EIB-Driver for Linux Kernel 2.4.

Georg Walkner
e9825265

April 8, 2000

Contents

1 EIB Linux Driver Module

1.1 Explanation

The Linux Driver Module was updated to the 2.4.x Kernel series. I have done the work on a 2.4.9. machine. The drivers program logic stays unchanged; many changes are only for newer styles and concepts introduced in the new Kernel.

1.2 Changes

names of the procedures init_module, cleanup_module

Used to give different modules different entry point names, maybe if the module is integrated into the kernel source. Anyway its a matter of style.
`#include <linux/init.h>`
`init_module` ⇒ `eib_init_module`
`cleanup_module` ⇒ `eib_cleanup_module`
At the end of the c-file:

```
module_init(eib_init_module)
module_exit(eib_cleanup_module)
```

Additional I have altered some variable names e.g. `io` ⇒ `eib_io`

wait queue definitions

```
#include <linux/wait.h>
156 struct wait_queue *eib_read_queue=NULL, *eib_write_queue=NULL; ⇒
214 DECLARE_WAIT_QUEUE_HEAD(eib_read_queue); /* wait queue */
215 DECLARE_WAIT_QUEUE_HEAD(eib_write_queue);
```

The calls `interruptible_sleep_on(&eib_read_queue);` and `wake_up_interruptible(&eib_read_queue)` stay the same.

definition of the file operation structure

```
119 static struct file_operations fops={
120 NULL,
121 eib_read,
122 eib_write,
123 NULL,
124 eib_poll,
125 eib_ioctl,
126 NULL,
127 eib_open,
128 eib_flush,
129 eib_release
130 };
```

changes to:

```
   150 struct file_operations eib_fops =
151 {
152 read:  eib_read,
153 write:  eib_write,
154 poll:  eib_poll,
155 ioctl:  eib_ioctl,
156 open:  eib_open,
157 flush:  eib_flush,
158 release:  eib_release,
159 };
```

unimplemented functions just miss in the structure.

Bottom half interrupt handler

Instead of a bottom half interrupt handler the 2.4 kernel introduces a tasklet object A bottom half handler would do the work either, but the tasklet object uses better internal mechanism especially on multiprocessor machines:
`229 DECLARE_TASKLET(temt_tasklet, do_temt_bh, 0);`
The second parameter specifies the tasklet procedure and the third parameter can be used to give the procedure one unsigned long value.
`96,492 void do_temt_bh(void);` ⇒
`117,839 void do_temt_bh(unsigned long unused);`

In the interrupt handler the call looks like: `tasklet_schedule(&temt_tasklet);`

interrupt disabling

The 2.2. version uses a construct to disable all interrupts in critical sections:

```
save_flags(flags); cli();
   critical section
restore_flags(flags);
```

In the 2.4. kernel its recommended to use spinlocks instead (on a single proccessor machine the following code behaves exactly like the old code).
Declaration:
`231 spinlock_t splock = SPIN_LOCK_UNLOCKED;`
protect critical sections:

```
spin_lock_irqsave(&splock, flags);

   critical section
spin_unlock_irqrestore(&splock, flags);
```

2 Java Testprogram Function

The Program manipulates and monitors to and from the EIB-Bus via the BCU1 Linux driver module. In normal operation when a button is pressed the BCU gets the Information and returns this message (Then its logged).

2.1 Running the Program

In the makefile the path of the rt.jar file has to be adjusted. The Program needs a source and a classes directory - In the classes directory there are the EIBLET Files already needed by the Program. The makefile prepares the java classpath for the Java-Tools (java, javac, jikes, ...). Therefore when executing make you have to be in the source directory.

2.2 Main Window

The Main Window contains fields to send Group Data to any address entered into the Group Address field. The Log field logs every message from the BCU1.

3 Java Testprogram Implementation

3.1 Class LinuxBCU1EIBBusCommunicator - Communication with the driver module

This class builds the layer for the low level communication with the driver. In the case of Linux this class uses the file `/dev/eib` (hardcorded in the source). It is the only class which accesses this file (the Interface to the kernel module). It implements the EIBLET-Interface `EIBLowLevelGroupObjectBusCommunicator`.

StatusEventListener

To get Status Events fired from the BusCommunicator every class (EIB) registers itself as StatusEventListener. This classes implements the `BusCommunicatorStatusEventListener`-Interface. A Status Event is for Example the disconnection of the BCU.

EventListener

To get messages from the BCU one class has to register itself as EventListener (differently to the StatusEvents theres only one). This class (EIBFrame) has to implement the `EIBLowLevelGroupObjectBusCommunicatorEventListener` Interface. If the BCU sends a Message to the driver Module the `LinuxBCU1EIBBusCommunicator` invokes the Procedure at the Eventlistener with one Parameter: an `EIBLowLevelGroupObjectBusCommunicato` which contains the whole Information of the Message (Address, Data, ...).

In my Example this Procedure is invoked on every instance which needs EIB-Data (`handleEvent(Event)`). (DemoFrame, LampButton, ...). So the information is passed to every EIB-Control Element.

3.2 EIBLampButton Class

One Button - when the Button is pressed - the message is sent; the buttons signals the yellow colour comes not until the commit message from the BCU comes back.

3.3 DimmerPanel Class

DimmerPanel contains four Buttons related to the set-up of the Demo Application in the Lab - One Dimmer contains two related Group Address Messages and two Messages to dim the light. (Actually it work either with only one switch).