
Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by [Dean Nelson](#) on Thu, 17 May 2007 13:44:50 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Wed, May 02, 2007 at 09:44:11AM -0600, Eric W. Biederman wrote:
> Dean Nelson <dcn@sgi.com> writes:
> > On Thu, Apr 26, 2007 at 01:11:15PM -0600, Eric W. Biederman wrote:
> > >
> > > Ok. Because of the module unloading issue, and because we don't have
> > > a lot of these threads running around, the current plan is to fix
> > > thread_create and kthread_stop so that they must always be paired,
> > > and so that kthread_stop will work correctly if the task has already
> > > exited.
> > >
> > > Basically that just involves calling get_task_struct in kthread_create
> > > and put_task_struct in kthread_stop.
> > >
> > > Okay, so I need to expand upon Christoph Hellwig's patch so that all
> > > the kthread_create()'d threads are kthread_stop()'d.
> > >
> > > This is easy to do for the XPC thread that exists for the lifetime of XPC,
> > > as well as for the threads created to manage the SGI system partitions.
> > >
> > > XPC has the one discovery thread that is created when XPC is first started
> > > and exits as soon as it has finished discovering all existing SGI system
> > > partitions. With your forthcoming change to kthread_stop() that will allow
> > > it to be called after the thread has exited, doing this one is also easy.
> > > Note that the kthread_stop() for this discovery thread won't occur until
> > > XPC is rmmmod'd. This means that its task_struct will not be freed for
> > > possibly a very long time (i.e., weeks). Is that a problem?
> > >
> > > As long as there is only one, not really. It would be good if we could
> > > get rid of it though.
> > >
> > > The practical problem is the race with rmmmod, in particular if someone
> > > calls rmmmod while this thread is still running.

I guess I'm not seeing the race with rmmmod that you're talking about? In XPC's case, rmmmod calls xpc_exit() which currently does a wait_for_completion() on the discovery thread and on the other thread mentioned above. These will be changed to kthread_stop() calls. And if the discovery thread has already exited the kthread_stop() will return immediately and if not it will wait until the discovery thread has exited. rmmmod won't return from xpc_exit() until both threads have exited.

Any thought as to when the changes to kthread_stop() that allow it to be called for a kthread that has already exited will get into the -mm tree?

> > Is there any way to have a version of kthread_create() that doesn't
> > require a matching kthread_stop()? Or add a kthread_not_stopping()
> > that does the put_task_struct() call, so as to eliminate the need for
> > calling kthread_stop()?
>
> Yes. I was thinking calling it kthread_orphan or something like that.
> We can't make anything like that the default, because of the modular
> remove problem, but it's not too hard.

Again, when xpc_exit() is called by rmmmod it waits for XPC's pool of threads to exit before it returns, so not a problem.

Any thought as to when kthread_orphan() will get into the -mm tree? Once kthread_stop() is changed and kthread_orphan() added I can proceed with a patch to change XPC to use the kthread API.

> > Or should we reconsider the kthread pool approach
> > (and get XPC out of the thread management business altogether)? Robin
> > Holt is putting together a proposal for how one could do a kthread pool,
> > it should provide a bit more justification for going down that road.

Robin has changed his mind about tying in the management of a pool of threads with the kthread API, so there won't be the fore mentioned proposal.

Thanks,
Dean

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