Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Tue, 07 Feb 2006 16:57:04 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:
> Rik van Riel wrote:
>
>> On Mon, 6 Feb 2006, Eric W. Biederman wrote:
>>
>>
>>> We are never going to form a consensus if all of the people doing
>>> implementations don't talk.
>>
>>
>> Speaking of which - it would be interesting to get Kirill's
>> comments on Eric's patchset ;)
>>
>> Once we know what's good and bad about both patchsets, we'll
>> be a lot closer to knowing what exactly should go upstream.
>
>
> Let's compare approaches of patchsets before the patchsets themselves.
>
> It seems to be, should we:
>
> A) make a general form of virtualising PIDs, and hope this assists
> later virtualisation efforts (Eric's patch)
>
> B) make a general form of containers/jails/vservers/vpses, and layer
> PID virtualisation on top of it somewhere (as in openvz, vserver)
>
> I can't think of any real use cases where you would specifically want A)
> without B).
>
> Also, the problem space in B) is now very well explored. To start with
> A) would mean to throw away 4+ years of experience at this approach
> (just counting vserver and variants - not FreeBSD Jail, etc). Trying to
> re-base B) atop a massive refactoring and new patch like A) would incur
> a lot of work; however fitting it into B) is natural and solved
> conceptually and in practice, with the only drawback I see being that
> the use cases mentioned above wouldn't suffer from the side-effects of
> B).
>
Sam, that is a bit far fetched. I looked and experienced myself with both
approaches and there is a lot of functional overlap, with both of them
having advantages and disadvantages.
What Eric provides is an alternative to the PID virtualization part of openvz.
Indeed it is a pid isolation more then anything else (with some dealing at

the boundary condition).
I personally don't see much problem in replacing the pid virtualization of
openvz with that of pidspaces.
So the correct thing to do here is as RvR points out, simple discuss the
merits and drawbacks of each PID approach for now and settle on one and
move on....

Here are my two cents on this.

The pid-namespace (pspace) provides an approach of fully separate
the allocation and maintenance of the pids and treating the <pspace,pid>
tuple as an entity to uniquely identify a task and vice versa.
As a result the logic of lookup can be pushed down the find_task_by_pid()
lookup. There are specific cases where the init_task of a container or
pspace needs to be checked to ensure that signals/waits and alike are properly
handled across pspace boundaries. I think this is an intuitive and clean way.
It also completely avoids the problem of having to think about all the locations
at the user/kernel boundary where a vpid/pid conversion needs to take place.
It also avoids the problems that logically vpids and pids are different types and
therefore it would have been good to have type checking automatically identify
problem spots.
On the negative side, it does require to maintain a pidmap per pidspace.

The vpid approach has the drawbacks of having to identify the conversion spots
of all vpid vs. pid semantics. On the otherhand it does take advantage
of the fact that no virtualization has to take place until a "container"
has been migrated, thus rendering most of the vpid<->pid calls to be
noops.

What I like about the pspace approach is that it explicitely defines in the code
when I am using a different pspace for the lookup. That is kind of hidden in
the vpid/pid approach.

The container is just an umbrella object that ties every "virtualized" subsystem
together.

So, what do other folks, see as pluses and minus of each approach.
Once we have a more complete listing of these, maybe the decision becomes
more obvious !

Regards
-- Hubertus