

## Re: [GIT PATCH] more Driver core patches for 2.6.19

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-12/msg04171.html>

---

- *From:* Thomas Gleixner <[tglx@xxxxxxxxxxxxxx](mailto:tglx@xxxxxxxxxxxxxx)>
  - *Date:* Wed, 13 Dec 2006 22:36:21 +0100
- 

On Wed, 2006-12-13 at 12:58 -0800, Linus Torvalds wrote:

In other words, I'd like to see code that uses this that is actually `_better_` than an in-kernel driver in some way.

For USB, the user-mode thing made sense. You have tons of random devices, and the abstraction level is higher to begin with. Quite frankly, I simply don't even see the same being true for something like this.

We started to work on this for industrial I/O devices. Many of them are dual port memory based, others are dedicated chips for motion control or field busses.

The design requires to have an in kernel stub driver with interrupt handler which is capable to handle shared interrupts. User space `_cannot_` override an `irq_disable()`, it just has access to the chip registers of the device, which is possible right now as well.

The risk, that such a driver stalls the kernel is exactly the same as the risk you have with any other badly written driver.

This is a real world example of such a drivers interrupt handler:

```
/*
 * The chip specific portion of the interrupt handler. The framework code
 * takes care of userspace notification when we return IRQ_HANDLED
 */
static irqreturn_t sercos_handler(int irq, void *dev_id, struct pt_regs *reg)
{
    /* Check, if this interrupt is originated from the SERCOS chip */
    if (!(sercos_read(IRQ_STATUS) & SERCOS_INTERRUPT_MASK))
        return IRQ_NONE;

    /* Acknowledge the chip interrupts */
    sercos_write(IRQ_ACK1, SERCOS_INTERRUPT_ACK1);
    sercos_write(IRQ_ACK2, SERCOS_INTERRUPT_ACK2);
}
```

Re: [GIT PATCH] more Driver core patches for 2.6.19

```
return IRQ_HANDLED;  
}
```

With a full kernel driver we need:

1. Interrupt handler  
check interrupt  
acknowledge interrupt  
copy data from/to chip into a kernel buffer  
wakeup user space task
2. read data from driver, which goes through copy to user
3. do calculations
4. write data to driver, which goes through copy from user

After changing the driver concept we have only:

1. Interrupt handler  
check interrupt  
acknowledge interrupt  
wakeup user space task
2. User space task handles the mmaped chip directly

The change gave a serious performance gain in the range of 20% after the application was optimized for dealing with the chip directly.

There are tons of such exotic hardware devices out there, which now have either a closed source driver or an out of tree patch with an horrible amount of individual ioctl functions to get to the same point with less performance.

Btw: there's one driver we know we want to support in user space, and that's the X kind of direct-rendering thing. So if you can show that this driver infrastructure actually makes sense as a replacement for the DRI layer, then that would be a hell of a convincing argument.

I did not look closely into that, but I think that it is a valid usage candidate. The interface of graphic cards is user space mappable and it probably needs some interrupt handling + notification mechanism as well as the devices mentioned above.

tglx

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>

Re: [GIT PATCH] more Driver core patches for 2.6.19