Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Mon, 30 Apr 2007 15:22:30 GMT
View Forum Message <> Reply to Message

On Fri, Apr 27, 2007 at 02:33:32PM -0600, Eric W. Biederman wrote:
> Dean Nelson <dcn@sgi.com> writes:
>
> > On Fri, Apr 27, 2007 at 12:34:02PM -0600, Eric W. Biederman wrote:
> >> Dean Nelson <dcn@sgi.com> writes:
> >> >
> >> > XPC is in need of threads that can block indefinitely, which is why XPC
> >> > is in the business of maintaining a pool of threads. Currently there is
> >> > no such capability (that I know of) that is provided by linux. Workqueues
> >> > can't block indefinitely.
> >>
> >> I'm not certain I understand this requirement. Do you mean block indefinitely
> >> unless requested to stop?
> >
> > These threads can block waiting for a hardware DMA engine, which has a 28
> > second timeout setpoint.
>
> Ok. So this is an interruptible sleep?

No, the hardware DMA engine's software interface, doesn't sleep
nor relinquish the CPU. But there are other spots where we do sleep
interruptibly.

> Do you have any problems being woken up out of that interruptible sleep
> by kthread_stop?
>
> I am in the process of modifying kthread_stop to wake up thread in an
> interruptible sleep and set signal_pending, so they will break out.

No, this is fine, just avoid designing the kthread stop mechanism
to require a thread being requested to stop to actually stop in some
finite amount of time, and that by it not stopping other kthread stop
requests are held off. Allow the thread to take as much time as it needs
to respond to the kthread stop request.

> >> > And for performance reasons these threads need to be able to be created
> >> > quickly. These threads are involved in delivering messages to XPC's users
> >> > (like XPNET) and we had latency issues that led us to use kernel_thread()
> >> > directly instead of the kthread API. Additionally, XPC may need to have
> >> > hundreds of these threads active at any given time.
> >>
> >> Ugh. Can you tell me a little more about the latency issues?
> >
> > After placing a message in a local message queue, one SGI system partition

> > will interrupt another to retrieve the message. We need to minimize the
> > time from entering XPC's interrupt handler to the time that the message
> > can be DMA transferred and delivered to the consumer (like XPNET) to
> > whom it was sent.
> >
> >> Is having a non-halting kthread_create enough to fix this?
> >> So you don't have to context switch several times to get the
> >> thread running?
> >>
> >> Or do you need more severe latency reductions?
> >>
> >> The more severe fix would require some significant changes to copy_process
> >> and every architecture would need to be touched to fix up copy_thread.
> >> It is possible, it is a lot of work, and the reward is far from obvious.
> >
> > I think a non-halting kthread_create() should be sufficient. It is in
> > effect what XPC has now in calling kernel_thread() directly.
>
> A little different but pretty close.
>
> We call kthread_create() it prepares everything and places it on
> a queue and wakes up kthreadd.
>
> kthreadd then wakes up and forks the thread.
>
> After the thread has finishing setting up it will call complete on
> a completion so kthread_create can continue on it's merry way
> but it should not need to go to sleep waiting for someone to
> call kthread_bind.
>
> But if you can live with what I have just described that will
> be easy to code up.
>
> It is a little slower then kernel_thread but hopefully not much.

I was aware of this behavior of kthread_create(), which I consider
'halting' in that the thread doing the kthread_create() blocks waiting
for kthreadd to get scheduled, call kernel_thread(), and then call
complete(). By your mentioning a 'non-halting' kthread_create() I
thought you were planning to create a new flavor of kthread_create()
that called kernel_thread() directly and reparented the child thread to
kthreadd. My mistake.

So there will be more overhead (time-wise) for XPC in calling
kthread_run() as opposed to it formerly calling kernel_thread() directly.
Thus requiring XPC to utilize a pool of kthread_create()'d threads.

> > Taking it one step further, if you added the notion of a thread pool,

> > where upon exit, a thread isn't destroyed but rather is queued ready to
> > handle the next kthread_create_quick() request.
>
> That might happen. So far I am avoiding the notion of a thread pool for
> as long as I can. There is some sense in it, especially in generalizing
> the svc thread pool code from nfs. But if I don't have to go there I would
> prefer it.

This means that XPC will have to retain its thread pool, but I can
understand you not wanting to go there.

Thanks,
Dean

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