14th November 2012

I2c driver in linux

In a nutshell I2C [http://en.wikipedia.org/wiki/I%C2%B2C] is a simple serial bus that is often used to communicate with devices such as EEPROMs and peripherals such as touchscreens – there is also SMBus [http://en.wikipedia.org/wiki/System_Management_Bus] which can sometimes be considered a subset of I2C. *The kernel breaks down I2C into 'Buses' and 'Devices', and then further breaks down buses into 'Algorithms [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L308] ' and 'Adapters [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L331] ', and devices into 'Drivers [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L99] ' and 'Clients [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L172] '.* We'll take a look at these in a little more detail.

Algorithms

An Algorithm performs the actual reading and writing of I2C messages to the hardware – this may involve bit banging GPIO lines or writing to an I2C controller chip. An Algorithm is represented by the very straight-foward 'struct i2c_algorithm [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L308] ' structure and allows you to define function pointers to functions that can write I2C messages (master_xfer) or SMBus messages (smbus_xfer).

Adapters

An Adapter effectively represents a bus – it is used to tie up a particular I2C/SMBus with an algorithm and bus number. It is represented by the 'struct i2c_adapter [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L331] ' structure. If you imagine a system where there are many I2C buses – perhaps two controlled by a controller chip and one bit-banged – then you would expect to see 3 instances of an i2c_adapter and 2 instances of an i2c_algorithm.

Clients

A Client represents a chip (slave) on the I2C/SMBus such as a Touchscreen peripheral and is represented by a 'struct i2c_client [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L172] ' structure. This includes various members such as chip address, name and pointers to the adapter and driver.

Drivers

Finally a driver, represented by a 'struct i2c_driver [http://lxr.linux.no/linux+v2.6.31/include/linux/i2c.h#L99] ' structure represents the device driver for a particular class of I2C/SMBus slave devices, e.g. a touchscreen driver. The structure contains a bunch of function pointers – the ones of interest to us are the 'probe' and 'remove' pointers – which we'll shortly come onto.

So where do we start with writing an I2C slave device driver? We start by populating a 'struct i2c_driver' structure. In this tutorial we'll populate just the 'driver', 'id_table', 'probe', and 'remove' members. We'll also use the 'migor_ts.c [http://lxr.linux.no/linux+v2.6.31/drivers/input/touchscreen/migor_ts.c] ' touchscreen driver as reference code.

```
1  static struct i2c_driver migor_ts_driver = {
2    .driver = {
3       .name = "migor_ts",
4    },
5    .probe = migor_ts_probe,
6    .remove = migor_ts_remove,
7    .id_table = migor_ts_id,
8  };
```

We tell the kernel about our I2C slave driver by registering the 'struct i2c_driver' structure with the I2C core - we do this with a call to 'i2c_add_driver' in our __init function as follows:

```
1  static int __init migor_ts_init(void)
2 {
3         return i2c_add_driver(&migor_ts_driver);
4 }
```

Back to our 'struct i2c_driver' structure – the id_table member allows us to tell the framework which I2C slaves chips we support – id_table is simply an array of 'struct i2c_device_id' – here's the migors:

```
1 static struct i2c_device_id migor_ts_id[] = {
2 { "migor_ts", 0 },
3 { }
4 };
```

So in this case we're saying that we only support a slave chip named 'migor_ts'. This makes us wonder – how does the kernel know which slave chips are present and what they're called? – we know the chip doesn't contain such friendly naming strings and I2C doesn't support enumeration. There is no magic - there are a number of mechanisms to discover and name an I2C slave [http://lxr.linux.no/linux+v2.6.31/Documentation/i2c/instantiating-devices] – the most

and name. For example in the arch/sh/boards/mach-migor/setup.c [http://lxr.linux.no/linux+v2.6.31/arch/sh/boards/mach-migor/setup.c] file we see the following:

```
1 static struct i2c_board_info migor_i2c_devices[] = {
2 {
3 I2C_BOARD_INFO("migor_ts", 0x51),
4 .irq = 38, /* IRQ6 */
5 },
6 };
7
8 i2c_register_board_info(0, migor_i2c_devices, ARRAY_SIZE(migor_i2c_devices));
```

Without going into too much detail here – we populate a 'struct i2c_board_info' structure with the help of a I2C_BOARD_INFO macro - crucially we include a made up name for the slave chip and it's I2C bus address.

So back to the migor driver. During boot, thanks to the 'i2c_register_board_info' call, the kernel will try to find any I2C driver which supports the 'migor_ts' name we provided. Upon finding such a driver, the kernel will call it's 'probe' function. The idea being that the kernel believes it has found an I2C slave and would like us to probe to make sure that we are a suitable driver (It's worth noting that the kernel won't attempt to communicate with this device in any way).

So if all is well – upon start up the kernel should call our probe function – let's take a look.

```
1 static int migor_ts_probe(struct i2c_client *client, const struct i2c_device_id *idp)
2 {
3 ...
4 }
```

Passed along as a parameter to the probe function is a 'struct i2c_client' structure – this represents our I2C slave chip. Typically a probe function where possible will communicate with the device to ensure it really is the device we think it is – this can be done by reading values from a revision ID register or the like. We can communicate with the slave by calling the 'i2c_master_send' function with a pointer to our 'struct i2c_client' – e.g:

```
1 i2c_master_send(client, buf, 1);
```

With regards to driver initialisation this is the last we'll here from the kernel – so the probe function should also register with the input layer (assuming an input device) and perhaps set up an interrupt/timer. It's also a good idea to make a note of the i2c_client for use later on. Now you can talk to your I2C slave device – what else goes on really depends on the type of device which is being supported.

Posted 14th November 2012 by venkatesh yadav

8 View comments

Nandita Gupta December 21, 2013 at 9:14 AM

Extremely good content & gud effort put in xplaining every minute detail of codeflow of I2C Subsystem in driver model.
Appluads!!!
Eagerly waiting for more posts regarding Input Subsystem & Kernel Device Drivers.

Reply

israr February 24, 2014 at 12:35 AM

Excellent Explanation, To the point ,specific .It takes too long to find the same information in Linux Documentation.I want to do a project in I2c Client Driver, it will help me a lot ,thanks a ton.

Reply

asodi.t.venkateswara reddy April 8, 2014 at 10:58 PM

good work.....

Reply

Jamal Mohammad January 31, 2015 at 7:03 PM

good job

Reply

Naveen Kumar April 6, 2015 at 10:13 AM

Great tutorial, keep posting such a nice kernel articles.

Reply

Singh Yuvraj July 16, 2015 at 11:46 AM

really gd doccuments thanks

Reply

thanks, it will help me very much!

Reply

Jagannath Pattar February 20, 2016 at 3:58 AM

Simply good explanation on algorithm, adapter, driver, client and kernel probing for devices.

Reply

Enter your comment...

Comment as: Unknown (Goo ▼ Sign out

Publish Preview Notify me