

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-07/msg01950.html>

- *From:* "Magnus Damm" <magnus.damm@xxxxxxxxxx>
 - *Date:* Sat, 5 Jul 2008 07:51:27 +0900
-

On Fri, Jul 4, 2008 at 10:26 PM, Hans J. Koch <hjk@xxxxxxxxxxxxxxxx> wrote:

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@xxxxxxxxxxxxxxxx> wrote:

On Thu, Jul 03, 2008 at 09:10:19AM +0200, Uwe Kleine-König wrote:

Hans J. Koch wrote:

On Wed, Jul 02, 2008 at 07:59:51PM +0900, Magnus Damm wrote:

From: Uwe Kleine-König
<Uwe.Kleine-Koenig@xxxxxxxxxx>

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

Re: [PATCH] uio: User IRQ Mode

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.

This is crap. I'm sure you're having all sorts of problems with broken FPGAs, but that's outside the scope of this discussion. If a device needs to be acknowledged or not to deassert the interrupt line is totally device specific and it has nothing to do with broken hardware design.

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.

As much as I enjoy arguing with you, we are doing this because there is need for it. =)

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

Shared
interrupts
are not
supported
by this
mode.

Signed-off-by:
Uwe
Kleine-König
<Uwe.Kleine-Koenig@xxxxxxxx>
Signed-off-by:
Magnus
Damm
<damm@xxxxxxxx>

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 concerns 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:

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

– 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.

Yes, and that's fine.

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.

It's not a matter if it's possible or not. It's a matter of abstraction.

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.

Exactly what do you mean by "so much objections"? You're the only one blocking this, and it looks to me like you're just pushing your own policy without thinking outside your little device driver box.

I'm not doing this because i want to change how UIO is working in general or push some generic solution and force everyone to use it. I only want to export our devices. If other people want to export their devices the same way, then that's great. Uwe seems to want to do that. Why do you feel you have to stop us? It's pretty natural for you to have no insight in we are doing on the SuperH side, and because of that you should just take our word for what we need and work with us to resolve technical issues.

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

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?

The advantage is that we don't need to be aware of device-specific details in the kernel.

You seem to be happy with the usual UIO way of splitting the driver code in a kernel stub and a user space driver. Good for you, but we want to skip the device specific kernel stub.

For our needs – on-chip memory mapped hardware blocks – we'd like to replace the kernel stub with a single reusable implementation, and that way we end up with a single per-hardware block specific driver – in user space. This way we can let people with RTOS experience do their thing in user space, without worrying about them overwriting our precious kernel data structures.

You are right that we still need to feed the platform driver information about the device. We only want to pass the base address of the hardware block and the interrupt number. There is no point for us to be aware about the inner workings of the hardware block – we only want to export the thing to user space and be done with it. And in some rare cases we don't even have documentation for a certain block, so acknowledging interrupts from inside the kernel is out of the question.

– 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.

Re: [PATCH] uiio: User IRQ Mode

Wrong. It's not always that easy. The way to acknowledge interrupts is very device specific. A good in-kernel example of not so easy acknowledging would be `drivers/input/touchscreen/ads7846.c`. Look at the interrupt handler code `ads7846_irq()` – what does it do? Exactly what the User IRQ Mode is doing – using `disable_irq()`. The reason for this is slow SPI bus speed. It takes a long time to write to the touch screen controller over the serial bus, and the interrupt line will remain asserted until the device has been acknowledged properly.

– 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.

Again, exactly how to acknowledge an interrupt is device or hardware block specific. And it's not always as easy as writing to a single register. Of course we will acknowledge the device in the way the chip designer want us to, but we will do it from user space.

– 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 reenale the irq by writing to a hardware register it has mapped anyway. In your concept, it has to use `write()` on `/dev/uiioX`, which means you have to go all the way through `libc`, `vfs`, and the UIO core

Re: [PATCH] uio: User IRQ Mode

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.

I'm not forcing you to use my code on your platforms. Feel free to use it if it does what you want to do, if not you should just keep on using your regular code. How can this be my problem? Don't NAK just because the code isn't suitable for all your needs. I'm not trying to solve your needs. I'm posting this code because `_we_` need it.

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 :-)

No, you are wrong. It really doesn't get tested. Mainly because you are refusing to accept my code. UIO `_would_` be used and tested if we can agree on these 50 lines of code and get on with our lives.

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

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

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.

You still haven't given any real technical reason why it's better to acknowledge in kernel space over acknowledging in user space. Having all hardware block / device specific code in a single place will keep interfaces clean and make maintenance easier for us.

SuperH hardware just happens to use unique interrupt lines for most internal hardware blocks, and we'd like to take advantage of that and skip the entire UIO kernel stub thing.

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?

There is nothing suboptimal about it.

– 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.

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

Thank you.

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.

It is and has been working for more than a month. It's not suitable for submission yet though, mainly since the UIO kernel interface is unstable until we've agreed on this patch. The kernel interface changed with the write() re-enabling method, remember? And I feel this may happen again since it seems very hard for us to agree on things.

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.

That may look like nonsense to you, but you fail to see the big picture if so.

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

You think I have some? Thank you ;-)

No, I don't think so. But I do think you should work a bit on understanding your position in the community and pay closer attention to people with more experience than you. I may understand that you feel the need to argue with me – since I only have a few small contributions behind me – but when you're not listening to what Paul has to say then you're just stupid. Stop ignoring people above you in the food chain.

You can use git shortlog to get statistics.

/ magnus

Re: [PATCH] uio: User IRQ Mode

Re: [PATCH] uio: User IRQ Mode

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

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

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