DMA safety in buffers for Linux Kernel device drivers

Wolfram Sang, Consultant / Renesas

23.10.2018, ELCE2018

I2C & DMA

usual use case

- lots of messages
- small payloads

DMA was not considered when I2C went into Linux

That was unfortunate

If you don't have a clear rule, things will go in all directions[1]

[1] including wrong ones

The reality

Buffers come from

- the heap
- the stack
- rodata
- kmapped memory
- ...

Clear rule was made

DMA is optional for I2C!

- to avoid regressions
- to not convert zillions of drivers which we don't have the HW for
- to honor most use cases (small payloads)

make it known

- document it

  see `Documentation/i2c/DMA-considerations` for details
- mention it in reviews
- speak at conferences about it

Now, how to differentiate?

Ideally, at runtime!

...but why is there no `is_dma_capable()`?

What do we have

- is_vmalloc_addr()

What do we have

- is_vmalloc_addr()
- is_vmalloc_or_module_addr()

What do we have

- is_vmalloc_addr()
- is_vmalloc_or_module_addr()
- virt_addr_valid()

[2] gkh: `https://www.spinics.net/lists/linux-usb/msg156359.html`

What do we have

- is_vmalloc_addr()
- is_vmalloc_or_module_addr()
- virt_addr_valid()
- object_is_on_stack()

  I didn't know we had that macro...[2]

[2] gkh: `https://www.spinics.net/lists/linux-usb/msg156359.html`

What do we have

- is_vmalloc_addr()
- is_vmalloc_or_module_addr()
- virt_addr_valid()
- object_is_on_stack()

  I didn't know we had that macro...[2]

- ___cacheline_aligned

[2] gkh: `https://www.spinics.net/lists/linux-usb/msg156359.html`

kthxbye

This is not going to fly

- there is a reason `lib/dma-debug.c` is ~~>45~~ 46 kB in size
- switch to a manual opt-in approach
- `I2C_M_DMA_SAFE` now marks a DMA-safe message buffer

new I2C API calls - master side

`i2c_get_dma_safe_msg_buf()`

- will return a buffer guaranteed to be DMA-safe
- either the original message buffer or a bounce buffer
- simple 1:1 allocation, no pool of buffers

~~`i2c_release_dma_safe_msg_buf()`~~ `i2c_put_dma_safe_msg_buf()`

- clean up the buffer from above

All optional, if your master driver can do more advanced, you can work directly with `I2C_M_DMA_SAFE`.

new I2C API calls - client side

`i2c_master_recv_dmasafe()`

- basically `i2c_master_recv()` with the new flag set

`i2c_master_send_dmasafe()`

- likewise

- messages from userspace (`i2c-dev`) are always copied to DMA-safe buffers and marked accordingly
- same goes for emulated SMBus block data transfers

What about regmap?

Regmap unifies access to device, abstracting away busses like I2C, SPI.

How does it handle its buffers?

We pretty much assume everything is DMA safe already, [...] but for bulk transfers we use the caller buffer and there might be some problem users. I can't really think of a particularly good way of handling it off the top of my head, [...] Doing _dmasafe() isn't particularly appealing either but might be what we end up with.[3]

[3] `https://lkml.org/lkml/2017/11/8/1021`

I certainly agree to that

Mark Brown on the current I2C solution...

It's hard to summon enthusiasm but yes, without changes to the DMA stuff it's probably as good as we can do.[4]

...and what we would like to have

It would really help a lot of things if there were a way to detect if a given memory block is DMA safe, having to pass the information around causes so much pain.[5]

[4] `https://lkml.org/lkml/2017/11/28/1061`

[5] `https://lkml.org/lkml/2017/11/8/1021`

How are other subsystems doing?

Still some in the same situation I2C used to be in

- SLIMBus
- SPMI
- OneWire

All have buffers, DMA is not mentioned anywhere, but are handled by regmap as well.

SPI

- DMA required
- documented (from `Documentation/spi/spi-summary`):

  "Follow standard kernel rules, and provide DMA-safe buffers in your messages."

- even has helper functions for DMA mapping/unmapping

Want some fun?

- read this thread[6] about doing DMA correctly with an SPI-NOR flash (MTD subsys) carrying some filesystem (block layer)
- or read here[7] for proper cache flushing in the same setup

[6] `https://patchwork.kernel.org/patch/10131845/`
[7] `https://patchwork.kernel.org/patch/9579553/`

It is subtle!

Even with clear rules, users can get it wrong:

- buffers embedded in structs[8]
- `kmalloc()` vs. `devm_kmalloc()`[9]
- buffers on stack work on their platforms

  unless someone activates `CONFIG_VMAP_STACK`
- (rules are overlooked/ignored)

[8] still in for fun? `https://patchwork.kernel.org/patch/9965809/`

[9] read here: `http://linux-arm-kernel.infradead.narkive.com/vyJqy0RQ/question-devm-kmalloc-for-dma`

Conclusions

- We have DMA transfers in the kernel which work on assumptions which may not be true for everyone or not for the future, sometimes at subsystem level.

Conclusions

- There is no easy way to detect that at runtime; the best we have is a debug option not probably not suitable for most production cases.

Conclusions

- A proper generic solution might be annotated buffers (yet I am not aware they are on the horizon).

Conclusions

- Underlying problems can be hard and subtle. Sometimes they are known but long-standing beacuse of this.

Conclusions (Subsystem maintainers)

- make clear rules as soon as possible
- keep an eye on this during review

Conclusions (Developers)

- always develop with CONFIG_DMA_DEBUG
- fix all issues it reports, even if DMA is working for you
- audit code you are touching anyhow for DMA safety

Conclusions (everyone)

- if you care about safety, pay attention to DMA
- spread the word, document status-quo
- collaborate

The End

Thanks for listening!

Questions?

- Right here, right now...
- At the conference
- `wsa@the-dreams.de`

And thanks again to Renesas for funding this work!