Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 16:40:01 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> 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 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.
>
> Consider how I'm using try_to_cancel_delayed_work(): I use this when I want to
> queue a delayed work item with a particular timeout (usually for immediate
> processing), but the work item may already be pending.
>
> If try_to_cancel_delayed_work() returns 0 or 1 (not pending or pending but
> dequeued) then I can go ahead and just schedule the work item (I'll be holding
> a lock to prevent anyone else from interfering).
>
> However, if try_to_cancel_delayed_work() returns -1 then there's no usually no
> point attempting to schedule the work item because I know the timer expiry
> handler is doing that or going to do that.
>
>
> The code looks like this in pretty much all cases:
>
```
>  if (try_to_cancel_delayed_work(&afs_server_reaper) >= 0)
>   schedule_delayed_work(&afs_server_reaper, 0);
```

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).

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

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.
```

So I still don't think try_to_del_timer_sync() can help in this particular
case.

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.

Oleg.