Subject: Re: Thoughts on Namespace / Subsystem unification
Posted by Paul Menage on Mon, 03 Sep 2007 16:27:14 GMT
View Forum Message <> Reply to Message

On 9/3/07, Eric W. Biederman <ebiederm@xmission.com> wrote:
> >
> > -how does a process in the parent namespace read/write the shmmni
> > value in the child namespace? Having to fork a child into the
> > namespace via something like sys_hijack() seems overly expensive.
>
> When complete we should be able to see the appropriate /proc file
> from if we can see the process. This is just a matter of sorting
> out the implementation in /proc.

So inside a child namespace we have a "limit" values such as shmmni, which normally are system-wide value that root can set to limit total resource usage in the OS; if the only way to access this value is via a namespace view, then who "owns" that value? - is it something that the parent namespace can set to limit the child's total resource usage, or is it something that the child can set to limit its own total resource usage. If the former, then we break virtualization a bit since in the child the value will be read-only; if the latter then the parent has less control over the child's resource usage that it would like. A separate monitoring/control API that the parent (who *knows* that virtualization is involved) can access lets you set an external limit as well as an internal limit.

> >
> > But the fact that this is a distinction between namespaces and
> > subsystems is a bit artificial. I think it's quite possible to imagine
> > some namespaces whose implementation can quite easily handle tasks
> > changing their namespace pointer unexpectedly, since they're written
> > to handle the tricky issues this introduces, and aren't so performance
> > critical that they can't do locking when necessary.
>
> So far I think the extra volatility of subsystems is a misfeature.
> I think with a little care you could get the cheapness of the
> current namespaces with the flexibility of containers. Although
> this is something that needs great care.

For read-only access to a subsystem state object, an rcu_read_lock() is sufficient - once in the RCU section, access to a subsystem state is as cheap as accessing a namespace - they're both constant indexed offsets from a pointer in task_struct. Similarly if you're updating a value but aren't too worried if you update the state that the task just moved away from (e.g. in a rate-based scheduler, you probably don't care too much if you charge the task's old container for, say, CPU cycles, rather than its new container).

>
> Maybe. So far the subsystems interfaces to user space seem
> overdesigned and inflexible in really weird ways to me.

Can you elaborate on that? I'm always interested in trying to make my
interfaces less weird where possible ...

> > The ns_container subsystem is a first step towards linking subsystems
> > and namespaces - it associates an entire set of namespaces (via an
> > nsproxy) with a "task container", so the nsproxy is on the same level
> > with other subsystems.
>
> And I never understood why anyone did it that way.

The ns_container subsystem lets you name a collection of namespaces,
and associate them with a bunch of resource controllers, so in that
sense it's definitely useful. The question in my mind is whether it
goes far enough.

> > - should a process be allowed to enter this "container" (a property
> > specified by the code itself)
>
> There are weird security aspects to enter which is why Serge's
> sys_hijack thing may make more sense. Frankly I'm not convinced
> that there is a better way to do things.

Agreed for some kinds of namespaces, which is why certain
namespaces/subsystems might want to declare that they can't be entered
without a hijack-like interface. But I think it's too strong a
restriction to enforce for all namespaces/subsystems.

For instance:

- a webserver that wants to charge resource usage based on which
client it's currently doing work for, needs to be able to shuffle its
threads between different resource controller instances.

- we're experimenting with using cpusets for memory isolation; part of
this involves an early initscript that creates a "sys" container and
isolates all the running system daemons in that container. To do this
with no support for migrating tasks between cpusets, and only relying
on sys_hijack, would involve a custom init, I think, rather than a
simple initscript.

>
> Currently I am not convinced that we want a first class container
> object in the kernel. There is all kinds of weirdness that results.

Such as? The "first class container" object doesn't have to be called a container, or even directly analogous to a "virtual server container". It's just a way to name an association between a collection of namespaces/subsystems and a collection of tasks.

Paul

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers