Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Tue, 24 Apr 2007 18:22:50 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> Sure, I'll grep for cancel_delayed_work(). But unless I missed something,
> this change should be completely transparent for all users. Otherwise, it
> is buggy.

I guess you will have to make sure that cancel_delayed_work() is always
followed by a flush of the workqueue, otherwise you might get this situation:

```
 CPU 0    CPU 1
 =============================== ======================
     <timer expires>
 cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)
 kfree(x);   -->do_IRQ()
 y = kmalloc(); // reuses x
     <--do_IRQ()
     __queue_work(x)
 --- OOPS ---
```

That's my main concern.  If you are certain that can't happen, then fair
enough.

Note that although you can call cancel_delayed_work() from within a work item
handler, you can't then follow it up with a flush as it's very likely to
deadlock.

> > Because calling schedule_delayed_work() is a waste of CPU if the timer
> > expiry handler is currently running at this time as *that* is going to
> > also schedule the delayed work item.
>
> Yes. But otoh, try_to_del_timer_sync() is a waste of CPU compared to
> del_timer(), when the timer is not pending.

I suppose that's true.  As previously stated, my main objection to del_timer()
is the fact that it doesn't tell you if the timer expiry function is still
running.

Can you show me a patch illustrating exactly how you want to change
cancel_delayed_work()?  I can't remember whether you've done so already, but
if you have, I can't find it.  Is it basically this?:

```
 static inline int cancel_delayed_work(struct delayed_work *work)
 {
  int ret;
```

```
- ret = del_timer_sync(&work->timer);
+ ret = del_timer(&work->timer);
  if (ret)
   work_release(&work->work);
  return ret;
 }
```

I was thinking this situation might be a problem:

```
 CPU 0    CPU 1
 ================================ =======================
     <timer expires>
 cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)
 schedule_delayed_work(x,0) -->do_IRQ()
 <keventd scheduled>
 x->work()
     <--do_IRQ()
     __queue_work(x)
```

But it won't, will it?

> > Ah, but the timer routine may try to set the work item pending flag
> > *after* the work_pending() check you have here.
>
> No, delayed_work_timer_fn() doesn't set the _PENDING flag.

Good point. I don't think that's a problem because cancel_delayed_work()
won't clear the pending flag if it didn't remove a timer.

> First, this is very unlikely event, delayed_work_timer_fn() is very fast
> unless interrupted.

Yeah, I guess so.

Okay, you've convinced me, I think - provided you consider the case I
outlinded at the top of this email.

If you give me a patch to alter cancel_delayed_work(), I'll substitute it for
mine and use that that instead. Dave Miller will just have to live with that
patch being there:-)

David