

Re: Serial related oops

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-02/msg06890.html>

- *From:* Russell King <rmk+lkml@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 19 Feb 2007 23:20:20 +0000
-

On Mon, Feb 19, 2007 at 02:16:41PM -0800, Michael K. Edwards wrote:

Right. But as soon as you turn the source back on, in the postamble of the interrupt dispatch handler, it fires again. At least on ARM, that gives you recursive hits to `__irq_svc` and a couple of nested calls within it.

I think something else is going on here. I think you're getting an interrupt for the UART, and another interrupt is also pending.

When the UART interrupt is handled, it is masked at the interrupt controller, and the CPU mask is dropped.

The second interrupt comes in, and when you go to disable that source, you inadvertently re-enable the UART interrupt, despite it still being serviced.

This leads to the UART interrupt again triggering an IRQ.

Please show your interrupt controller (mask, unmask, mask_ack) handling functions corresponding with the interrupt which your UART is connected to.

But its context is not. Shared IRQ lines are a `_problem_`. You cannot safely enable an IRQ until all devices that share it have had their ISRs installed, unless you can absolutely guarantee at a hardware level that the uninitialized ones cannot assert the IRQ line.

Linux assumes that all interrupt sources on a shared IRQ line are disabled at the point in time when the kernel boots. When a device is to be used, an interrupt handler is installed and then the kernel will enable the interrupt on the device, not before.

Re: Serial related oops

Linux assumes incorrectly in this instance.
It would improve the kernel if all drivers' `__init` code were refactored into an IRQ-discovery-ISR-installation pass, followed by a chip-reset-data-structure-initialization pass, followed by a chip-configuration-driver-activation pass. This is unlikely to happen overnight.

This shows that you don't actually have an understanding of the Linux kernel boot, especially in respect of serial devices. At boot, devices are detected and initialised to a safe state, where they will not spuriously generate interrupts.

When a userspace program opens a serial port, which can only happen once the kernel boot has completed (ergo, devices have been initialised and placed in a safe state) the interrupts are claimed, and enabled at the source.

In the meantime, weird UART states on entry into `platform_device_init` are a reality.

Yes, uart states are indeterminate at this point. However, as soon as the 8250 driver loads it takes control of the 8250 ports, and **DISABLES** the interrupt on ALL ports found, **LONG BEFORE** any service handlers are installed.

So, by the time the system is up and running `_all_` 8250 ports have had their IERs written with zero. Interrupts disabled at source.

By the time you get to open any serial port, the initialisation has completed.

We follow that rule in the 8250 driver – in fact, when we initialise we ensure that interrupts are disabled on any devices we find.

No, you rely on the caller of `serial8250_init` to have punctured the abstraction

Can you add any other useless complex words into that sentence?

and forced any and all UARTs to a state where they cannot possibly generate an IRQ.

Re: Serial related oops

That is being done already at initialisation time.

Now, please show your interrupt mask/unmask/mask_ack code, which is where I believe your problem to lie.

--

Russell King

Linux kernel 2.6 ARM Linux – <http://www.arm.linux.org.uk/>
maintainer of:

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

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