
Subject: Re: Getting the new RxRPC patches upstream
Posted by [David Howells](#) on Tue, 24 Apr 2007 16:58:27 GMT
[View Forum Message](#) <> [Reply to Message](#)

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

> > > 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.
> >
> > That's my objection to using `cancel_delayed_work()` as it stands, although in
> > most cases it's a relatively minor waste of time. However, if the timer
> > expiry routine gets interrupted then it may not be so minor... So, yes, I'm
> > in full agreement with you there.
>
> 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()`.

> 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.

> If `delayed_work_timer_fn()` is not running - both variants (let's denote them
> as 1 and 2) do the same.

Yes, but that's not the point.

> Now suppose that `delayed_work_timer_fn()` is running.
>
> 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?

> 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.

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

David

Containers mailing list

Containers@lists.linux-foundation.org

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