Subject: Re: Summary of resource management discussion
Posted by Herbert Poetzl on Fri, 16 Mar 2007 14:26:45 GMT
View Forum Message <> Reply to Message

On Thu, Mar 15, 2007 at 12:12:50PM -0700, Paul Menage wrote:
> On 3/15/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> > On Thu, Mar 15, 2007 at 04:24:37AM -0700, Paul Menage wrote:
> > > If there really was a grouping that was always guaranteed to match
> > > the way you wanted to group tasks for e.g. resource control, then
> > > yes, it would be great to use it. But I don't see an obvious
> > > candidate. The pid namespace is not it, IMO.
> >
> > In vserver context, what is the "normal" case then? Atleast for
> > Linux Vserver pid namespace seems to be normal unit of resource
> > control (as per Herbert).
>
> Yes, for vserver the pid namespace is a good proxy for resource
> control groupings. But my point was that it's not universally
> suitable.
>
> >
> > (the best I could draw using ASCII art!)
>
> Right, I think those diagrams agree with the point I wanted to make -
> that resource control shouldn't be tied to the pid namespace.

first, strictly speaking they aren't (see previous mail)
it is more the lack of a separate pid space for now which
basically makes pid space == context, and in turn, the
resource limits are currently tied to the context too,
which again addresses the very same group of tasks

I'm fine with having a separate pid space, and another
(possibly different) cpu limit space or resource limit
space(s) as long as they do not complicate the entire
solution without adding any _real_ benefit ...

for example, it might be really nice to have a separate
limit for VM and RSS and MEMLOCK and whatnot, but IMHO
there is no real world scenario which would require you
to have those limits for different/overlaping groups
of tasks ... let me know if you have some examples

best,
Herbert

> > The benefit I see of this approach is it will avoid introduction
> > of additional pointers in struct task_struct and also additional

> > structures (struct container etc) in the kernel, but we will still
> > be able to retain same user interfaces you had in your patches.

> > Do you see any drawbacks of doing like this? What will break if we
> > do this?
>
> There are some things that benefit from having an abstract
> container-like object available to store state, e.g. "is this
> container deleted?", "should userspace get a callback when this
> container is empty?". But this indirection object wouldn't need to be
> on the fast path for subsystem access to their per-taskgroup state.
>
> > > >a. Paul Menage's patches:
> > > >
> > > > (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
> > >
> > > So what's the '-X' that you're referring to
> >
> > Oh ..that's to seek pointer to begining of the cpulimit structure (subsys
> > pointer in 'struct container' points to a structure embedded in a larger
> > structure. -X gets you to point to the larger structure).
>
> OK, so shouldn't that be listed as an overhead for your rcfs version
> too? In practice, most subsystems that I've written tend to have the
> subsys object at the beginning of the per-subsys state, so X = 0 and
> is optimized out by the compiler. Even if it wasn't, X is constant and
> so won't hurt much or at all.
>
> >
> > Yes me too. But maybe to keep in simple in initial versions, we should
> > avoid that optimisation and at the same time get statistics on duplicates?.
>
> That's an implementation detail - we have more important points to
> agree on right now ...
>
> Paul
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers