Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Thu, 09 Feb 2006 00:24:16 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> Hello,
>
> I tried to take into account all the comments from you guys (thanks a lot for
> them!) and prepared a new version of virtualization patches. I will send only 4
> patches today, just not to overflow everyone and keep it clear/tidy/possible to
> review.
>
> This patch introduces some abstract container kernel structure and a number of
> operations on it.
>
> The important properties of the proposed container implementation:
> - each container has unique ID in the system
> - each process in the kernel can belong to one container only
> - effective container pointer (econtainer()) is used on the task to avoid
> insertion of additional argument "container" to all functions where it is
> required.
> - kernel compilation with disabled virtualization should result in old good
> linux kernel
>
> Patches following this one will be used for virtualization of the kernel
> resources based on this container infrastructure, including those VPID patches I
> sent before. Every virtualized resource can be given separate config option if
> needed (just give me to know if it is desired).

After having digested this I think there is something sane with regard to
this container idea. I still think the implementation is totally wrong but
there is a potential problem the basic idea solves.

In the traditional linux/plan9 style of namespaces the question is
which resources different tasks share, and we pass clone bits to determine
what we want to share and what we don't want to share. Semantically this
is very clean and allows for a great deal of flexibility. However
as the flexibility increases we get more code in do_fork more
reference counts and ultimately the performance decreases. In addition
it is not common for us to change which resources we share.

So our traditional route is flexible but it does not optimize the common
case where we share all of the same things. Containers can potentially
optimize that case. So long as anyone can create a new container even
someone inside a container we do not loose flexibility.

To deal with networking there are currently a significant number of

variables with static storage duration. Making those variables global
and placing them in structures is neither as efficient as it could be
nor is it as maintainable as it should be. Other subsystems have
similar problems.

So if we put econtainer in struct thread_info and optimize it like we
do current, create an interface to variables similar to
DEFINE_PER_CPU, create a syscall interface similar to clone, and
show that by doing this we don't loose flexibility, then it looks like
a good idea.

If we can't do better than the current clone model of shared resources
then this model feels like gratuitous change.

Eric