Subject: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Thu, 23 Mar 2006 04:17:27 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

>>I tried to make it as basic as possible, but my basic problem with the
>>patch you linked to is that it shouldn't be so small that you don't get
>>a decent bunch of the internal API functions listed in description of
>>part 1 of the set
>>(http://vserver.utsl.gen.nz/patches/utsl/2.6.16-rc4-vsi/01-VS erver-Umbrella.diff)
>>
>>
>
>Right. I think the smallest thing that we can reasonably discuss with
>real problems is the sysvipc namespace. This is why I suggested efforts
>in that direction.
>
>

I apologise for not contributing to that effort, however I was expecting
to be able to plug the deliverables of it into this framework. I hope to
participate in as many of these subsystem virtualisation discussions as
I can, but at that time it looked like there was enough input going in.

>>But I think the whole point of this patchset (which sadly vger ate for
>>the wider audience, and I still don't know why) is to group tasks and to
>>give userland, and hence the administrator, something tangible with
>>which to group processes with and tack namespace structures onto. I can
>>split out the ID related stuff into another patch, but it would need to
>>go back in before a useful syscall interface can be added. I'll
>>consider it anyway, though.
>>
>>
>So as best I can determine that is simply an implementation optimization.
>(That is with a little refactoring we can add a structure like that later
> if the performance concerns warrant it.)
>
>Skipping that optimization we should be able to concentrate on the
>fundamentals. Which should be simpler and allow better forward progress.
>
>

Sure, this is where we came to last time. I agree that it is an
optimisation, but not just an implementation optimisation.

In the abstract sense you have tuples of (task, namespace) for each type
of namespace. However for (task, utsname) this is a much weaker thing to

want per-process, and much more like you want it per process family. So, it is also a design optimisation. Some tuples really are (family, namespace) in the first instance.

>>Note that a given vx_info structure still might share namespace
>>structures with other vx_info objects - this is what I was alluding to
>>with the "we haven't made any restrictions on the nature of
>>virtualisation yet" comment. All we've made is the (XID, PID) tuple
>>(although the default for XID=0 is that all processes are visible in one
>>big PID space). The intention is that flags will control how you want
>>your PIDs to work for that vserver.
>>
>>
>Do we need flags or can we move this to user space?
>
>

Some of them influence behaviour in fastpath code. For instance, one example of a flag is the scheduling policy. A process family might have a flag set on it to group its scheduling together, to provide assertions like "this family should get ¼ of the CPU". Clearly userspace can't make this call. But perhaps I don't get your meaning.

I think they are a pragmatic necessity.

>>The only thing that they can't do is share processes - it is a one to
>>many relationship. However, if there can be a parent/child
>>containership relationship on the XIDs themselves, you can still achieve
>>the behaviour of these unusual situations. I'm working on the
>>assumption that if we ever have to migrate trees of XIDs then we can
>>virtualise how they look inside a vserver using flags.
>>
>>Eric, perhaps you can comment or refer to the earlier post where you
>>made this argument. I tried to follow it but perhaps I missed the jist
>>of one of the messages or one of the most important messages entirely.
>>
>>
>
>I'm not certain which argument you are referring to but I will state my general
>argument against adding additional global ids.
>
>Ultimately you want to nest you vservers and the like inside of each
>other, because if you haven't captured everything you can do in user
>space there will be some applications that don't work, and ultimately
>it is desirable to support all applications. When doing that you want
>to use the same mechanism on the inside as you have on the outside.
>Additional global identifiers make this very difficult.
>

>

Sure, you might set a flag on a process family to allow family_id values
to be rewritten on the system/user barrier.

>You can always talk about resources by specifying the processes that
>use them. It is one level of indirection, but it works and is simple.
>
>

Sometimes you might not know of any particular processes in the family.
Maybe it's got one process that is just doing "while (1) { fork && exit
}". How do you "catch" it ?

>>For the record, I like the term "process family" most. It implies the
>>possibility strict grouping, like last name, as well as allowing
>>heirarchies, but of course in our modern, post-nuclear family age, does
>>not imply a fixed nature of anything ;-).
>>
>>
>
>Families don't sound bad, but this all feels like putting the cart
>before the horse. It is infrastructure solving a problem that I am
>not at all certain is interesting.
>
>

So, you're breeding and selecting the horse. Fine, can't I start nailing
together the frame for the cart, and tanning the leather for the bridle?
Surely whichever horse we end up with, we'll still need a cart. And
don't forget the stables, too. We've got a lot of stables already, so we
know roughly what size the cart needs to be. Surely we won't want people
knocking those down and rebuilding them.

I'm just a bit concerned about this merging effort turning into a
research project. I'd rather try and merge the approaches used by
systems people are currently using than try to come up with something
wildly new.

After all, if the new developments are that flexible, they should be
easily able to support the presented interfaces ranging from the circa
1999 FreeBSD jail() to a more modern vserver/container/openvz/etc API.

Sam.