Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 14:22:44 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > > > We only care when del_timer() returns true. In that case, if the timer
> > > > function still runs (possible for single-threaded wqs), it has already
> > > > passed __queue_work().
> > >
> > > Why do you assume that?
>
> Sorry, I should have been more clear. I meant the assumption that we only
> care about a true return from del_timer().
>
> > If del_timer() returns true, the timer was pending. This means it was
> > started by work->func() (note that __run_timers() clears timer_pending()
> > before calling timer->function). This in turn means that
> > delayed_work_timer_fn() has already called __queue_work(dwork), otherwise
> > work->func() has no chance to run.
>
> But if del_timer() returns 0, then there may be a problem. We can't tell the
> difference between the following two cases:
>
> (1) The timer hadn't been started.
>
> (2) The timer had been started, has expired and is no longer pending, but
>   another CPU is running its handler routine.
>
> try_to_del_timer_sync() _does_, however, distinguish between these cases: the
> first is the 0 return, the second is the -1 return, and the case where it
> dequeued the timer is the 1 return.

Of course, del_timer() and del_timer_sync() are different. What I meant the
latter buys nothing for cancel_delayed_work() (which in fact could be named
try_to_cancel_delayed_work()).

Let's look at (2). cancel_delayed_work() (on top of del_timer()) returns 0,
and this is correct, we failed to cancel the timer, and we don't know whether
work->func() finished, or not.

The current code uses del_timer_sync(). It will also return 0. However, it will
spin waiting for timer->function() to complete. So we are just wasting CPU.

I guess I misunderstood you. Perhaps, you propose a new helper which use
try_to_del_timer_sync(), yes? Unless I missed something, this doesn't help.

Because the return value == -1 should be treated as 0. We failed to stop
the timer, and we can't free dwork.

IOW, currently we should do:

```
 if (!cancel_delayed_work(dwork))
  cancel_work_sync(dwork));
```

The same if we use del_timer(). If we use try_to_del_timer_sync(),

```
 if (cancel_delayed_work(dwork) <= 0)
  cancel_work_sync(dwork));
```

(of course, dwork shouldn't re-arm itself).

Could you clarify if I misunderstood you again?

> BTW, can a timer handler be preempted? I assume not... But it can be delayed
> by interrupt processing.

No, it can't be preempted, it runs in softirq context.

Oleg.