

Re: [PATCH RFD] alternative kobject release wait mechanism

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg06458.html>

- *From:* Alan Stern <stern@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 18 Apr 2007 15:07:28 -0400 (EDT)
-

On Thu, 19 Apr 2007, Tejun Heo wrote:

More afterthoughts. If a mutex is used to protect access against removal. There is no reason to hold reference to it.

```
kernel_thread()
{
/* wanna dereference my_obj */
mutex_lock();
verify my_obj is there and use it if so.
mutex_unlock();
}
```

```
remove()
{
mutex_lock();
kill_it();
mutex_unlock();
}
```

I probably have over simplified it but using both mutex and reference counts doesn't make much sense. IOW, you get an active reference when you grab the mutex excluding its removal and verified it's still there.

There probably are other reasons why things are done that way and we can and probably will have to resort to mixed solutions in foreseeable future but I don't think there is any inherent problem in applying immediate-disconnect in the described situation.

Feel free to scream at me if I'm getting it totally wrong. :-)

This doesn't solve a related problem: a subsystem wants to register devices and to provide a set of mutually-exclusive services to the devices' drivers. The mutual exclusion has to be provided by a mutex or something similar, and the drivers need a way to unbind even while waiting to acquire the mutex.

Re: [PATCH RFD] alternative kobject release wait mechanism

The obvious answer is to introduce a different sort of synchronization primitive: a mutex (or semaphore or rwsem) which can be invalidated.

The semantics would be straightforward. When `mutex_invalidate()` is called, it marks the mutex so that all future lock attempts will fail with `-ENODEV`. It also wakes up all threads that are blocked trying to lock the mutex and causes them to fail with the same error. Once all that is done `mutex_invalidate()` returns. In particular, it doesn't wait for the current lock to be released — in fact, you would call it while holding the lock.

This would solve a lot of your problems. But it would also mean making extensive changes to the kernel. For one thing, `mutex_lock()` would return `int` instead of `void`, and you would want to mark it `__must_check`. Every place where a mutex is locked, the code would have to be changed to add an error pathway. That's the sort of thing I was talking about when I said it was going to be a tremendous job.

I thought of something else that could also be done: There should be a way to cancel an outstanding workqueue request. At the moment all