Subject: Re: [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by serue on Wed, 07 Mar 2007 17:43:46 GMT
View Forum Message <> Reply to Message

Quoting Srivatsa Vaddagiri (vatsa@in.ibm.com):
> On Tue, Mar 06, 2007 at 06:32:07PM -0800, Paul Menage wrote:
> > I'm not really sure that I see the value of having this be part of
> > nsproxy rather than the previous independent container (and
> > container_group) structure.
>
> *shrug*
>
> I wrote the patch mainly to see whether the stuff container folks (Sam Vilain
> et al) were complaining abt (that container structure abstraction
> inside the kernel is redundant/unnecessary) made sense or not.

I still think the complaint was about terminology, not implementation.
They just didn't want you calling them containers.

> The rcfs patches demonstrate that it is possible to implement resource control
> on top of just nsproxy -and- give the same interface that you now
> have. In essense, I would say that the rcfs patches are about 70% same as your
> original V7 container patches.
>
> However as I am converting over cpusets to work on top of nsproxy, I
> have learnt few things:
>
> container structure in your patches provides for these things:
>
> a. A way to group tasks
> b. A way to maintain several hierarchies of such groups
>
> If you consider just a. then I agree that container abstraction is
> redundant, esp for vserver resource control (nsproxy can already be used
> to group tasks).
>
> What nsproxy doesn't provide is b - a way to represent hierarchies of
> groups.
>
> So we got several choices here.
>
> 1. Introduce the container abstraction as is in your patches
> 2. Extend nsproxy somehow to represent hierarchies
> 3. Let individual resource controllers that -actually- support
>   hierarchical resource management maintain hierarchy in their code.
>
> In the last option, nsproxy still is unaware of any hierarchy. Some of

> the resource objects it points to (for ex: cpuset) may maintain a
> hierarchy. For ex: nsproxy->ctlr_data[cpuset_subsys.subsys_id] points to
> a 'struct cpuset' structure which could maintains the hierarchical
> relationship among cpuset objects.
>
> If we consider that most resource controllers may not implement hierarchical
> resource management, then 3 may not be a bad compromise. OTOH if we
> expect *most* resource controllers to support hierarchical resource
> management, then we could be better of with option 1.
>
> Anyway, summarizing on "why nsproxy", the main point (I think) is about
> using existing abstraction in the kernel.

But nsproxy is not an abstraction, it's an implementation
detail/optimization. I'm mostly being quiet because i don't
particularly care if it gets expanded upon, but it's nothing more than
that right now.

> > As far as I can see, you're putting the
> > container subsystem state pointers and the various task namespace
> > pointers into the same structure (nsproxy) but then they're remaining
> > pretty much independent in terms of code.
> >
> > The impression that I'm getting (correct me if I'm wrong) is:
> >
> > - when you do a mkdir within an rcfs directory, the nsproxy associated
> > with the parent is duplicated, and then each rcfs subsystem gets to
> > set a subsystem-state pointer in that nsproxy
>
> yes.
>
> > - when you move a task into an rcfs container, you create a new
> > nsproxy consisting of the task's old namespaces and its new subsystem
> > pointers. Then you look through the current list of nsproxy objects to
> > see if you find one that matches. If you do, you reuse it, else you
> > create a new nsproxy and link it into the list
>
> yes
>
> > - when you do sys_unshare() or a clone that creates new namespaces,
> > then the task (or its child) will get a new nsproxy that has the rcfs
> > subsystem state associated with the old nsproxy, and one or more
> > namespace pointers cloned to point to new namespaces. So this means
> > that the nsproxy for the task is no longer the nsproxy associated with
> > any directory in rcfs. (So the task will disappear from any "tasks"
> > file in rcfs?)
>
> it "should" disappear yes, although I haven't carefully studied the

> unshare requirements yet.
>
> > You seem to have lost some features, including fork/exit subsystem callbacks
>
> That was mainly to keep it simple for a proof-of-concept patch! We can add it
> back later.
>
> > >What follows is the core (big) patch and the cpu_acct subsystem to serve
> > >as an example of how to use it. I suspect we can make cpusets also work
> > >on top of this very easily.
> >
> > I'd like to see that. I suspect it will be a bit more fiddly than the
> > simple cpu_acct subsystem.
>
> I am almost done with the conversion. And yes cpuset is a beast to
> convert over! Will test and send the patches out tomorrow.
>
> --
> Regards,
> vatsa

_____