

---

Subject: Re: Getting the new RxRPC patches upstream  
Posted by [Oleg Nesterov](#) on Tue, 24 Apr 2007 17:33:29 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 04/24, David Howells wrote:

>  
> Oleg Nesterov <oleg@tv-sign.ru> wrote:  
>  
> > Great. I'll send the s/del\_timer\_sync/del\_timer/ patch.  
>  
> I didn't say I necessarily agreed that this was a good idea. I just meant that  
> I agree that it will waste CPU. You must still audit all uses of  
> cancel\_delayed\_work().

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.

> > Aha, now I see what you mean. However. Why the code above is better then  
> >  
> > cancel\_delayed\_work(&afs\_server\_reaper);  
> > schedule\_delayed\_work(&afs\_server\_reaper, 0);  
> >  
> > ? (I assume we already changed cancel\_delayed\_work() to use del\_timer).  
>  
> 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.

> > 1: lock\_timer\_base(), return -1, skip schedule\_delayed\_work().  
> >  
> > 2: check timer\_pending(), return 0, call schedule\_delayed\_work(),  
> > return immediately because test\_and\_set\_bit(WORK\_STRUCT\_PENDING)  
> > fails.  
>  
> I don't see what you're illustrating here. Are these meant to be two steps in  
> a single process? Or are they two alternate steps?

two alternate steps.

1 means  
if (try\_to\_cancel\_delayed\_work())  
schedule\_delayed\_work();

2 means

```
cancel_delayed_work();
schedule_delayed_work();
```

> > So I still don't think try\_to\_del\_timer\_sync() can help in this particular  
> > case.  
>  
> It permits us to avoid the test\_and\_set\_bit() under some circumstances.

Yes. But lock\_timer\_base() is more costly.

```
> > To some extent, try_to_cancel_delayed_work is
> >
> > int try_to_cancel_delayed_work(dwork)
> > {
> >     ret = cancel_delayed_work(dwork);
> >     if (!ret && work_pending(&dwork->work))
> >         ret = -1;
> >     return ret;
> > }
> >
> > iow, work_pending() looks like a more "precise" indication that work->func()
> > is going to run soon.
>
> 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.

```
>
> Furthermore, it would be better to avoid
> the work_pending() check entirely because that check involves interacting with
> atomic ops done on other CPUs.
```

Sure, the implementation of try\_to\_cancel\_delayed\_work() above is just for illustration. I don't think we need try\_to\_cancel\_delayed\_work() at all.

```
>
> try_to_del_timer_sync() returning -1 tells us
> without a shadow of a doubt that the work item is either scheduled now or will
> be scheduled very shortly, thus allowing us to avoid having to do it ourself.
```

First, this is very unlikely event, delayed\_work\_timer\_fn() is very fast unless interrupted.

\_PENDING flag won't be cleared until this work is executed by run\_workqueue(). In general, work\_pending() after del\_timer() is imho better way to avoid the unneeded schedule\_delayed\_work().

But again, I can't understand the win for that particular case.

Oleg.

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---