Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Wed, 02 May 2007 15:44:11 GMT
View Forum Message <> Reply to Message

Dean Nelson <dcn@sgi.com> writes:

```
> On Mon, Apr 30, 2007 at 10:22:30AM -0500, Dean Nelson wrote:
>> On Fri, Apr 27, 2007 at 02:33:32PM -0600, Eric W. Biederman wrote:
>> > Dean Nelson <dcn@sgi.com> writes:
>> >
>> > > 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.
>
> 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 rmmod'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 rmmod, in particular if someone
calls rmmod while this thread is still running.

If I get clever I think this is likely solvable with something like.

```
kthread_maybe_stop(struct task_struct **loc)
{
	struct task_struct *tsk;
	tsk = xchg(loc, NULL);
	if (tsk)
	 kthread_stop(tsk);
}

kthread_stop_self(struct task_struct **loc, int exit_code)
{
	struct task_struct *tsk;

	tsk = xchg(loc, NULL);
	if (tsk)
	 put_task_struct(tsk);
 do_exit(tsk);
}
```

I'm not quite convinced that is a common enough paradigm to implement
that.

> But then we come to XPC's pool of threads that deliver channel messages
> to the appropriate consumer (like XPNET) and can block indefinitely. As
> mentioned earlier there could be hundreds if not thousands of these
> (our systems keep getting bigger). So now requiring a kthread_stop()
> for each one of these becomes more of a problem, as it is a lot of
> task_struct pointers to maintain.
>
> Currently, XPC maintains these threads via a
> wait_event_interruptible_exclusive() queue so that it can wakeup as many
> or as few as needed at any given moment by calling wake_up_nr(). When XPC
> is rmmod'd, a flag is set which causes them to exit and wake_up_all()
> is called. Therefore XPC dosen't need to remember their pids or
> task_struct pointers.
>
> So what would you suggest we do for this pool of threads?

Good question.

The whole concept of something that feels like a core part of the
platform code being modular I'm still looking at strange.

> 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 to hard.

> 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.

Eric