Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Wed, 08 Feb 2006 19:02:30 GMT
View Forum Message <> Reply to Message

On Wed, Feb 08, 2006 at 10:57:24AM -0500, Hubertus Franke wrote:
>Kirill Korotaev wrote:
>>>>Eric W. Biederman wrote:
>>>>So it seems the clone(flags) is a reasonable approach to create new
>>>>namespaces. Question is what is the initial state of each namespace?
>>>>In pidspace we know we should be creating an empty pidmap !
>>>>In network, someone suggested creating a loopback device
>>>>In uts, create "localhost"
>>>>Are there examples where we rather inherit ? Filesystem ?
>>>
>>>Of course filesystem is already implemented, and does inheret a full
>>>copy.

I try to comment on both mails here because I thing that
clone() is basically a good interface, but will require
some redesign and/or extension ...

>> why do we want to use clone()?

because it is a natural and existing interface for this purpose
at least in Linux-VServer it would work (to some extend). why?

because we aleady use tools like chbind and chcontext which
do similar things as chroot, and chroot could, in theory, use
clone() and rbind to do it's job ...

>> Just because of its name and flags?

extending the flags seems natural to me, but the problem might
actually be that there are not enough of them left

>> I think it is really strange to fork() to create network context.

if you look at it as network namespace and sharing existing
spaces and/or creating new ones, then clone() and unshare()
make pretty much sense there ...

>> What has process creation has to do with it?

it is a natural interface where you can decide whether to
share a space or acquire a new one ... IMHO it would make
sense to get trivial userspace tools to create those
new spaces in one go, so that the user can use those

'building blocks' to create new spaces whenever she needs

>> After all these clone()'s are called, some management actions
>> from host system are still required, to add these IPs/routings/etc.

not necessarily, for example Linux-VServer uses some kind
of 'priviledged' mode, in which the initial guest process
can modify various things (like in this case the networking)
and setup whatever is required, then, shortly after, giving
up those priviledges ...

>> So? Why mess it up?
>> Why not create a separate clean interface for container management?

I'm not against a clean interface at all, but how would
such a 'clean' interface look like?

- a complicated sysfs interface where you write strange
  values into even stranger places?
- 40 different syscalls to do stuff like adding or removing
  various parts from the spaces?
- a new ioctl for processes?

>> Kirill
>
> We need a "init" per container, which represents the context of the
> system represented by the container.

that's only partially true, for example Linux-VServer also
allows for light-weight guests/containers which do not
have a separate init process, just a 'fake' one, so we can
save the resources consumed by a separate init process ...

it turns out that this works perfectly fine, even without
that fake init, if you teach a few tools like pstree that
they should not blindly assume that there is a pid=1 :)

> If that is the case, then why not create the container such that
> we specify what namespaces need to be new for a container at the
> container creation time and initialize them to a well understood
> state that makes sense (e.g. copy namespace (FS, uts) , new fresh
> state (pid)).

agreed, but now comes the interesting part, how does such
a well understood state look like for all contexts?

obviously the name space makes a complete copy, leading to
various issues when you 'try' to get rid of the 'copied'

data, like host filesystem mount points and such ...

removing the mounts 'above' a certain chroot() path might
seem like a good solution here, but actually it will cause
issues when you want to maintain/access a guest/container
from the host/parent

leaving all mounts in place will require to either inherit
changes from the host/parent to all guests/containers, just
to avoid having e.g. /mnt/cdrom mounted in process A, which
does not even see it (as it's accessible space starts
somewhere else) and therefore being unable to eject the
thing, although it is obviously unused

> Then use the standard syscall to modify state (now "virtualized"
> through the task->xxx_namespace access).

works as long as you have a handle for that, and actually
you do not need one init per guest/container, you need
one 'uniquely identified' task 'outside' the container

which in the typical case already makes two of them, the
'handle' task outside and the 'init' task inside ...

> Do you see a need to change the namespace of a container after it
> has been created. I am not referring to the state of the namespace
> but truely moving to a completely different namespace after the
> container has been created.

container itself, probably not, task, definitely yes ...
i.e. you definitely want to move between all those spaces
given that you are sufficiently priviledged, which is a
completely different can of worms ...

> Obviously you seem to have some other usage in mind, beyond what my
> limited vision can see. Can you share some of those examples, because
> that would help this discussion along ...

I guess one case where the separate container setup is
desired is when you want to keep a container alive even
after the processes have seized to exist. for example to
visit a 'stopped' guest/context just to do some emergency
operations or install some files/packages/etc

IMHO this can easily be solved by keeping the 'handle'
process (or whatever handle may be used for the complete
context consisting of all spaces) around, even if no
process is using those spaces ...

the context as 'collection' of certain namespaces is definitely something which will be required to allow to 'move' into a specific container from the host/parent side, as the parent process obviously does not hold that information.

best,
Herbert

> Thanks a 10^6.
>
> -- Hubertus
>
>
>