Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 16:50:52 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 11:57:18AM -0700, Paul Menage wrote:
> > Very true. That's a bug and can be rectified. Atm however in my patch
> > stack, I have dropped this whole find_nsproxy() and instead create a new
> > nsproxy whenever tasks move ..Not the best I agree on long run.
>
> Or even short term, I think - although it won't hurt too much for
> cases where a single process enters a container and all children stay
> in the container, it'll hurt a lot in cases where you ever move the
> contents of one container into another. (Possibly in an orthogonal
> hierarchy).
>
> BTW, what does rcfs do if a subsystem is registered when there are
> already mounted hierarchies? It looks as though it leaves the relevant
> pointers as NULLs.

The approach I am on currently doesnt deal with dynamically loaded
modules ..Partly because it allows subsystem ids to be compile-time
decided and also init_nsproxy.ctlr_data[] can be initialised to default
values at compile time itself.

For ex: init_nsproxy.ctlr_data[CPUSET_ID] can be set to &top_cpuset
at compile time.

If there is demand later, I agree we need to address the shortcoming you
point out ..

> I think you're only right if you're going to overload the nsproxy
> count field as the total refcount, rather than the number of tasks. If
> you do that then you risk having to allocate way more memory than you
> need in any routine that tries to allocate an array with one pointer
> per task in the container (e.g. when reading the tasks file, or moving
> a task into a new cpuset). My patch separates out the refcounts from
> the tasks counts for exactly that reason - there's a per-subsys_state
> refcount which can be cheaply manipulated via the subsystem. (I have a
> version with lower locking requirements that I've not yet posted).

We have been thr' this in a diff thread. I trust we have discussed all
that need to be discussed here?

> > Again note that I am not hell-bent on avoiding a container-like object
> > to store shared state of a group. My main desire was to avoid additional
> > pointer in task_struct and reuse nsproxy if possible. If the grand scheme of
> > things requires a 'struct container' object in addition to reusing ->nsproxy,

> > to store shared state like 'notify_on_release', 'hierarchical information'
> > then I have absolutely no objection.
>
> OK, but at that point you're basically back at my patches, but using
> nsproxy rather than container_group.

Ok ..by posting rcfs patches, I didn't mean to introduce a "yours" and
"mine" rift ..honestly. In fact you would notice that they have your
(sole) copyright still on them! It took me just two days to convert over the
patches to use nsproxy and come up with the rcfs patches and obviously I
couldnt have done that without your excellent patches to start with.

So if there is is larger interest in using nsproxy at some point, I hope
they serve as some reference patches to do the conversion. I would go ahead
and post what I have now (which incorporates several bug fixes) which
could be used as you deem necessary at a later point if/when considering
moving to nsproxy.

> > Why would it mean duplicating code? A generic function which takes a
> > dentry pointer and returns its vfs path will avoid this code
> > duplication?
>
> It's still adding boilerplate (extra pointers, extra /proc files,
> logic to hook them together) to every subsystem that needs this, that
> could be avoided. But I guess this aspect of it isn't a huge issue.

I agree that is not a big issue now (considering there are larger things
to go after!).

> > > > Did you mean to say "when the number of aggregators sharing the same
> > > > container object are more" ?
> > >
> > > Yes. Although having thought about the possibility of null groupings
> > > that I described above, I'm no longer convinced that argument is
> > > valid.
> >
> > I think the null grouping as defined so far is very fuzzy ..How would
> > the kernel use this grouping, which would require reserving N pointers
> > in 'struct container_group'/'struct nsproxy'?
>
> The kernel wouldn't use it, other than to act as an inescapable task
> grouping whose members could always be easily listed from userspace.

I am still trying to come to terms with this null groupings and how they
would be used in real life.

 - Can you list a real world use of it?

- If they are "inescapable" task groups, how does the first task enter
  such a group, using just the filesystem interface?

- If there is no real kernel use for such groups, can this be
  implemented in userspace (user ids, session ids etc)

- If userspace soln is not feasible, why would you want such null
  groupings in multiple hierarchies and not in one hierarchy?
  If having them in one hierarchy satisfies the requirements,
  then we could create empty cpusets at top level (or at an
  appropriate subcpuset level) and use them as null groups?
  By defining permissionss of the null-group cpuset directories,
  we could restrict movement of tasks across groups ?

--
Regards,
vatsa