Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 14:06:50 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>The important properties of the proposed container implementation:
>>>- each container has unique ID in the system
>> What namespace does this ID live in?
> global namespace. can be virtualized later.
> can be optional.

So a totally new kind of number in the kernel, that doesn't fall
in any of the traditional namespaces. That is something I would
like to avoid.

> But the idea is simple. Eventually you will need some management tools
> anyway. And they should be able to refer to containers.

In my implementation I refer to things indirectly via the task
in it since I have an appropriate pid. As does the current
kernel code for the filesystem mount namespace.

So far except a tool to create my things I don't need special
management tools the traditional unix ones suffice as I am
simply extending the traditional unix model.

The best counter argument I have is that filesystem namespaces
have already been implemented and we have not needed an additional id
in the kernel.

>>>- each process in the kernel can belong to one container only
>> Reasonable.
>>
>>>- effective container pointer (econtainer()) is used on the task to avoid
>>>insertion of additional argument "container" to all functions where it is
>>>required.
>> Why is that desirable?

Ok. I can finally see the effective thing. Writing an argument onto
the task structure instead of onto the stack seems weird to me.

> It was discussed with Linus and the reason is provided in this text actually.
> There are 2 ways:
> - to add additional argument "container" to all the functions where it is
> required.
> Drawbacks:

> a) lot's of changes,

I believe most of those changes make other places in the kernel where misuse happens easier to find. Turning places that need changes into compile errors seems a virtue to me. I do know there are places that store identifiers that might be used in a different context later on. Finding all of those places seems more important than generating compile errors for a few simple cases.

> b) compilation without virtualization is not the same.

Which is a reasonable consideration. But I expect most common kernels will ship with virtualization enabled so that is the important case to get correct.

> c) increased stakc usage

I think we are talking a few extra pointers on the stack. Our call nesting depth is not that great. So I would be surprised if the additional stack usage would be significant. I expect the gains from the locality of having everything in the same cache line are greater, than the gains from the reduced stack usage.

> - to add effective container pointer on the task. i.e. context which kernel
> should be in when works with virtualized resources.
> Drawbacks: a) there are some places where you need to change effective container
> context explicitly such as TCP/IP.

Yes.

Another is that it is easier to be aware and to think about what context you are working in if it is explicit. I expect other kernel developers are more likely to get it right if what is going is not hidden from them.

>>>- kernel compilation with disabled virtualization should result in old good
>>>linux kernel
>> A reasonable goal.
>> Why do we need a container structure to hold pointers to other pointers?
> can't catch what you mean :) is it prohibited somehow? :))))

No I just think an additional structure is unnecessary. Possibly it is a reasonable thing to do.

>> May I please be added to the CC list.
>> We are never going to form a consensus if all of the people doing
> implementations don't
>> talk.

> To make a consensus people need to make mutual concessions... Otherwise these
> talks are useless.

Consensus does not require concessions. We are after technical
excellence after all.

Yes people have to be willing to bend to work together to find the
best solution. I am willing. At the same time I am not easy to
convince that other solutions are necessarily better. To understand
which is better we need to understand why each of us made the
decisions we have made or hold the opinion we hold.

Once our reasons for doing things are clear and we can agree to a common
set of goals, picking the best solution should not be too difficult.

My problem in previous talks is that I don't think you have seen where I have
been coming from. Which is why I shut up earlier and posted code.

Eric
