

# Re: [PATCH RFD] alternative kobject release wait mechanism

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg06406.html>

---

- *From:* Tejun Heo <[htejun@xxxxxxxx](mailto:htejun@xxxxxxxx)>
  - *Date:* Thu, 19 Apr 2007 00:34:26 +0900
- 

Hello,

Alan Stern wrote:

On Wed, 18 Apr 2007, Tejun Heo wrote:

Hello, all.

Agreed with the problem but I'm not very enthusiastic for adding `kobj->owner`. How about the following? `exit()` routines will have to do `device_unregister_wait()` instead of `device_unregister()`. On return from it, it's guaranteed that all references to it are dropped and `->release` is finished. The caller is responsible for avoiding deadlock, of course.

There's a problem with this approach.

Many drivers, especially those for hot-pluggable buses, register and unregister devices dynamically. These events can occur in time-critical situations, where the driver cannot afford to wait for all the references to be dropped when unregistering a device. It's okay to wait in a module exit routine, but to make things work the routine would have to wait for references to `_all_` unregistered objects to go away, not just the references for the objects it unregisters at exit time.

So let's see what changes are needed to make the approach workable. We will have to maintain a count of objects whose release methods haven't been called yet. The count has to be incremented every time an object is unregistered (or registered, it doesn't matter which) and decremented `_after_` the release method returns — meaning somewhere in the driver core. When the count goes to zero, the exit routine is then allowed to terminate.

Hmmm, this is beginning to sound like a module-wide refcount which serves to block `mod->exit()`. In fact, it sounds almost identical to what Cornelia wrote, except that the refcount refers only to devices rather

## Re: [PATCH RFD] alternative kobject release wait mechanism

than arbitrary kobjects (and except that the blockage just before mod->exit returns instead of just after). You can see where I'm leading...

The goal of immediate-disconnect is to remove such lingering reference counts so that device\_unregister() or driver detach puts the last reference count.

You tell a higher layer that a device is going away, on return from the function, that layer isn't gonna access the device anymore. ie. On return from the unregistration function, the upper layer shouldn't have any reference to the device. If you unregister from all layers a device is registered to, you are left with only 1 reference which you put with device\_unregister(). After all are converted, reference count doesn't mean much. struct device isn't a reference counted object anymore.

I don't think this is gonna be too difficult to do. I think I can convert block layer and IDE/SCSI drivers without too much problem. Dunno much about other layers tho.

Incidentally, Tejun, I'm all in favor of a immediate-detach driver model approach. Unfortunately it's impossible to realize fully, although we could come much closer than we are now.

Here's an example where immediate-detach cannot be implemented. A driver binds to a device and uses that device as a kernel thread. The thread carries out certain operations which require it to hold the device semaphore (because, for example, they need to be mutually exclusive with unbind).

The driver's remove() method is called with the semaphore held. If the thread tries to lock the semaphore at the same time and blocks, there is no way at all for the remove() method to force the thread to drop its reference.

This isn't merely a theoretical example. The USB hub driver works in exactly this way.

Dunno if I understood the problem right but can't we do the following?

```
remove()
{
  acquire sem;
  device_del();
  release sem;
  device_put_wait();
}
```

Re: [PATCH RFD] alternative kobject release wait mechanism

—  
tejun

—

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

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

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