Subsystems with object lifetime issues
(in the embedded case)

Wolfram Sang, Consultant / Renesas

29.06.2023, EOSS 2023
1 Basics

2 The actual problem

3 Results

4 Conclusion
Basics
In computer science, reference counting is a programming technique of storing the number of references, pointers, or handles to a resource, such as an object, a block of memory, disk space, and others. ...

The main advantage of the reference counting ... is that objects are reclaimed as soon as they can no longer be referenced...

Done in the Kernel via

- struct kref\(^1\)

\(^1\)described in [Documentation/core-api/kref.rst](https://www.kernel.org/doc/Documentation/core-api/kref.rst)
Now to the Linux driver model

**struct device** is the research element here!

- embeds **struct kobject**\(^2\) which embeds **struct kref**
- refcount is modified with **get_device** and **put_device**
- a release callback is used when refcount is 0

\(^2\)described in [Documentation/core-api/kobject.rst](https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/core-api/kobject.rst)
Physical vs. logical device - simplified

Character device: /dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242
Physical vs. logical device - more realistic

Character device:
/dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter

Logical device
The actual problem
Good case - initialization during boot

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter
ref count = 1
Good case - userspace open()

Character device:
/dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter
ref count = 2
Good case - userspace close()

- Character device: /dev/i2c-2
- Platform device:
  Mem = 0xe60b2000
  IRQ = 242
- Intermediate device:
  struct i2c_adapter
  ref count = 1

Wolfram Sang, Consultant / Renesas
Good case - unbind/poweroff

Character device:
/dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter
ref count = 0
Problematic case 1 - not the main case here

Character device: /dev/i2c-2

Intermediate device: struct i2c_adapter
    ref count = 1

Platform device:
    Mem = 0xe60b2000
    IRQ = 242
struct rcar_i2c_priv {
    ...
    /* contains a struct device */
    struct i2c_adapter adap;
    ...
}

priv = devm_kzalloc(&pdev->dev, sizeof(*priv), GFP_KERNEL);
    ...

ret = i2c_add_adapter(&priv->adap);
Problematic case 2 - this is the main case

Character device:
/dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter
ref count = 1

Needs protection from the subsystem!
especially with devm_kzalloc (but this makes problems detectable)
My research questions

Which subsystems use structs with embedded kobjects and allow allocation with `devm_kzalloc`?

- coccinelle to the rescue!
- only scanned for `struct device` instead of all kobjects to limit the search space
- found ~630 structs embedding a `struct device`, either directly or indirectly
- each struct was then scanned for allocation with `devm_kzalloc`

And how does it protect against a too early release of the embedded kobject?

- needs manual inspection

\(^3\)like I2C uses `i2c_adapter` embedding a `struct device`
Results
I have good news and bad news

Good news
- less subsystems than I guessed are affected

Bad news
- every subsystem that was found I have issues with
I have good news and bad news

**Good news**
- less subsystems than I guessed are affected

**Bad news**
- every subsystem that was found I have issues with
Live Demo: MTD
struct spi_nor embeds struct mtd_info which embeds struct device
- crashes when unbinding while mtdXro is opened
- no protection mechanism found
- but this comment in mtdcore.c from March 2009

/* REVISIT once MTD uses the driver model better, whoever allocates
 * the mtd_info will probably want to use the release() hook...*/
struct i2c_adapter embeds struct device
blocks uninterruptible when unbinding while i2c-X is opened
i2c_del_adapter waits until all references are gone using a struct completion with the release function of the logical device
also has a comment from January 2015

/* wait until all references to the device are gone
 * FIXME: This is old code and should ideally be replaced by an
 * alternative which results in decoupling the lifetime of the struct
 * device from the i2c_adapter, like spi or netdev do. Any solution
 * should be thoroughly tested with DEBUG_KOBJECT_RELEASE enabled!
 */
How does this completion work?

Character device:
/dev/i2c-2

Platform device:
Mem = 0xe60b2000
IRQ = 242

Intermediate device:
struct i2c_adapter

1. in remove, call i2c_del_adapter
2. in i2c_del_adapter, wait_for_completion
6. waiting ended, everything can be freed now

3. After close, refcount goes 0

4. refcount is 0, call release
5. in release, call complete
struct i3c_master_controller embeds struct device
can’t test because no HW
there may be some protection, not fully understood yet
but for backwards compatibility, it also embeds struct i2c_adapter
and uses i2c_del_adapter to remove it
so it also uses a completion when unbinding

Sorry!
- struct ntb_dev embeds struct device
- can’t test because no HW
- review reveals it also uses a completion when unbinding
- virtio_device embeds struct device
- forces all drivers to use the release callback
- one driver gets it wrong with devres
- fails when enabling CONFIG_DEBUG_KOBJECT_RELEASE
How does this release callback work?

Platform device:
Mem = 0xe6800000
IRQ = 300

1. Probe: allocate private struct including a virtio_device with kzalloc
2. Probe: populate virtio_device.release with a callback calling kfree
   (something is done with these devices)
3. Unbind: delete (but not release) virtio_device, refcount goes 0
4. or 5. Unbind: remove all resources attached to the platform_device
   except private struct including the virtio_device
4. or 5. call release, private struct
Why did this one driver fail?

Platform device:
Mem = 0xe6800000
IRQ = 300

1. Probe: allocate private struct including a virtio_device with devm_kzalloc
2. Probe: populate virtio_device.release with a callback calling devm_kfree
   (something is done with these devices)
3. Unbind: delete (but not release) virtio_device, refcount goes 0

   6. Unbind: remove all resources attached to the platform_device
      including private struct including the virtio_device
      because of devres

4. refcount is 0, release will be called
5. actually calling release will be delayed by
   CONFIG_DEBUG_KOBJECT_RELEASE

7. try to call release but the device and its
   release function are gone
8. OOPS
I don’t like this approach

It was tried multiple times to get it right:

edfd52e63672 ("virtio: Add platform bus driver for memory mapped virtio device")
cecdadb371e ("virtio_mmio: Set dev.release() to avoid warning")
7eb78b1bb7 ("virtio_mmio: add cleanup for virtio_mmio_probe")
c2e90800aef2 ("virtio_mmio: fix devm cleanup")

And it was still not correct until a few hours minutes ago\(^4\).

Don’t let drivers handle the lifetime of objects outside their scope!

Bartosz formulated the same opinion in this talk.

\(^4\) The fix is here
also expects drivers to populate the release callback
at least extensively documented
still pmic_glink.c got it likely wrong (can’t test no HW)
but the other drivers do it right

Or? Let’s check this driver.
Sketch of a better API?

gp_aux_data->region_start = pci_resource_start(pdev, 0);
gp_aux_data->region_end = pci_resource_end(pdev, 0);

aux_dev = auxiliary_device_alloc(&pdev->dev, gp_aux_data);
aux_dev->name = aux_dev_otp_e2p_name;
aux_dev->id = ida_alloc(&gp_client_ida, GFP_KERNEL);

/* And if you still do something special */
aux_dev->release = my_release_func;

aux_dev_register(aux_bus, aux_dev);
\begin{itemize}
\item \texttt{struct usb_gadget} embeds \texttt{struct device}
\item also expects drivers to populate the release callback
\item on unbind, it blocks like a completion but because of \texttt{cancel_work_sync}
\item combinations of \texttt{devm_kzalloc} and release callback exist
\item I was not able to trigger a problem so far, though
\item USB is usually good, but I will investigate more
\end{itemize}
Conclusion
we do have different kinds of life cycle problems in the Kernel

including really long lasting ones

devm_kzalloc is not directly responsible for the ones researched here

devm_kzalloc makes it here easier to fall into the trap
Further research activities for this type of problem

- repeat the above search but for `kzalloc` with accompanying `kfree` in `remove` of the physical device (instead of the `release` callback in the logical device)

- scan for embedded `struct kobject` not only `struct device`

- scan for empty `release` functions

- ...
Potential solutions

- There seems to be an agreement that pushing responsibility to drivers is not the way to go

- garbage collector??
  Nobody likes it

- \_\_cleanup\_\_ + kref as suggested by Bartosz?
  Paradigm shift, probably looong way to go

- Layers using devices should keep responsibility for them.
  Convert faulty subsystems to *_alloc and *_register
  My favourite solution because it also creates consistency among subsystems.

In any case, this is a **lot** of work!
Reminder: more lifecycle issues exist

- character devices interaction with platform devices
- dependency hell between devices
  I heard V4L2 knows about this...
- existing protections work mostly but still have race conditions
  At least with character devices, but at least a prototype solution exists
-...

Wolfram Sang, Consultant / Renesas
Our safety process for lifecycle issues in drivers

If an issue regarding lifecycles of objects in Linux device drivers is discovered, the process is:

- add it to the list of already known lifecycle problems.

It is agreed that fixing these issues would be great. But nobody does it because there be dragons and it simply is a lot of annoying work.
Our safety process for lifecycle issues in drivers

If an issue regarding lifecycles of objects in Linux device drivers is discovered, the process is:
Our safety process for lifecycle issues in drivers

If an issue regarding lifecycles of objects in Linux device drivers is discovered, the process is:

- **add it to the list of already known lifecycle problems.**

It is agreed that fixing these issues would be great. But nobody does it because there be dragons and it simply is a lot of annoying work.
Frankly? Not bright.

I think these problems are not “raising spirits”, so I guess we see some of the FIXME comments above still in 10 years.

First step is to raise interest, so funding will be available to work on lifecycle issues consistently. I’ll try.

Until then, I will slowly start fixing the I2C subsystem, at least.

At least, I fixed VIRTIO for this talk :)}
Thank you for supporting this!
Thank you for coccinelle!
Questions? Comments?

- Right here, right now...
- At the conference
- wsan@kernel.org