Re: [PATCH] uio: User IRQ Mode

Source: http://linux.derkeiler.com/Mailing−Lists/Kernel/2008−07/msg01736.html

- *From*: "Hans J. Koch" <hjk@xxxxxxxxxxxxx>
- *Date*: Fri, 4 Jul 2008 15:26:33 +0200

On Fri, Jul 04, 2008 at 01:03:21PM +0900, Magnus Damm wrote:

> On Thu, Jul 3, 2008 at 9:45 PM, Hans J. Koch <hjk@xxxxxxxxxxxxx> wrote:
>
> > On Thu, Jul 03, 2008 at 09:10:19AM +0200, Uwe Kleine−K?nig wrote:
> >
> > > Hans J. Koch wrote:
> > >
> > > > On Wed, Jul 02, 2008 at 07:59:51PM +0900,
> > > > Magnus Damm wrote:
> > > >
> > > > > From: Uwe Kleine−K?nig
> > > > > <Uwe.Kleine−Koenig@xxxxxxxx>
> > > > >
> > > > > This patch adds a "User IRQ
> > > > > Mode" to UIO. In this mode
> > > > > the user space driver
> > > > > is responsible for
> > > > > acknowledging and
> > > > > re−enabling the interrupt.
> > > >
> > > > This can easily be done without your patch.
> > >
> > > BTW, the above wording "the user space driver is responsible for
> > > acknowledging and re−enabling the interrupt" is misleading. The kernel
> > > always has to acknowledge/disable/mask the interrupt. Userspace can only
> > > reenable it, ideally by writing to a chip register. In some cornercases
> > > for broken hardware we need the newly introduced write function.
>
> You seem to be mixing up masking/acknowledging the interrupt
> controller and masking/acknowledging the actual hardware device. In
> User IRQ Mode, the only thing the user space driver is accessing is
> the hardware device, with the exception of write() to re−enable
> interrupts which results in a enable_irq() that touches the interrupt
> controller.

> I'm not sure what kind of hardware devices you are talking about, but
> I have hardware devices here on my desk that need to be acknowledged
> by the device−specific driver to deassert the irq. And acknowledging
> from user space works just fine.

Yes, I know such devices. If you _need_ to do it this way, it's simply broken hardware design. You frequently find that in FPGAs designed by industry amateurs. It can easily be handled by UIO as it is.

If you do it without need, then it's simply broken software design because you use a kernel function which shows different behaviour on different archs and machines where a simple hardware register access would suffice.

> Shared interrupts are not
> supported by this mode.
>
> Signed−off−by: Uwe
> Kleine−K?nig
> <Uwe.Kleine−Koenig@xxxxxxxx>
> Signed−off−by: Magnus
> Damm
> <damm@xxxxxxxxxx>
> −−−
>
> Similar code has been
> posted some time ago as:
> "[PATCH] uio_pdrv:
> Unique IRQ Mode"
> "[PATCH 00/03][RFC]
> Reusable UIO Platform
> Driver".

> > Yes, and in that thread I gave detailed
> > explanations why I won't accept
> > that.
>
> I think for all of your concers one of the following is true:
>
> − they are not valid any more in this version; or

I question the whole concept as such. The concept of having a generic irq handler using disable_irq_nosync() makes no sense at all.

Reasons:

> − We do not introduce new possibilities. Everything can be done without that patch.

> Isn't it possible to use the same argument against UIO? There is no point in having it since it is possible to do it all in kernel space instead. But for various reasons people want to write their drivers in user space.

That's why we have UIO, mainly to allow clean interrupt handling.

> And for various reasons I'd like to acknowledge interrupts in user space. And I do that by disabling the interrupt in kernel space, notify the user space driver and acknowledge and re−enable the interrupt from there.

This is all possible with UIO as it is.

> The reason behind it is that I want to provide
> user space device driver people with consistent interface to the device without the need for any device specific code in kernel space.

And if it were a solution that was machine− and platform−independent and worked with all kinds of interrupts, there wouldn't be much objections.

BTW, you still need to touch kernel code to setup the structs for uio_pdrv, or did I misunderstand something? So where's the advantage?

> − We offer users an irq handler that shouldn't be used. It is seldom the best solution to call disable_irq_nosync() to disable an interrupt. In almost all cases you should use the irq mask register of the chip your driver handles. What do you want to write into the docs? Here's an irq handler, but please use it only if you're really desperate?

> What is wrong with using disable_irq_nosync() compared to using the device specific irq mask register? It works just fine, and it provides a nice abstraction in my mind.

It's the difference between one write to a chip register compared to a

call to an arch specific kernel function. The performance and latency issues might be negligible on some systems, but there's no guarantee. A register access is one write on all platforms.

> > − The only argument in favor of that concept was that it saves a few lines of irq handler code. Given the fact that all the handler has to do is toggle one bit in a hardware register, this is really not much. And you tempt people to delete 5 lines of good code and replace them with a sub−optimal generic irq handler.
>
> What is suboptimal about it? And since when is a device specific solution better than a generic one?

Drivers are always device specific. And disabling or masking an interrupt in the way the chip designer wanted you to do it is the fastest, most elegant, and most natural thing for a driver to do.

> > − You introduce the need for an irqcontrol function. This was not the intention of that concept. Normal UIO devices don't need a write function, this is only used for broken hardware. If you have normal hardware, and implement a proper 5 lines irq handler, userspace can simply reenable the irq by writing to a hardware register it has mapped anyway. In your concept, it has to use write() on /dev/uioX, which means you have to go all the way through libc, vfs, and the UIO core to finally call your irqcontrol function, which in turn calls enable_irq. As I said, there is broken hardware around where this is the only way, but most chips allow you to do it properly.
>
> You are the one who posted the irqcontrol code because it solved some issues you are having with broken hardware. I'm happy to use your code in a generic way. Actually, my first version of this patch avoided the extra write() call overhead by re−enabling things automatically, but since your irqcontrol patch is cleaner I'd rather use that and live with the small performance penalty.
>
> Performance in this case is really a non−issue.

Your talking about the board you've got in front of you. I'm talking about the UIO framework and the 20+ platforms it supports.

If we experience
performance problems in the future because of high interrupt overhead
then UIO is most likely no longer suitable for us. Right now it's a
perfect match.

That's your personal experience.

− I cannot understand it.

I'll try to list them all below. Please tell us if the list isn't
complete or if my comments doesn't convince you. You
might have to
repeat yourself, but for me it's hard to sort your arguments
because
Magnus' suggestion changed over time.

OK, the version Magnus sent last is different, so some of my arguments
are superfluous now.

Please, to make things simpler, let's only talk about the concept as
such and not go into implementation details. I deliberately do not review
that code (although I believe it has more bugs than the one Alan found),
because as long as the concept doesn't make sense, I don't care how it
is implemented.

I'd say it makes sense to Paul, Uwe and me.

Seems UIO gets tested intensively on SH at the moment :−)

Don't misunderstand me, even I can imagine cases where I _have_ to use
similar solutions. But that's no reason to patch it into the generic UIO
core. Because it should only be used if there's no other solution.
There's no point in inviting people to do it that way.

And please, I try to work out the pros and cons in a
constructive way
and hope there is nothing in it you will take personal. It's
really
that I consider the patch valuable and don't understand your
concerns.

> > You both keep telling me how valuable that patch is but never answered my question what the advantage would be. I cannot see it yet.
>
> Avoid having device specific interrupts handlers in kernel space?
>
> By replacing them with suboptimal generic handlers that work only in special cases?
>
> > > In the first thread[1] your unique and open concerns (to the best of my knowledge and belief) with my comments are:
> > >
> > > − "This only works for embedded devices [...]"
> > >
> > > OK, this doesn't work with shared IRQs which rules out x86. I don't claim to know all the 23[2] other architectures but IMHO if something is good for 3 archs and doesn't hurt the other 21, you should do it.
> > >
> > > − "This would save somebody the trouble to add the above 5 lines to the 30 lines of board/platform support code he has to write anyway. That's the only gain, and that is not enough."
> > >
> > > IMHO it's worth it. Because if you add the five lines to a central place you save 5 lines per platform using the driver.
> >
> > OK, that is one argument in favor. I always admitted that, but said that this is not enough to compensate for the disadvantages mentioned above.
>
> I still fail to see your technical arguments.
>
> > > Moreover this might prevent some bugs. (And obviously this function has the potential to have a buggy implementation as the comment by Alan Cox shows.)
> >
> > For me, this shows two things:
> >
> > − I never ever had to use disable_irq_nosync() in any UIO driver yet, otherwise I would have noticed.

> Is that so? Given the stunning UIO patch throughput it wouldn't surprise me if people end up using local patches instead of even trying to merge things upstream. Which means you wouldn't see the code.

> > − Magnus turned in a patch that he never tested.

> Is that so?
>
> I've tested the patch using two different processors on three different boards. I have written a user space uio driver that interfaces to mplayer using vidix, providing hardware accelerated color space conversion and stretching. So don't call my patch untested please.

OK, I apologize for that statement.

> I plan on submitting the vidix driver upstream soon after the uio interface gets stabilized, ie this patch gets merged.

I'm sure you get your vidix driver to work without this patch.

> > > − "And if you _know_ that on your platform the irq is not shared, this might really be a one−liner that simply calls irq_disable. That's OK in board specific code, but not in a generic driver."
> > >
> > > Please note that the patch only introduces a helper that the platform code *can* use. You still have the freedom not to use it without any overhead.

> > It's not a good idea to add nonsense code and tell the users to ignore it whenever they can...

> This nonsense code, exactly which code are you talking about?

The concept of having a generic irq handler that works only with

non−shared interrupts.

Could it
be one of your significant kernel contributions? =)

You think I have some? Thank you ;−)

− "I won't accept anything that changes the current UIO
behaviour."

Not valid anymore. There is no change in behaviour.

Well, at least the whole stuff would have to be explained in the docs.

Sure, documentation is always good.

You add an element to struct uio_info, together with a new #define. And
a whole class of drivers using that stuff would have a write() function
without needing it.

Another way to see it is that a whole class of drivers would need
kernel space interrupt handling code without really needing it.

In the second thread[3] I cannot find any open concerns that
are not
already listed above.

Changes since Uwe's last
version:
− flags should be unsigned
long
− simplify
uio_userirq_handler()

That's nearly nothing. All you do is sending
the same stuff three weeks
later in the hope somebody will merge it this

time. NAK.

I think nobody really is surprised that you're not happy with the new
post. But note that Magnus just did what you told him. ("I'm [not] the
big boss who makes the final decision. I can be critized and overridden.
If Greg loves your patch and merges it, fine. Try it.")

In the hope not to have kicked off a flame,

Oh, no, stay cool ;−)

Good, let's stay cool.

Thanks,
Hans

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