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 generak, work_pending() after del_timer() is imho better way to avoid the
unneeded schedule_delayed_work().

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

Oleg.

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers