** y cualquier otro dato que el desarrollador entienda necesario.

```
struct ioexp_dev {
   struct i2c_client * client;
   struct miscdevice ioexp_miscdevice;
   char name[9]; /* msedrvXX */
};
```

En ejemplos anteriores se especificó una función open() para trabajar con el framework misc, pero estrictamente esta no es necesaria, dado que el mismo framework genera automáticamente una función open().

También se utiliza la función **devm_kzalloc()** que cumple la función de asignar memoria dinámicamente en espacio kernel y vincular este bloque de memoria reservado a un dispositivo específico, el cual al momento de ser removido, libera automáticamente esta memoria. En específico, el bloque de memoria asignado tiene todos sus bytes a cero, por esto la letra **z** en el nombre de la función.

Teniendo en cuenta esta informacion, descargue el archivo **holamundo_i2c_driver.c** y una vez compilado, inserte el módulo y verifique que la función **probe()** sea ejecutada, y que las entradas en **/sys** sean generadas al igual que en la guia de practica anterior.

Asimismo, verifique que el device file correspondiente sea generado en **/dev** y compruebe el funcionamiento del driver con el programa ya compilado **ioctl_test.c**