Subject: Re: [PATCH 1/7] introduce atomic_dec_and_lock_irqsave()
Posted by paulmck on Thu, 31 Aug 2006 22:58:28 GMT
View Forum Message <> Reply to Message

On Wed, Aug 30, 2006 at 07:25:07PM +0200, Roman Zippel wrote:
> Hi,
>
> On Wed, 30 Aug 2006, Dipankar Sarma wrote:
>
> > > > uidhash_lock can be taken from irq context. For example, delayed_put_task_struct()
> > > > does __put_task_struct()->free_uid().
> > >
> > > AFAICT it's called via rcu, does that mean anything released via rcu has
> > > to be protected against interrupts?
> >
> > No. You need protection only if you have are using some
> > data that can also be used by the RCU callback. For example,
> > if your RCU callback just calls kfree(), you don't have to
> > do a spin_lock_bh().
>
> In this case kfree() does its own interrupt synchronization. I didn't
> realize before that rcu had this (IMO serious) limitation. I think there
> should be two call_rcu() variants, one that queues the callback in a soft
> irq and a second which queues it in a thread context.

How about just using synchronize_rcu() in the second situation?
This primitive blocks until the grace period completes, allowing you to
do the remaining processing in thread context. As a bonus, RCU code
that uses synchronize_rcu() is usually quite a bit simpler than code
using call_rcu().

Using synchronize_rcu():

```
 list_del_rcu(p);
 synchronize_rcu();
 kfree(p);
```

Using call_rcu():

```
 static void rcu_callback_func(struct rcu_head *rcu)
 {
  struct foo *p = container_of(rcu, struct foo, rcu);

  kfree(p);
 }

 list_del_rcu(p);
 call_rcu(&p->rcu, rcu_callback_func);
```

Furthermore, the call_rcu() approach requires a struct rcu_head somewhere
in the data structure, so use of synchronize_rcu() saves a bit of memory,
as well.

But if you have a situation where neither synchronize_srcu() nor
call_rcu() is working out for you, let's hear it!

Thanx, Paul