I2C/SMBus Functionality

INTRODUCTION

Because not every I2C or SMBus adapter implements everything in the I2C specifications, a client can not trust that everything it needs is implemented when it is given the option to attach to an adapter: the client needs some way to check whether an adapter has the needed functionality.

FUNCTIONALITY CONSTANTS

For the most up-to-date list of functionality constants, please check <uapi/linux/i2c.h>!

I2C_FUNC_I2C	Plain i2c-level commands (Pure SMBus adapters typically can not do these)
I2C_FUNC_10BIT_ADDR	Handles the 10-bit address extensions
I2C_FUNC_PROTOCOL_MANGLING	Knows about the I2C_M_IGNORE_NAK, I2C_M_REV_DIR_ADDR and I2C_M_NO_RD_ACK flags (which modify the I2C protocol!)
I2C_FUNC_NOSTART	Can skip repeated start sequence
I2C_FUNC_SMBUS_QUICK	Handles the SMBus write_quick command
I2C_FUNC_SMBUS_READ_BYTE	Handles the SMBus read_byte command
I2C_FUNC_SMBUS_WRITE_BYTE	Handles the SMBus write_byte command
I2C_FUNC_SMBUS_READ_BYTE_DATA	Handles the SMBus read_byte_data command
I2C_FUNC_SMBUS_WRITE_BYTE_DATA	Handles the SMBus write_byte_data command
I2C_FUNC_SMBUS_READ_WORD_DATA	Handles the SMBus read_word_data command
I2C_FUNC_SMBUS_WRITE_WORD_DATA	Handles the SMBus write_byte_data command
I2C_FUNC_SMBUS_PROC_CALL	Handles the SMBus process_call command
I2C_FUNC_SMBUS_READ_BLOCK_DATA	Handles the SMBus read_block_data command
I2C_FUNC_SMBUS_WRITE_BLOCK_DATA	Handles the SMBus write_block_data command
I2C_FUNC_SMBUS_READ_I2C_BLOCK	Handles the SMBus read_i2c_block_data command
I2C_FUNC_SMBUS_WRITE_I2C_BLOCK	Handles the SMBus write_i2c_block_data command

A few combinations of the above flags are also defined for your convenience:

I2C_FUNC_SMBUS_BYTE	Handles the SMBus read_byte and write_byte commands
I2C_FUNC_SMBUS_BYTE_DATA	Handles the SMBus read_byte_data and write_byte_data commands
I2C_FUNC_SMBUS_WORD_DATA	Handles the SMBus read_word_data and write_word_data commands
I2C_FUNC_SMBUS_BLOCK_DATA	Handles the SMBus read_block_data and write_block_data commands
I2C_FUNC_SMBUS_I2C_BLOCK	Handles the SMBus read_i2c_block_data and write_i2c_block_data commands
I2C_FUNC_SMBUS_EMUL	Handles all SMBus commands that can be emulated by a real I2C adapter (using the transparent emulation layer)

In kernel versions prior to 3.5 I2C_FUNC_NOSTART was implemented as part of I2C_FUNC_PROTOCOL_MANGLING.

ADAPTER IMPLEMENTATION

When you write a new adapter driver, you will have to implement a function callback `functionality`. Typical implementations are given below.

A typical SMBus-only adapter would list all the SMBus transactions it supports. This example comes from the i2c-piix4 driver:

```
static u32 piix4_func(struct i2c_adapter *adapter)
{
      return I2C_FUNC_SMBUS_QUICK | I2C_FUNC_SMBUS_BYTE |
             I2C_FUNC_SMBUS_BYTE_DATA | I2C_FUNC_SMBUS_WORD_DATA |
             I2C_FUNC_SMBUS_BLOCK_DATA;
}
```

A typical full-I2C adapter would use the following (from the i2c-pxa driver):

```
static u32 i2c_pxa_functionality(struct i2c_adapter *adap)
{
      return I2C_FUNC_I2C | I2C_FUNC_SMBUS_EMUL;
}
```

I2C_FUNC_SMBUS_EMUL includes all the SMBus transactions (with the addition of I2C block transactions) which i2c-core can emulate using I2C_FUNC_I2C without any help from the adapter driver. The idea is to let the client drivers check for the support of SMBus functions without having to care whether the said functions are implemented in hardware by the adapter, or emulated in software by i2c-core on top of an I2C adapter.

CLIENT CHECKING

Before a client tries to attach to an adapter, or even do tests to check whether one of the devices it supports is present on an adapter, it should check whether the needed functionality is present. The typical way to do this is (from the lm75 driver):

```
static int lm75_detect(...)
{
      (...)
      if (!i2c_check_functionality(adapter, I2C_FUNC_SMBUS_BYTE_DATA |
                                   I2C_FUNC_SMBUS_WORD_DATA))
              goto exit;
      (...)
}
```

Here, the lm75 driver checks if the adapter can do both SMBus byte data and SMBus word data transactions. If not, then the driver won't work on this adapter and there's no point in going on. If the check above is successful, then the driver knows that it can call the following functions: i2c_smbus_read_byte_data(), i2c_smbus_write_byte_data(), i2c_smbus_read_word_data() and i2c_smbus_write_word_data(). As a rule of thumb, the functionality constants you test for with i2c_check_functionality() should match exactly the i2c_smbus_* functions which you driver is calling.

Note that the check above doesn't tell whether the functionalities are implemented in hardware by the underlying adapter or emulated in software by i2c-core. Client drivers don't have to care about this, as i2c-core will transparently implement SMBus transactions on top of I2C adapters.

CHECKING THROUGH /DEV

If you try to access an adapter from a userspace program, you will have to use the /dev interface. You will still have to check whether the functionality you need is supported, of course. This is done using the I2C_FUNCS ioctl. An example, adapted from the i2cdetect program, is below:

```
int file;
if (file = open("/dev/i2c-0", O_RDWR) < 0) {
      /* Some kind of error handling */
      exit(1);
}
if (ioctl(file, I2C_FUNCS, &funcs) < 0) {
      /* Some kind of error handling */
      exit(1);
}
if (!(funcs & I2C_FUNC_SMBUS_QUICK)) {
      /* Oops, the needed functionality (SMBus write_quick function) is
         not available! */
      exit(1);
}
/* Now it is safe to use the SMBus write_quick command */
```