

[PATCH 1/5] call i2c_probe from i2c core

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/4108.html>

From: Nathan Lutchansky (lutchann_at_litech.org)

Date: 08/15/05

Date: Mon, 15 Aug 2005 13:52:57 -0400

To: LKML <linux-kernel@vger.kernel.org>, lm-sensors <lm-sensors@lm-sensors.org>

Add the `address_data` and `detect_client` fields to the `i2c_driver` structure. If these are set, `i2c` core will call `i2c_probe` directly when `attach_adapter` would have been called. If the `i2c_driver` class field is also set, probing will only be done on adapters with an intersecting class field.

The `attach_adapter` callback will still be called if it is present, but this makes it unnecessary for almost all in-tree `i2c` drivers.

Signed-off-by: Nathan Lutchansky <lutchann@litech.org>

Documentation/i2c/writing-clients | 58 ++++++-----
drivers/i2c/i2c-core.c | 21 ++++++---
include/linux/i2c.h | 11 ++++++
3 files changed, 65 insertions(+), 25 deletions(-)

Index: linux-2.6.13-rc6+gregkh/include/linux/i2c.h

```
-----  
--- linux-2.6.13-rc6+gregkh.orig/include/linux/i2c.h  
+++ linux-2.6.13-rc6+gregkh/include/linux/i2c.h  
@@ -48,6+48,7 @@ struct i2c_algorithm;  
struct i2c_adapter;  
struct i2c_client;  
struct i2c_driver;  
+struct i2c_client_address_data;  
union i2c_smbus_data;  
  
/*  
@@ -113,6+114,7 @@ struct i2c_driver {  
    int id;  
    unsigned int class;  
    unsigned int flags; /* div., see below */  
+ struct i2c_client_address_data *address_data;  
  
    /* Notifies the driver that a new bus has appeared. This routine  
    * can be used by the driver to test if the bus meets its conditions  
@@ -123,6+125,15 @@ struct i2c_driver {
```

Linux-Kernel: [PATCH 1/5] call i2c_probe from i2c core

```
int (*attach_adapter)(struct i2c_adapter *);
int (*detach_adapter)(struct i2c_adapter *);
```

```
+ /* Requests that the driver validate an address on a bus and attach a
+ * new client. If this routine is supplied, it will be called for
+ * each device on new buses that appear, provided the bus class
+ * matches the class field and devices exist at the addresses listed
+ * in the address_data field. For most drivers, this mechanism can
+ * be used instead of an attach_adapter routine.
+ */
+ int (*detect_client)(struct i2c_adapter *, int addr, int kind);
+
+ /* tells the driver that a client is about to be deleted & gives it
+ * the chance to remove its private data. Also, if the client struct
+ * has been dynamically allocated by the driver in the function above,
```

Index: linux-2.6.13-rc6+gregkh/drivers/i2c/i2c-core.c

```
----- linux-2.6.13-rc6+gregkh.orig/drivers/i2c/i2c-core.c
+++ linux-2.6.13-rc6+gregkh/drivers/i2c/i2c-core.c
@@ -193,9 +193,16 @@ int i2c_add_adapter(struct i2c_adapter *
    /* inform drivers of new adapters */
    list_for_each(item,&drivers) {
        driver = list_entry(item, struct i2c_driver, list);
- if (driver->flags & I2C_DF_NOTIFY)
- /* We ignore the return code; if it fails, too bad */
- driver->attach_adapter(adap);
+ if (driver->flags & I2C_DF_NOTIFY) {
+ /* We ignore the return codes; if it fails, too bad */
+ if (driver->attach_adapter)
+ driver->attach_adapter(adap);
+ if (driver->detect_client && driver->address_data &&
+ ((driver->class & adap->class) ||
+ driver->class == 0))
+ i2c_probe(adap, driver->address_data,
+ driver->detect_client);
+ }
    }
}
```

out_unlock:

```
@@ -307,7 +314,13 @@ int i2c_add_driver(struct i2c_driver *dr
    if (driver->flags & I2C_DF_NOTIFY) {
        list_for_each(item,&adapters) {
            adapter = list_entry(item, struct i2c_adapter, list);
- driver->attach_adapter(adapter);
+ if (driver->attach_adapter)
+ driver->attach_adapter(adapter);
+ if (driver->detect_client && driver->address_data &&
+ ((driver->class & adapter->class) ||
+ driver->class == 0))
+ i2c_probe(adapter, driver->address_data,
+ driver->detect_client);
```

```

    }
}

```

Index: linux-2.6.13-rc6+gregkh/Documentation/i2c/writing-clients

```

----- linux-2.6.13-rc6+gregkh.orig/Documentation/i2c/writing-clients
+++ linux-2.6.13-rc6+gregkh/Documentation/i2c/writing-clients
@@ -27,8 +27,10 @@ address.
static struct i2c_driver foo_driver = {
    .owner = THIS_MODULE,
    .name = "Foo version 2.3 driver",
+ .class = I2C_CLASS_HWMON,
    .flags = I2C_DF_NOTIFY,
- .attach_adapter = &foo_attach_adapter,
+ .address_data = &addr_data,
+ .detect_client = &foo_detect_client,
    .detach_client = &foo_detach_client,
    .command = &foo_command /* may be NULL */
}

```

@@ -147,8 +149,8 @@ are defined to help determine what address are defined in i2c.h to help you support them, as well as a generic detection algorithm.

-You do not have to use this parameter interface; but don't try to use
-function i2c_probe() if you don't.
+You do not have to use this parameter interface; but then the i2c core won't
+be able to probe for devices for you.

NOTE: If you want to write a `sensors' driver, the interface is slightly different! See below.

@@ -207,35 +209,49 @@ Attaching to an adapter

Whenever a new adapter is inserted, or for all adapters if the driver is
-being registered, the callback attach_adapter() is called. Now is the
-time to determine what devices are present on the adapter, and to register
-a client for each of them.

-
-The attach_adapter callback is really easy: we just call the generic
-detection function. This function will scan the bus for us, using the
-information as defined in the lists explained above. If a device is
-detected at a specific address, another callback is called.
+being registered, your driver may be notified through one of two
+callbacks, depending on the degree of control you need to exercise over
+the probing process. This is the time to determine what devices are
+present on the adapter and to register a client for each device your
+driver supports.

+
+The easiest way to handle the probing process is to simply set the `class',
+`address_data', and `detect_client' fields in the i2c_driver structure.
+The `class' field is a bitmask of all the adapter classes which should be

Linux–Kernel: [PATCH 1/5] call i2c_probe from i2c core

+probed for devices supported by this driver. Typically you would just set
+this to I2C_CLASS_HWMON, which is appropriate for `sensors' drivers. The
+`address_data' field should be set to `&addr_data', which is defined by the
+macros explained above, so you do not have to define it yourself. When a
+new adapter is attached, the bus is scanned for the addresses defined in
+the lists above, and the detect_client callback gets called when a device
+is detected at a specific address.

+
+If you prefer, you can omit the `class', `address_data', and
+`detect_client' fields from your i2c_driver structure, and instead set
+`attach_adapter'. The `attach_adapter' callback gets called every time a
+new adapter is attached and the bus needs to be scanned, so if you need to
+perform any special checks or configuration before you scan a bus for
+devices, you should use attach_adapter. If the bus is suitable, you can
+then call the generic i2c_probe function to scan for the addresses in the
+lists explained above, and the callback passed in the third parameter will
+get called for each device detected.

```
int foo_attach_adapter(struct i2c_adapter *adapter)
{
    return i2c_probe(adapter,&addr_data,&foo_detect_client);
}
```

–Remember, structure `addr_data' is defined by the macros explained above,
–so you do not have to define it yourself.

–
–The i2c_probe function will call the foo_detect_client
–function only for those i2c addresses that actually have a device on
–them (unless a `force' parameter was used). In addition, addresses that
–are already in use (by some other registered client) are skipped.
+With either mechanism, addresses that are already in use (by some other
+registered client) are skipped.

The detect client function

–The detect client function is called by i2c_probe. The `kind' parameter
–contains –1 for a probed detection, 0 for a forced detection, or a positive
–number for a forced detection with a chip type forced.
+The detect client function is called by the address probing mechanism.
+The `kind' parameter contains –1 for a probed detection, 0 for a forced
+detection, or a positive number for a forced detection with a chip type
+forced.

Below, some things are only needed if this is a `sensors' driver. Those
parts are between `/* SENSORS ONLY START */` and `/* SENSORS ONLY END */`

–
To unsubscribe from this list: send the line "unsubscribe linux–kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo–info.html>

Linux-Kernel: [PATCH 1/5] call i2c_probe from i2c core

Please read the FAQ at <http://www.tux.org/lkml/>