Lockdep-RCU Splat

Lockdep-RCU was added to the Linux kernel in early 2010 (http://lwn.net/Articles/371986/). This facility checks for some common misuses of the RCU API, most notably using one of the rcu_dereference() family to access an RCU-protected pointer without the proper protection. When such misuse is detected, an lockdep-RCU splat is emitted.

The usual cause of a lockdep-RCU slat is someone accessing an RCU-protected data structure without either (1) being in the right kind of RCU read-side critical section or (2) holding the right update-side lock. This problem can therefore be serious: it might result in random memory overwriting or worse. There can of course be false positives, this being the real world and all that.

So let's look at an example RCU lockdep splat from 3.0-rc5, one that has long since been fixed:

```
===============================
WARNING: suspicious RCU usage
-------------------------------
block/cfq-iosched.c:2776 suspicious rcu_dereference_protected() usage!
```

other info that might help us debug this:

```
rcu_scheduler_active = 1, debug_locks = 0
3 locks held by scsi_scan_6/1552:
#0:  (&shost->scan_mutex){+.+.}, at: [<ffffffff8145efca>]
scsi_scan_host_selected+0x5a/0x150
#1:  (&eq->sysfs_lock){+.+.}, at: [<ffffffff812a5032>]
elevator_exit+0x22/0x60
#2:  (&(&q->__queue_lock)->rlock){-.-.}, at: [<ffffffff812b6233>]
cfq_exit_queue+0x43/0x190

stack backtrace:
Pid: 1552, comm: scsi_scan_6 Not tainted 3.0.0-rc5 #17
Call Trace:
[<ffffffff810abb9b>] lockdep_rcu_dereference+0xbb/0xc0
[<ffffffff812b6139>] __cfq_exit_single_io_context+0xe9/0x120
[<ffffffff812b626c>] cfq_exit_queue+0x7c/0x190
[<ffffffff812a5046>] elevator_exit+0x36/0x60
[<ffffffff812a802a>] blk_cleanup_queue+0x4a/0x60
[<ffffffff8145cc09>] scsi_free_queue+0x9/0x10
[<ffffffff81460944>] __scsi_remove_device+0x84/0xd0
[<ffffffff8145dca3>] scsi_probe_and_add_lun+0x353/0xb10
[<ffffffff817da069>] ? error_exit+0x29/0xb0
[<ffffffff817d98ed>] ? _raw_spin_unlock_irqrestore+0x3d/0x80
[<ffffffff8145e722>] __scsi_scan_target+0x112/0x680
[<ffffffff812c690d>] ? trace_hardirqs_off_thunk+0x3a/0x3c
[<ffffffff817da069>] ? error_exit+0x29/0xb0
[<ffffffff812bcc60>] ? kobject_del+0x40/0x40
[<ffffffff8145ed16>] scsi_scan_channel+0x86/0xb0
[<ffffffff8145f0b0>] scsi_scan_host_selected+0x140/0x150
[<ffffffff8145f149>] do_scsi_scan_host+0x89/0x90
[<ffffffff8145f170>] do_scan_async+0x20/0x160
[<ffffffff8145f150>] ? do_scsi_scan_host+0x90/0x90
[<ffffffff810975b6>] kthread+0xa6/0xb0
[<ffffffff817db154>] kernel_thread_helper+0x4/0x10
[<ffffffff81066430>] ? finish_task_switch+0x80/0x110
[<ffffffff817d9c04>] ? retint_restore_args+0xe/0xe
[<ffffffff81097510>] ? __kthread_init_worker+0x70/0x70
[<ffffffff817db150>] ? gs_change+0xb/0xb
```

Line 2776 of block/cfq-iosched.c in v3.0-rc5 is as follows:

```
if (rcu_dereference(ioc->ioc_data) == cic) {
```

This form says that it must be in a plain vanilla RCU read-side critical section, but the "other info" list above shows that this is not the case. Instead, we hold three locks, one of which might be RCU related. And maybe that lock really does protect this reference. If so, the fix is to inform RCU, perhaps by changing __cfq_exit_single_io_context() to take the struct request_queue "q" from cfq_exit_queue() as an argument, which would permit us to invoke rcu_dereference_protected as follows:

```
if (rcu_dereference_protected(ioc->ioc_data,
                              lockdep_is_held(&q->queue_lock)) == cic) {
```

With this change, there would be no lockdep-RCU splat emitted if this code was invoked either from within an RCU read-side critical section or with the ->queue_lock held. In particular, this would have suppressed the above lockdep-RCU splat because ->queue_lock is held (see #2 in the list above).

On the other hand, perhaps we really do need an RCU read-side critical section. In this case, the critical section must span the use of the return value from rcu_dereference(), or at least until there is some reference count incremented or some such. One way to handle this is to add rcu_read_lock() and rcu_read_unlock() as follows:

```
rcu_read_lock();
if (rcu_dereference(ioc->ioc_data) == cic) {
        spin_lock(&ioc->lock);
        rcu_assign_pointer(ioc->ioc_data, NULL);
        spin_unlock(&ioc->lock);
}
rcu_read_unlock();
```

With this change, the rcu_dereference() is always within an RCU read-side critical section, which again would have suppressed the above lockdep-RCU splat.

But in this particular case, we don't actually dereference the pointer returned from rcu_dereference(). Instead, that pointer is just compared to the cic pointer, which means that the rcu_dereference() can be replaced by rcu_access_pointer() as follows:

```
if (rcu_access_pointer(ioc->ioc_data) == cic) {
```

Because it is legal to invoke rcu_access_pointer() without protection, this change would also suppress the above lockdep-RCU splat.