

device driver race conditions problems (kernel 2.4)

Source: <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.system/2004-08/0271.html>

From: Paulo Garcia (paulo_at_digivoice.com.br)

Date: 08/27/04

Date: 27 Aug 2004 13:14:52 -0700

Hi,

I'm working in a device driver for kernel 2.4 to use with our new CTI E1/T1 board (using PCI bus with PLX9030 bridge).

This driver is already used with another boards but now I'm including some features to use with this PLX card. My firmware interrupts each 2ms to send and receive data.

I have an ISR (see below) that receives data from board, put them into a circular buffer (sbuffer) and a *read* function that reads this buffer and send data to usermode.

My driver must be able to manage more than one card using the same ISR and putting all signals into the same sbuffer. My approach is to use just one inode file (/dev/vbe1_s) to send data of all cards. To do that I'm reading card memory and inserting data into *sbuffer*.

My ISR calls *spin_lock_irqsave* to avoid one interrupt interrupts another one of another card.

The problem is when I put more than one card into PC, I have experienced some hangs. The driver is working well during some time. Analysing using an oscilloscope I can see that the problems occurs when one interrupt reaches another. The oscilloscope show me that one interrupt cannot interrupts the other but when one reaches another, my computer hangs. Sometimes, instead of hang my system turns unstable.

I think I'm experience some kind of race condition problem but I cannot see what happened.

Any help will be appreciated.

[]'s
Paulo Garcia

The code below is a fragment of my driver:

```
//----- isr -----
```

```

void plx_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
    spin_lock_irqsave(&irq_lock,flags);
    for (current_board=0;current_board<nCardsCount;current_board++)
    {
        if(it's my card?)
            break;
    }

    //check if it is commands from card
    if (num_cmds>0)
    {
        //add data to circular buffer
        sbuffer.signal[sbuffer.wp] = (u8)(buffer[loop_cmds]);
        inc_pointer_signal(&sbuffer.wp,1);
    }

    spin_unlock_irqrestore(&irq_lock,flags);

}
//increment function
static inline void inc_pointer_signal(volatile unsigned int *index,
int delta)
{
    unsigned long new_p = *index + delta;
    barrier (); /* Don t optimize these two together */
    *index = (new_p >= (SIGNAL_BUFFER_SIZE)) ? 0 : new_p;
}
//read function
vppci_read (struct file *filp, char *buf, size_t count,loff_t *pos) //
{
    unsigned int s_remaning,d_remaning,transfered;
    if (wait_event_interruptible(sbuffer.signal_queue, (sbuffer.rp!=
sbuffer.wp)))
    {
        xprintk("vppci_read: Interrupted by signal\n");
        return -ERESTARTSYS;
    }
    if (sbuffer.rp!= (sbuffer.wp)
    {
        //send data to user mode (/dev/vbe1_s)
        s_remaning = copy_to_user(buf,&sbuffer.signal[sbuffer.rp],1);
        d_remaning = copy_to_user(buf+1,&sbuffer.data[sbuffer.rp],1);
        ret = copy_to_user(buf+2,&sbuffer.port[sbuffer.rp],1);
        transfered = 1 - s_remaning;
        inc_pointer_signal(&sbuffer.rp, transfered);
        return 1;
    }
}
//----- end of code

```