Subject: Re: [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by serue on Wed, 07 Mar 2007 20:58:46 GMT
View Forum Message <> Reply to Message

Quoting Srivatsa Vaddagiri (vatsa@in.ibm.com):
> On Wed, Mar 07, 2007 at 11:43:46AM -0600, Serge E. Hallyn wrote:
> > I still think the complaint was about terminology, not implementation.
>
> I don't think that is what http://lkml.org/lkml/2007/2/12/426 conveyed!

I don't have that in my inbox anymore so won't reply to it itself unfortunately, but what it conveyed is also not that nsproxy should be the 'resource control' object. If anything it seems to argue that all of Paul's patchset should be done in userspace.

Sam writes

> That's a great idea for a set of
> tightly integrated userland utilities to simplify the presentation to
> the admin, but I don't see why you need to enshrine this in the kernel.
> Certainly not for any of the other patches in your set as far as I can
> see.

I disagree.

Sam, there are two very separate concepts here. Actually three.

What you had originally presented in a patchset was resource virtualization: so when process A asks for some resource X, rather than get resource_table[X] he gets resource_table[hash(x)]. The concensus you mention at the start, and which you say Eric was arguing for, was to not do such translation, but just get rid of the global 'resource_table'. By allowing processes to have and manipulate their own private resource_table, you implement namespaces.

And as I've said, the nsproxy is just an implementation detail to keep namespace pointers out of the task struct. Using nsproxy or not has nothing to do with the namespace approach versus global resource tables with id translation at all userspace->kernel boundaries.

So virtualization via explicit translation through global resource tables is one concept, and private namespaces could be said to be a second. The third is resource controls, which Paul's container patchset implements. It has nothing to do with the previous two, and it is what Paul's patches are addressing.

Sam asks:

> Ask yourself this - what do you need the container structure for so
> badly, that virtualising the individual resources does not provide for?

To keep track of a process' place in any of several hierarchies, where each node in a tree inherit default values from the parent and can customize in some way.

> You don't need it to copy the namespaces of another process ("enter")
> and you don't need it for checkpoint/migration.

Because these have nothing to do with resource controls.

> What does it mean to make a new container?

It means to create a new set of limits, probably inheriting defaults from a parent set of limits, customizable but probably within some limits imposed by the parent. For instance, if there is a cpuset container which limits its tasks to cpus 7 and 8, then when a new container is created as a child of that one, it can be further restricted, but can never have more than cpus 7 and 8.

> That's a great idea for a set of
> tightly integrated userland utilities to simplify the presentation to
> the admin, but I don't see why you need to enshrine this in the kernel.

If you want to argue that resource controls should be done in userspace i *suspect* you'll find that approach insufficient but am interested to see attempts to do so.

But just moving the container structure into nsproxy is just that - moving the container structure. Since Sam argues vehemently that there should be no such thing, I don't see how he can be seen as wanting to move it.

All that being said, if it were going to save space without overly complicating things I'm actually not opposed to using nsproxy, but it looks to me like it does complicate things. One reason for this is that through the nsproxy subsystem we are mixing pointers to data (the namespaces) and pointers to pointers to the same data (nsproxy subsystem containers pointing to nsproxies) in the same structure. Yuck.

-serge