Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Mon, 02 Apr 2007 14:27:27 GMT
View Forum Message <> Reply to Message

On Mon, Apr 02, 2007 at 09:09:39AM -0500, Serge E. Hallyn wrote:
> Losing the directory isn't a big deal though. And both unsharing a
> namespace (which causes a ns_container_clone) and mounting the hierarchy
> are done by userspace, so if for some installation it is a big deal,
> the init scripts on initrd can mount the hierarchy before doing any
> unsharing.

Yes I thought of that after posting this (that initrd can mount the
hierarchies), so this should not be an issue in practice ..

> Can you think of a reason why losing a few directories matters?

If we loose directories, then we don't have a way to manage the
task-group it represents thr' the filesystem interface, so I consider
that bad. As we agree, this will not be an issue if initrd
mounts the ns hierarchy atleast at bootup.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by ebiederm on Mon, 02 Apr 2007 18:02:35 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri <vatsa@in.ibm.com> writes:

> On Mon, Apr 02, 2007 at 09:09:39AM -0500, Serge E. Hallyn wrote:
>> Losing the directory isn't a big deal though. And both unsharing a
>> namespace (which causes a ns_container_clone) and mounting the hierarchy
>> are done by userspace, so if for some installation it is a big deal,
>> the init scripts on initrd can mount the hierarchy before doing any
>> unsharing.
>
> Yes I thought of that after posting this (that initrd can mount the
> hierarchies), so this should not be an issue in practice ..
>
>> Can you think of a reason why losing a few directories matters?
>
> If we loose directories, then we don't have a way to manage the
> task-group it represents thr' the filesystem interface, so I consider

> that bad. As we agree, this will not be an issue if initrd
> mounts the ns hierarchy atleast at bootup.

I suspect that could be a problem if we have recursive containers.
Even by having a separate mount namespace for isolation you really
don't want to share the mount. If you can see all of the processes
you do want to be able to see and control everything.

I guess I want to ask before this gets to far. Why are all of the
namespaces lumped into one group? I would think it would make much
more sense to treat each namespace individually (at least for the
user space interface).

Eric

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 14:09:24 GMT
View Forum Message <> Reply to Message

On Mon, Apr 02, 2007 at 12:02:35PM -0600, Eric W. Biederman wrote:
> > If we loose directories, then we don't have a way to manage the
> > task-group it represents thr' the filesystem interface, so I consider
> > that bad. As we agree, this will not be an issue if initrd
> > mounts the ns hierarchy atleast at bootup.
>
> I suspect that could be a problem if we have recursive containers.
> Even by having a separate mount namespace for isolation you really
> don't want to share the mount. If you can see all of the processes
> you do want to be able to see and control everything.

Won't there be some master (VFS) namespace which can see everything? The
idea would be then to list all containers in that namespace. I am
visualizing that a master namespace listing all containers like that
will be like a management console, from which you can monitor/control
resource consumption of all containers.

I agree the individual containers themselves should not be able to
mount and view other containers in this container/resource-control
filesystem. I presume existing VFS namespace mechanism would enforce
that restriction.

> I guess I want to ask before this gets to far. Why are all of the
> namespaces lumped into one group?

I don't think they are. From Serge's patches, a new group (or a directory in
container filesystem) is created everytime a new nsproxy is created

(copy_namespaces/sys_unshare).

> I would think it would make much
> more sense to treat each namespace individually (at least for the
> user space interface).

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by serue on Tue, 03 Apr 2007 15:32:20 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Srivatsa Vaddagiri <vatsa@in.ibm.com> writes:
>
> > On Mon, Apr 02, 2007 at 09:09:39AM -0500, Serge E. Hallyn wrote:
> >> Losing the directory isn't a big deal though. And both unsharing a
> >> namespace (which causes a ns_container_clone) and mounting the hierarchy
> >> are done by userspace, so if for some installation it is a big deal,
> >> the init scripts on initrd can mount the hierarchy before doing any
> >> unsharing.
> >
> > Yes I thought of that after posting this (that initrd can mount the
> > hierarchies), so this should not be an issue in practice ..
> >
> >> Can you think of a reason why losing a few directories matters?
> >
> > If we loose directories, then we don't have a way to manage the
> > task-group it represents thr' the filesystem interface, so I consider
> > that bad. As we agree, this will not be an issue if initrd
> > mounts the ns hierarchy atleast at bootup.
>
> I suspect that could be a problem if we have recursive containers.
> Even by having a separate mount namespace for isolation you really
> don't want to share the mount. If you can see all of the processes
> you do want to be able to see and control everything.
>
> I guess I want to ask before this gets to far. Why are all of the
> namespaces lumped into one group? I would think it would make much
> more sense to treat each namespace individually (at least for the
> user space interface).

You mean why is there one subsystem for all namespaces?

Convenience and no reason not to. What advantage would there be for
the userspace tools if we have them split up?

There's no real reason other than memory waste not to split them
up - vserver users could always just mount all namespace subsystems
under one hierarchy.

But frankly I don't know where we stand right now wrt the containers
patches. Do most people want to go with Vatsa's latest version moving
containers into nsproxy? Has any other development been going on?
Paul, have you made any updates?

-serge

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by serue on Tue, 03 Apr 2007 15:41:13 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Srivatsa Vaddagiri <vatsa@in.ibm.com> writes:
>
> > On Mon, Apr 02, 2007 at 09:09:39AM -0500, Serge E. Hallyn wrote:
> >> Losing the directory isn't a big deal though. And both unsharing a
> >> namespace (which causes a ns_container_clone) and mounting the hierarchy
> >> are done by userspace, so if for some installation it is a big deal,
> >> the init scripts on initrd can mount the hierarchy before doing any
> >> unsharing.
> >
> > Yes I thought of that after posting this (that initrd can mount the
> > hierarchies), so this should not be an issue in practice ..
> >
> >> Can you think of a reason why losing a few directories matters?
> >
> > If we loose directories, then we don't have a way to manage the
> > task-group it represents thr' the filesystem interface, so I consider
> > that bad. As we agree, this will not be an issue if initrd
> > mounts the ns hierarchy atleast at bootup.
>
> I suspect that could be a problem if we have recursive containers.
> Even by having a separate mount namespace for isolation you really
> don't want to share the mount. If you can see all of the processes
> you do want to be able to see and control everything.

Are you asking about what happens if initrd in a vserver asks to
 mount -t container -o ns,cpusets /containers
?

I suppose in that case it would make sense to allow a separate mount
instance with it's own superblock, with s_root pointing to the dentry
for the container the vserver is in?

> I guess I want to ask before this gets to far. Why are all of the
> namespaces lumped into one group? I would think it would make much
> more sense to treat each namespace individually (at least for the
> user space interface).
>
> Eric

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 15:45:37 GMT
View Forum Message <> Reply to Message

On 4/3/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> But frankly I don't know where we stand right now wrt the containers
> patches. Do most people want to go with Vatsa's latest version moving
> containers into nsproxy? Has any other development been going on?
> Paul, have you made any updates?

I've not made major changes since the last patch post, just some small
optimizations and fixes - I've been too tied up with other stuff.

Whilst I've got no objection in general to using nsproxy rather than
the container_group object that I introduced in my latest patches, I
think that Vatsa's approach of losing the general container object is
flawed, since it loses any kind of per-group generic state (e.g. "this
container is being deleted") and last time I saw it, I think it would
tend to lose processes so that they didn't show up in any directory in
the container fs.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by serge on Tue, 03 Apr 2007 15:54:14 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On 4/3/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> >

> >But frankly I don't know where we stand right now wrt the containers
> >patches. Do most people want to go with Vatsa's latest version moving
> >containers into nsproxy? Has any other development been going on?
> >Paul, have you made any updates?
>
> I've not made major changes since the last patch post, just some small
> optimizations and fixes - I've been too tied up with other stuff.
>
> Whilst I've got no objection in general to using nsproxy rather than
> the container_group object that I introduced in my latest patches, I

Hmm, my largest objection had been that the nsproxy as a container
structure would end up pointing to nsproxy as a namespace proxy.

But if we do as Eric suggests and have one subsystem per namespace type,
rather than one subsystem for all namespaces, I guess that is no longer
a problem.

That still leaves yours.

> think that Vatsa's approach of losing the general container object is
> flawed, since it loses any kind of per-group generic state (e.g. "this
> container is being deleted") and last time I saw it, I think it would
> tend to lose processes so that they didn't show up in any directory in
> the container fs.
>
> Paul
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 16:03:43 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 10:32:20AM -0500, Serge E. Hallyn wrote:
> But frankly I don't know where we stand right now wrt the containers
> patches. Do most people want to go with Vatsa's latest version moving
> containers into nsproxy?

> Has any other development been going on?

I have another update to the rcfs patches more or less ready to go. I
hope to post them out this week.

Note that from user or controller perspective, there should be very
little difference between the two patches. rcfs is, I would say, same as
container patches to the extent of 70-80%. The rest of the difference is
mainly avoiding a new pointer in task_struct and new structures to
represent a task_group.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by dev on Tue, 03 Apr 2007 16:14:49 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 4/3/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
>>But frankly I don't know where we stand right now wrt the containers
>>patches. Do most people want to go with Vatsa's latest version moving
>>containers into nsproxy? Has any other development been going on?
>>Paul, have you made any updates?
>
>
> I've not made major changes since the last patch post, just some small
> optimizations and fixes - I've been too tied up with other stuff.
>
> Whilst I've got no objection in general to using nsproxy rather than
> the container_group object that I introduced in my latest patches, I
> think that Vatsa's approach of losing the general container object is
> flawed, since it loses any kind of per-group generic state (e.g. "this
> container is being deleted") and last time I saw it, I think it would
> tend to lose processes so that they didn't show up in any directory in
> the container fs.

Sounds reasonable.
Pavel is preparing new RSS patches on top of your patches
which take into account other comments and Andrew objections.
Can we cooperate to send it altogher then?

Thanks,
Kirill

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to

nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 16:16:04 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 08:45:37AM -0700, Paul Menage wrote:
> Whilst I've got no objection in general to using nsproxy rather than
> the container_group object that I introduced in my latest patches, I
> think that Vatsa's approach of losing the general container object is
> flawed, since it loses any kind of per-group generic state (e.g. "this
> container is being deleted")

I think I should post the updated version what I have soon. It handles
the "container/task group being deleted" case well, by setting
task/ns_proxy->count = 0. The only problem I need to resolve is
notify_on_release support (and also I haven't looked at the resource conters
patches yet).

> and last time I saw it, I think it would
> tend to lose processes so that they didn't show up in any directory in
> the container fs.

I think these are fixed in the latest version I have. Will send out to
you later this week (as soon as I drag myself off a higher prio task!).

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 16:19:28 GMT
View Forum Message <> Reply to Message

On 4/3/07, Kirill Korotaev <dev@sw.ru> wrote:
>
> Sounds reasonable.
> Pavel is preparing new RSS patches on top of your patches
> which take into account other comments and Andrew objections.
> Can we cooperate to send it altogher then?

Sure. Do you want to send me any core container changes that you have?

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 16:39:12 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 08:45:37AM -0700, Paul Menage wrote:
> Whilst I've got no objection in general to using nsproxy rather than
> the container_group object that I introduced in my latest patches,

So are you saying lets (re-)use tsk->nsproxy but also retain 'struct container' to store general per-group state? If so, I think that would address my main concern of redundant/avoidable new pointers in task_struct introduced in the container patches ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 16:52:35 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Tue, Apr 03, 2007 at 08:45:37AM -0700, Paul Menage wrote:
> > Whilst I've got no objection in general to using nsproxy rather than
> > the container_group object that I introduced in my latest patches,
>
> So are you saying lets (re-)use tsk->nsproxy but also retain 'struct
> container' to store general per-group state? If so, I think that would
> address my main concern of redundant/avoidable new pointers in
> task_struct introduced in the container patches ..

I'm not saying "let's use nsproxy" - I'm not yet convinced that the lifetime/mutation/correlation rate of a pointer in an nsproxy is likely to be the same as for a container subsystem; if not, then reusing nsproxy could actually increase space overheads (since you'd end up with more, larger nsproxy objects, compared to smaller numbers of smaller nsproxy objects and smaller numbers of smaller container_group objects), even though it saved (just) one pointer per task_struct.

Reusing nsproxy versus using a separate container_group object as the aggregator is mostly an implementation detail, i think, and it would be pretty easy to switch from one to the other without any user-visible changes. So I'm inclined to keep them separate at this

point.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 17:06:22 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 09:52:35AM -0700, Paul Menage wrote:
> I'm not saying "let's use nsproxy" - I'm not yet convinced that the
> lifetime/mutation/correlation rate of a pointer in an nsproxy is
> likely to be the same as for a container subsystem; if not, then
> reusing nsproxy could actually increase space overheads (since you'd
> end up with more, larger nsproxy objects, compared to smaller numbers
> of smaller nsproxy objects and smaller numbers of smaller
> container_group objects), even though it saved (just) one pointer per
> task_struct.

Even if nsproxy objects are made larger a bit, the number of such object will
be -much- lesser compared to number of task_structs I would think, so
the win/lose in space savings would need to take that into account.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 17:10:35 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Tue, Apr 03, 2007 at 09:52:35AM -0700, Paul Menage wrote:
> > I'm not saying "let's use nsproxy" - I'm not yet convinced that the
> > lifetime/mutation/correlation rate of a pointer in an nsproxy is
> > likely to be the same as for a container subsystem; if not, then
> > reusing nsproxy could actually increase space overheads (since you'd
> > end up with more, larger nsproxy objects, compared to smaller numbers
> > of smaller nsproxy objects and smaller numbers of smaller
> > container_group objects), even though it saved (just) one pointer per
> > task_struct.
>
> Even if nsproxy objects are made larger a bit, the number of such object will

You're not making them "a bit" larger, you're adding N+M pointers
where N is the number of container hierarchies and M is the number of
subsystem slots.

Basically, it means that anyone that uses containers without
namespaces or vice versa ends up paying the space overheads for both.

> be -much- lesser compared to number of task_structs I would think, so
> the win/lose in space savings would need to take that into account.

Agreed. So I'm not saying it's fundamentally a bad idea - just that
merging container_group and nsproxy is a fairly simple space
optimization that could easily be done later.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 17:24:32 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 10:10:35AM -0700, Paul Menage wrote:
> > Even if nsproxy objects are made larger a bit, the number of such object will
>
> You're not making them "a bit" larger, you're adding N+M pointers
> where N is the number of container hierarchies and M is the number of
> subsystem slots.

Hmm no .. I currently have nsproxy having just M additional pointers, where
M is the maximum number of resource controllers and a single dentry
pointer. As tasks get moved around in different hierarchies, new nsproxy's get
created to point to proper objects in those hierarchies (which can be done with
just M pointers in nsproxy). I have not considered the case of "search/reuse
nsproxy that has the combination you want" just to keep it simple, but
we should at some point in time do that.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 17:27:54 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 10:10:35AM -0700, Paul Menage wrote:
> Agreed. So I'm not saying it's fundamentally a bad idea - just that
> merging container_group and nsproxy is a fairly simple space
> optimization that could easily be done later.

IMHO, if we agree that space optimization is important, then its better
we tackle it right at design phase, rather than ripping it out later ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 17:29:01 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Tue, Apr 03, 2007 at 10:10:35AM -0700, Paul Menage wrote:
> > Agreed. So I'm not saying it's fundamentally a bad idea - just that
> > merging container_group and nsproxy is a fairly simple space
> > optimization that could easily be done later.
>
> IMHO, if we agree that space optimization is important, then its better
> we tackle it right at design phase, rather than ripping it out later ..
>

"Premature optimization is the root of all evil".

I think we're in agreement that there's value in having an aggregator
to avoid every task having to have a forest of pointers. But it
wouldn't be a major design change to switch that aggregator from being
container_group to being nsproxy at some later point.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 17:30:28 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
>
> Hmm no .. I currently have nsproxy having just M additional pointers, where
> M is the maximum number of resource controllers and a single dentry

> pointer.

So how do you implement something like the /proc/<PID>/container info
file in my patches? (Or more generally, tell which container a task is
in for a given hierarchy?)

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Tue, 03 Apr 2007 17:43:47 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 10:30:28AM -0700, Paul Menage wrote:
> So how do you implement something like the /proc/<PID>/container info
> file in my patches?

I havent implemented that yet, so I will look at your next question:

> (Or more generally, tell which container a task is
> in for a given hierarchy?)

Why is the hierarchy bit important here? Usually controllers need to
know "tell me what cpuset this task belongs to", which is answered
by tsk->nsproxy->ctlr_data[CPUSET_ID].

The only usecase of storing hierarchy I have found is in container_clone()
where you need to create a directory in the right hierarchy, so knowing
the hierarchy to which a controller is bound becomes important.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Tue, 03 Apr 2007 17:49:49 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> > (Or more generally, tell which container a task is
> > in for a given hierarchy?)
>
> Why is the hierarchy bit important here? Usually controllers need to
> know "tell me what cpuset this task belongs to", which is answered

> by tsk->nsproxy->ctlr_data[CPUSET_ID].

I was thinking of queries from userspace.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Wed, 04 Apr 2007 03:00:46 GMT
View Forum Message <> Reply to Message

On Tue, Apr 03, 2007 at 10:49:49AM -0700, Paul Menage wrote:
> > Why is the hierarchy bit important here? Usually controllers need to
> > know "tell me what cpuset this task belongs to", which is answered
> > by tsk->nsproxy->ctlr_data[CPUSET_ID].
>
> I was thinking of queries from userspace.

User space queries like "what is the cpuset to which this task belongs",
where the answer needs to be something of the form "/dev/cpuset/C1"?
The patches address that requirement atm by having a dentry pointer in
struct cpuset itself.

Do you see similar queries coming in for every resource controller object
(show me the path of cpu_acct, cpu_ctl, rss_ctl ... objects to which this
task belongs)? IMO that will not be the case, in which case we can avoid
adding N pointers (N = max hierarchies) in nsproxy just to support queries of
those sort.

If additional resource objects need to support such queries in future, we could
add a dentry pointer in those objects as and when the need arises.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Jackson on Wed, 04 Apr 2007 03:44:02 GMT
View Forum Message <> Reply to Message

vatsa wrote:
> User space queries like "what is the cpuset to which this task belongs",
> where the answer needs to be something of the form "/dev/cpuset/C1"?

I think that answer should be of the form "/C1", and not include the
cpuset file system mount point ... though for the purposes of the
present thread, this is probably not an interesting detail.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Wed, 04 Apr 2007 04:04:59 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> User space queries like "what is the cpuset to which this task belongs",
> where the answer needs to be something of the form "/dev/cpuset/C1"?
> The patches address that requirement atm by having a dentry pointer in
> struct cpuset itself.

Have you posted the cpuset implementation over your system yet?

The drawback to that is that every subsystem has to add a dentry to
its state, and handle the processing.

>
> Do you see similar queries coming in for every resource controller object
> (show me the path of cpu_acct, cpu_ctl, rss_ctl ... objects to which this
> task belongs)? IMO that will not be the case, in which case we can avoid
> adding N pointers (N = max hierarchies) in nsproxy just to support queries of
> those sort.

OK, I see your argument that putting it in the aggregator probably
isn't the best thing to do from a space point of view in the case when
the number of aggregators

This seems like a place where my container_subsys_state object is
useful - it can store a pointer to the container object (and be
maintained by the generic container system), at a space cost of 1
pointer per subsystem grouping, rather than N pointers per aggregator.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem

On Tue, Apr 03, 2007 at 09:04:59PM -0700, Paul Menage wrote:
> Have you posted the cpuset implementation over your system yet?

Yep, here:

http://lists.linux-foundation.org/pipermail/containers/2007- March/001497.html

For some reason, the above mail didnt make it into lkml (maybe it exceeded the max size allowed). I also have a updated version of that which I hope to post as soon as I am done with something else I am working on (sigh ..)

> The drawback to that is that every subsystem has to add a dentry to
> its state, and handle the processing.

Again this depends on whether every subsystem need to be able to support the user-space query you pointed out.

> >Do you see similar queries coming in for every resource controller object
> >(show me the path of cpu_acct, cpu_ctl, rss_ctl ... objects to which this
> >task belongs)? IMO that will not be the case, in which case we can avoid
> >adding N pointers (N = max hierarchies) in nsproxy just to support queries
> >of
> >those sort.
>
> OK, I see your argument that putting it in the aggregator probably
> isn't the best thing to do from a space point of view in the case when
> the number of aggregators

Sorry that sentence seems to be garbled by some mail router :)

Did you mean to say "when the number of aggregators sharing the same container object are more" ?

I agree ..Putting N pointers in container_group object just to support queries isn't justified at this point, because we don't know whether all subsystems need to support such queries.

> This seems like a place where my container_subsys_state object is
> useful - it can store a pointer to the container object (and be
> maintained by the generic container system), at a space cost of 1
> pointer per subsystem grouping, rather than N pointers per aggregator.

Yes that would be better than having N pointers in aggregator. From supporting purely user-space query pov, I think that is roughly same as having a 'dentry pointer' per resource object (what I mentioned earlier).

IMO we should add a dentry/container_subsys_state pointer only for those
subsystems which need to support such queries ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Wed, 04 Apr 2007 07:00:07 GMT
View Forum Message <> Reply to Message

On 4/3/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Tue, Apr 03, 2007 at 09:04:59PM -0700, Paul Menage wrote:
> > Have you posted the cpuset implementation over your system yet?
>
> Yep, here:
>
> http://lists.linux-foundation.org/pipermail/containers/2007- March/001497.html
>
> For some reason, the above mail didnt make it into lkml (maybe it
> exceeded the max size allowed). I also have a updated version of that
> which I hope to post as soon as I am done with something else I am
> working on (sigh ..)

OK, looking at that, I see a few problems related to the use of
nsproxy and lack of a container object:

- your find_nsproxy() function can return an nsproxy object that's
correct in its set of resource controllers but not in its other
nsproxy pointers.

- rcfs_rmdir() checks the count on the dentry's nsproxy pointer. But
it doesn't check for any of the other nsproxy objects that tasks in
the same grouping in this hierarchy might have.

- rcfs_rmdir() sets ns->count to 0. But it doesn't stop anyone else
from picking up a reference to that nsproxy from the list. Which could
happen if someone else has opened the container's tasks file and is
trying to write into it (but is blocked on manage_mutex). You can
possibly get around this by completely freeing the namespace and
setting dentry->fsdata to NULL before you release manage_mutex (and
treat a NULL fsdata as a dead container).

- how do you handle additional reference counts on subsystems? E.g.

beancounters wants to be able to associate each file with the container that owns it. You need to be able to lock out subsystems from taking new reference counts on an unreferenced container that you're deleting, without making the refcount operation too heavyweight.

- I think there's value in being able to mount a containerfs with no attached subsystems, simply for secure task grouping (e.g. job tracking). My latest patch set didn't support that, but it's a trivial small change to allow it. How would you do that with no container-like object?

You could have a null subsystem that does nothing other than let you attach processes to it, but then you'd be limited to one such grouping. (Since a given subsystem can only be instantiated in one hierarchy).

>
> > The drawback to that is that every subsystem has to add a dentry to
> > its state, and handle the processing.
>
> Again this depends on whether every subsystem need to be able to support
> the user-space query you pointed out.

Well, if more than one wants to support it, it means duplicating code that could equally easily be generically provided.

>
> > >Do you see similar queries coming in for every resource controller object
> > >(show me the path of cpu_acct, cpu_ctl, rss_ctl ... objects to which this
> > >task belongs)? IMO that will not be the case, in which case we can avoid
> > >adding N pointers (N = max hierarchies) in nsproxy just to support queries
> > >of
> > >those sort.
> >
> > OK, I see your argument that putting it in the aggregator probably
> > isn't the best thing to do from a space point of view in the case when
> > the number of aggregators
>
> Sorry that sentence seems to be garbled by some mail router :)

Nope, it got garbled because I didn't proof-read my email sufficiently ...

>
> Did you mean to say "when the number of aggregators sharing the same
> container object are more" ?

Yes. Although having thought about the possibility of null groupings

that I described above, I'm no longer convinced that argument is
valid. It would depend a lot on how people used containers in practice
- whether the number of aggregators was very small (when all
subsystems are in one hierarchy, or in different hierarchies with
isomorphic groupings) or very large (when all subsystems are in
different hierarchies, with orthogonal groupings).

>
> I agree ..Putting N pointers in container_group object just to support
> queries isn't justified at this point, because we don't know whether all
> subsystems need to support such queries.

It's not just to support such queries - that's just one example. There
are definitely other uses to having the container pointer directly in
the aggregator, including those that I mentioned above.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by xemul on Wed, 04 Apr 2007 08:30:56 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 4/3/07, Kirill Korotaev <dev@sw.ru> wrote:
>>
>> Sounds reasonable.
>> Pavel is preparing new RSS patches on top of your patches
>> which take into account other comments and Andrew objections.
>> Can we cooperate to send it altogher then?
>
> Sure. Do you want to send me any core container changes that you have?

Actually I don't have any right now. Everything is built at
the top of 2.6.20 patches you sent.

Maybe you will send me (give an URL) the preliminary patches
for more fresh kernel and I'll check what is still needed
for RSS container?

I remind that I had only two problems with containers:
1. fork hook was called to early - before the new task
   initializes completely
2. early need for rss containers (earlier than initcalls
   are called)

> Paul

>

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Wed, 04 Apr 2007 17:19:35 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 12:00:07AM -0700, Paul Menage wrote:
> OK, looking at that, I see a few problems related to the use of
> nsproxy and lack of a container object:

Before we (and everyone else!) gets lost in this thread, let me say somethings upfront:

- From my understanding, task_struct is considered very pristine and any changes to it will need to have a good reason, from performance and/or memory footprint pov.

  Coming from that understanding, my main intention of doing the rcfs patch was to see if it makes technical sense in reusing the nsproxy pointer in task_struct for task-group based resource management purpose.

  The experience I have gained so far doing rcfs leads me to think that it is technically possible to reuse tsk->nsproxy for both task-group based resource management purpose, w/o breaking its existing user : containers (although rcfs patches that I have posted may not be the best way to exploit/reuse nsproxy pointer).

- If reusing nsproxy may not be a bad idea and is technically feasible for use in res mgmt, even if later down the lane, then IMHO it deserves some attention right at the begining, because it is so centric to task-grouping function and because we haven't merged anything yet.

That said, now onto some replies :)

> - your find_nsproxy() function can return an nsproxy object that's
> correct in its set of resource controllers but not in its other
> nsproxy pointers.

Very true. That's a bug and can be rectified. Atm however in my patch stack, I have dropped this whole find_nsproxy() and instead create a new nsproxy whenever tasks move ..Not the best I agree on long run.

> - rcfs_rmdir() checks the count on the dentry's nsproxy pointer. But
> it doesn't check for any of the other nsproxy objects that tasks in
> the same grouping in this hierarchy might have.

This is rectified in my latest stack. I do a rcfs_task_count() [same routine which is used in rcfs_tasks_open] to count tasks sharing a resource object attached to the task_proxy. Not the most optimal solution, but works and is probably ok if we consider rmdir to be infrequent operation. If we have a 'struct container' or some object to have shared state, as you mentioned in a diff thread, in addition to just reusing nsproxy, it can be made more optimal than that.

> - rcfs_rmdir() sets ns->count to 0. But it doesn't stop anyone else
> from picking up a reference to that nsproxy from the list. Which could
> happen if someone else has opened the container's tasks file and is
> trying to write into it (but is blocked on manage_mutex). You can
> possibly get around this by completely freeing the namespace and
> setting dentry->fsdata to NULL before you release manage_mutex (and
> treat a NULL fsdata as a dead container).

Isnt that handled already by the patch in rcfs_common_file_write()?

```
mutex_lock(&manage_mutex);

ns = __d_ns(file->f_dentry);
if (!atomic_read(&ns->count)) {
        retval = -ENODEV;
        goto out2;
}
```

> - how do you handle additional reference counts on subsystems? E.g.
> beancounters wants to be able to associate each file with the
> container that owns it. You need to be able to lock out subsystems
> from taking new reference counts on an unreferenced container that
> you're deleting, without making the refcount operation too
> heavyweight.

Firstly, this is not a unique problem introduced by using ->nsproxy. Secondly we have discussed this to some extent before (http://lkml.org/lkml/2007/2/13/122). Essentially if we see zero tasks sharing a resource object pointed to by ->nsproxy, then we can't be racing with a function like bc_file_charge(), which simplifies the problem quite a bit. In other words, seeing zero tasks in xxx_rmdir() after taking manage_mutex is permission to kill nsproxy and associated objects. Correct me if I am wrong here.

> - I think there's value in being able to mount a containerfs with no
> attached subsystems, simply for secure task grouping (e.g. job

> tracking). My latest patch set didn't support that, but it's a trivial
> small change to allow it. How would you do that with no container-like
> object?
>
> You could have a null subsystem that does nothing other than let you
> attach processes to it, but then you'd be limited to one such
> grouping. (Since a given subsystem can only be instantiated in one
> hierarchy).

Again note that I am not hell-bent on avoiding a container-like object
to store shared state of a group. My main desire was to avoid additional
pointer in task_struct and reuse nsproxy if possible. If the grand scheme of
things requires a 'struct container' object in addition to reusing ->nsproxy,
to store shared state like 'notify_on_release', 'hierarchical information'
then I have absolutely no objection.

Having said that, I don't know if there is practical use for what you
describe above.

> > > The drawback to that is that every subsystem has to add a dentry to
> > > its state, and handle the processing.
> >
> > Again this depends on whether every subsystem need to be able to support
> > the user-space query you pointed out.
>
> Well, if more than one wants to support it, it means duplicating code
> that could equally easily be generically provided.

Why would it mean duplicating code? A generic function which takes a
dentry pointer and returns its vfs path will avoid this code
duplication?

> > Did you mean to say "when the number of aggregators sharing the same
> > container object are more" ?
>
> Yes. Although having thought about the possibility of null groupings
> that I described above, I'm no longer convinced that argument is
> valid.

I think the null grouping as defined so far is very fuzzy ..How would
the kernel use this grouping, which would require reserving N pointers
in 'struct container_group'/'struct nsproxy'?

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Wed, 04 Apr 2007 17:35:07 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 10:56:43PM +0530, Srivatsa Vaddagiri wrote:
> I think the null grouping as defined so far is very fuzzy ..How would
> the kernel use this grouping, which would require reserving N pointers
> in 'struct container_group'/'struct nsproxy'?

aside from needing to support 'cat tasks' that is ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Wed, 04 Apr 2007 18:57:18 GMT
View Forum Message <> Reply to Message

On 4/4/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Wed, Apr 04, 2007 at 12:00:07AM -0700, Paul Menage wrote:
> > OK, looking at that, I see a few problems related to the use of
> > nsproxy and lack of a container object:
>
> Before we (and everyone else!) gets lost in this thread, let me say somethings
> upfront:
>
> - From my understanding, task_struct is considered very pristine and any
> changes to it will need to have a good reason, from performance
> and/or memory footprint pov.

Well, my latest set of patches replaced a struct cpuset pointer with a
struct container_group pointer - so no net increase there.

>
> Coming from that understanding, my main intention of doing the rcfs patch was
> to see if it makes technical sense in reusing the nsproxy pointer in
> task_struct for task-group based resource management purpose.
>
> The experience I have gained so far doing rcfs leads me to think that it is
> technically possible to reuse tsk->nsproxy for both task-group based
> resource management purpose, w/o breaking its existing user : containers
> (although rcfs patches that I have posted may not be the best way to
> exploit/reuse nsproxy pointer).

Agreed - I think that it is technically possible, with a bunch of
changes to nsproxy to support this.

>
> - If reusing nsproxy may not be a bad idea and is technically feasible
> for use in res mgmt, even if later down the lane, then IMHO it deserves some
> attention right at the begining, because it is so centric to task-grouping
> function and because we haven't merged anything yet.

Agreed, it's worth thinking about it - but reimplementing/renaming the
whole set of patches just to change container_group to nsproxy seems a
bit extreme.

I'd much rather get something that works with container_group, and
then work with the folks who have a lot invested in nsproxy to see
about merging the two.

>
> That said, now onto some replies :)
>
> > - your find_nsproxy() function can return an nsproxy object that's
> > correct in its set of resource controllers but not in its other
> > nsproxy pointers.
>
> Very true. That's a bug and can be rectified. Atm however in my patch
> stack, I have dropped this whole find_nsproxy() and instead create a new
> nsproxy whenever tasks move ..Not the best I agree on long run.

Or even short term, I think - although it won't hurt too much for
cases where a single process enters a container and all children stay
in the container, it'll hurt a lot in cases where you ever move the
contents of one container into another. (Possibly in an orthogonal
hierarchy).

BTW, what does rcfs do if a subsystem is registered when there are
already mounted hierarchies? It looks as though it leaves the relevant
pointers as NULLs.

>
> > - rcfs_rmdir() sets ns->count to 0. But it doesn't stop anyone else
> > from picking up a reference to that nsproxy from the list. Which could
> > happen if someone else has opened the container's tasks file and is
> > trying to write into it (but is blocked on manage_mutex). You can
> > possibly get around this by completely freeing the namespace and
> > setting dentry->fsdata to NULL before you release manage_mutex (and
> > treat a NULL fsdata as a dead container).
>
> Isnt that handled already by the patch in rcfs_common_file_write()?

>
>   mutex_lock(&manage_mutex);
>
>   ns = __d_ns(file->f_dentry);
>   if (!atomic_read(&ns->count)) {
>   retval = -ENODEV;
>   goto out2;
>   }

You're right, I missed that, sorry.

>
> Firstly, this is not a unique problem introduced by using ->nsproxy.

True - but it is one that I think needs to be addressed.

> Secondly we have discussed this to some extent before
> (http://lkml.org/lkml/2007/2/13/122). Essentially if we see zero tasks
> sharing a resource object pointed to by ->nsproxy, then we can't be
> racing with a function like bc_file_charge(), which simplifies the
> problem quite a bit. In other words, seeing zero tasks in xxx_rmdir()
> after taking manage_mutex is permission to kill nsproxy and associated
> objects. Correct me if I am wrong here.

I think you're only right if you're going to overload the nsproxy
count field as the total refcount, rather than the number of tasks. If
you do that then you risk having to allocate way more memory than you
need in any routine that tries to allocate an array with one pointer
per task in the container (e.g. when reading the tasks file, or moving
a task into a new cpuset). My patch separates out the refcounts from
the tasks counts for exactly that reason - there's a per-subsys_state
refcount which can be cheaply manipulated via the subsystem. (I have a
version with lower locking requirements that I've not yet posted).

>
> > - I think there's value in being able to mount a containerfs with no
> > attached subsystems, simply for secure task grouping (e.g. job
> > tracking). My latest patch set didn't support that, but it's a trivial
> > small change to allow it. How would you do that with no container-like
> > object?
> >
> > You could have a null subsystem that does nothing other than let you
> > attach processes to it, but then you'd be limited to one such
> > grouping. (Since a given subsystem can only be instantiated in one
> > hierarchy).
>
> Again note that I am not hell-bent on avoiding a container-like object
> to store shared state of a group. My main desire was to avoid additional

> pointer in task_struct and reuse nsproxy if possible. If the grand scheme of
> things requires a 'struct container' object in addition to reusing ->nsproxy,
> to store shared state like 'notify_on_release', 'hierarchical information'
> then I have absolutely no objection.

OK, but at that point you're basically back at my patches, but using
nsproxy rather than container_group.

>
> Why would it mean duplicating code? A generic function which takes a
> dentry pointer and returns its vfs path will avoid this code
> duplication?

It's still adding boilerplate (extra pointers, extra /proc files,
logic to hook them together) to every subsystem that needs this, that
could be avoided. But I guess this aspect of it isn't a huge issue.

>
> > > Did you mean to say "when the number of aggregators sharing the same
> > > container object are more" ?
> >
> > Yes. Although having thought about the possibility of null groupings
> > that I described above, I'm no longer convinced that argument is
> > valid.
>
> I think the null grouping as defined so far is very fuzzy ..How would
> the kernel use this grouping, which would require reserving N pointers
> in 'struct container_group'/'struct nsproxy'?

The kernel wouldn't use it, other than to act as an inescapable task
grouping whose members could always be easily listed from userspace.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by ebiederm on Wed, 04 Apr 2007 23:02:39 GMT
View Forum Message <> Reply to Message

Next time I have a moment I will try and take a closer look. However
currently these approaches feel like there is some unholy coupling going
on between different things.

In addition there appear to be some weird assumptions (an array with
one member per task_struct) in the group. The pid limit allows
us millions of task_structs if the user wants it. A several megabyte
array sounds like a completely unsuitable data structure. What

is wrong with just traversing task_list at least initially?

What happened to the unix philosophy of starting with simple and
correct designs?

Further we have several different questions that are all mixed up
in this thread.

- What functionality do we want to provide.
- How do we want to export that functionality to user space.

You can share code by having an embedded structure instead of a magic
subsystem things have to register with, and I expect that would be
preferable. Libraries almost always are easier to work with then
a subsystem with strict rules that doesn't give you choices.

Why do we need a subsystem id? Do we expect any controllers to
be provided as modules? I think the code is so in the core that
modules of any form are a questionable assumption.

.....
There is a real issue to be solved here that we can't add
controls/limits for a group of processes if we don't have a user space
interface for it.

Are the issues of building a user space interface so very hard,
and the locking so very nasty or is this a case of over engineering?

I'm inclined to the rcfs variant and using nsproxy (from what I have
seen in passing) because it is more of an exercise in minimalism, and I
am strongly inclined to be minimalistic. The straight cpuset
derivative seems to start with everything but the kitchen sink and
then add on to it. Which at first glance seems unhealthy.

Eric

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 01:35:28 GMT
View Forum Message <> Reply to Message

On 4/4/07, Eric W. Biederman <ebiederm@xmission.com> wrote:
>
> In addition there appear to be some weird assumptions (an array with
> one member per task_struct) in the group. The pid limit allows
> us millions of task_structs if the user wants it. A several megabyte
> array sounds like a completely unsuitable data structure. What

> is wrong with just traversing task_list at least initially?

The current code creates such arrays when it needs an atomic snapshot
of the set of tasks in the container (e.g. for reporting them to
userspace or updating the mempolicies of all the tasks in the case of
cpusets). It may be possible to do it by traversing tasklist and
dropping the lock to operate on each task where necessary - I'll take
a look at that.

>
> - What functionality do we want to provide.
> - How do we want to export that functionality to user space.

I'm working on a paper for OLS that sets out a lot of these issues -
I'll send you a draft copy hopefully tonight.

>
> You can share code by having an embedded structure instead of a magic
> subsystem things have to register with, and I expect that would be
> preferable. Libraries almost always are easier to work with then
> a subsystem with strict rules that doesn't give you choices.
>
> Why do we need a subsystem id? Do we expect any controllers to
> be provided as modules? I think the code is so in the core that
> modules of any form are a questionable assumption.

One of the things that I'd really like to be able to do is allow
userspace to mount one filesystem instance with several
controllers/subsystems attached to it, so that they can define a
single task->container mapping to be used for multiple different
subsystems.

I think that would be hard to do without a central registration of subsystems.

I'm not particularly expecting modules to register as subsystems,
since as you say they'd probably need to be somewhat embedded in the
kernel in order to operate. And there are definite performance
advantages to be had from allowing in-kernel subsystems to have a
constant subsystem id.

It would be nice if subsystems could optionally have a dynamic
subsystem id so that new subsystem patches didn't have to clash with
one another in some global enum, but maybe that's not worth worrying
about at this point.

>
> I'm inclined to the rcfs variant and using nsproxy (from what I have
> seen in passing) because it is more of an exercise in minimalism, and I

> am strongly inclined to be minimalistic.

There's not a whole lot of difference between the two patch sets,
other than the names, and whether they piggyback on nsproxy or on a
separate object (container_group) that could be merged with nsproxy
later if it seemed appropriate. Plus mine have the ability to support
subsystem callbacks on fork/exit, with no significant overhead in the
event that no configured subsystems actually want those callbacks.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 01:37:23 GMT
View Forum Message <> Reply to Message

On 4/4/07, Paul Menage <menage@google.com> wrote:
> The current code creates such arrays when it needs an atomic snapshot
> of the set of tasks in the container (e.g. for reporting them to
> userspace or updating the mempolicies of all the tasks in the case of
> cpusets). It may be possible to do it by traversing tasklist and
> dropping the lock to operate on each task where necessary - I'll take
> a look at that.

Just to clarify this - the cases that currently need an array of task
pointers *do* already traverse tasklist in order to locate those tasks
as needed - its when they want to be able to operate on those tasks
outside of the tasklist lock that the array is needed - lock
tasklist_lock, fill the array with tasks (with added refcounts), drop
tasklist_lock, do stuff.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 02:57:40 GMT
View Forum Message <> Reply to Message

On 4/4/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> > - how do you handle additional reference counts on subsystems? E.g.
> > beancounters wants to be able to associate each file with the
> > container that owns it. You need to be able to lock out subsystems
> > from taking new reference counts on an unreferenced container that
> > you're deleting, without making the refcount operation too
> > heavyweight.

>
> Firstly, this is not a unique problem introduced by using ->nsproxy.
> Secondly we have discussed this to some extent before
> (http://lkml.org/lkml/2007/2/13/122). Essentially if we see zero tasks
> sharing a resource object pointed to by ->nsproxy, then we can't be
> racing with a function like bc_file_charge(), which simplifies the
> problem quite a bit. In other words, seeing zero tasks in xxx_rmdir()
> after taking manage_mutex is permission to kill nsproxy and associated
> objects. Correct me if I am wrong here.
>

OK, I've managed to reconstruct my reasoning remembered why it's important to have the refcounts associated with the subsystems, and why the simple use of the nsproxy count doesn't work. Essentially, there's no way to figure out which underlying subsystem the refcount refers to:

1) Assume the system has a single task T, and two subsystems, A and B

2) Mount hierarchy H1, with subsystem A and root subsystem state A0, and hierarchy H2 with subsystem B and root subsystem state B0. Both H1/ and H2/ share a single nsproxy N0, with refcount 3 (including the reference from T), pointing at A0 and B0.

3) Create directory H1/foo, which creates subsystem state A1 (nsproxy N1, refcount 1, pointing at A1 and B0)

4) Create directory H2/bar, which creates subsystem state B1 (nsproxy N2, refcount 1, pointing at A0 and B1)

5) Move T into H1/foo/tasks and then H2/bar/tasks. It ends up with nsproxy N3, refcount 1, pointing at A1 and B1.

6) T creates an object that is charged to A1 and hence needs to take a reference on A1 in order to uncharge it later when it's released. So N3 now has a refcount of 2

7) Move T back to H1/tasks and H2/tasks; assume it picks up nsproxy N0 again; N3 has a refcount of 1 now. (Assume that the object created in step 6 isn't one that's practical/desirable to relocate when the task that created it moves to a different container)

In this particular case the extra refcount on N3 is intended to keep A1 alive (which prevents H1/foo being deleted), but there's no way to tell from the structures in use whether it was taken on A1 or on B1. Neither H1/foo nor H2/bar can be deleted, even though nothing is intending to have a reference count on H2/bar.

Putting the extra refcount explicitly either in A1, or else in a
container object associated with H1/foo makes this more obvious.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 06:32:41 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 07:57:40PM -0700, Paul Menage wrote:
> >Firstly, this is not a unique problem introduced by using ->nsproxy.
> >Secondly we have discussed this to some extent before
> >(http://lkml.org/lkml/2007/2/13/122). Essentially if we see zero tasks
> >sharing a resource object pointed to by ->nsproxy, then we can't be
> >racing with a function like bc_file_charge(), which simplifies the
> >problem quite a bit. In other words, seeing zero tasks in xxx_rmdir()
> >after taking manage_mutex is permission to kill nsproxy and associated
> >objects. Correct me if I am wrong here.

Let me clarify first that I wasn't proposing an extra ref count in
nsproxy to account for non-task references to a resource object pointed
to by nsproxy (say nsproxy->ctlr_data[BC_ID]). Refcounts needed
on beancounter because a non-task object is pointing to it (like struct
file) will be put in the beancounter itself.

What I did want to say was this (sorry about the verbose rant):

```
 mount -t container -obeancounter none /dev/bean
 mkdir /dev/bean/foo
 echo some_pid > /dev/bean/foo
```

Associated with foo is a beancounter object A1 which contains (among other
things) max files that can be opened by tasks in foo. Also upon
successful file open, file->f_bc will point to A1.

Now lets say that someone is doing

```
 rmdir /dev/bean/foo
```

while will lead us to xxx_rmdir() doing this:

```
 mutex_lock(&manage_mutex);

 count = rcfs_task_count(foo's dentry);
```

rcfs_task_count will essentially return number of tasks pointing to A1

thr' their nsproxy->ctlr_data[BC_ID].

IF (note that /if/ again) the count returned is zero, then my point was
we can destroy nsproxy behind foo and also B1, not worrying about a
'struct file' still pointing to B1. This stems from the fact that you
cannot have a task's file->f_bc pointing to B1 w/o the task itself
pointing to B1 also (task->nsproxy->ctlr_data[BC_ID] == B1). I also
assume f_bc will get migrated with its owner task across beancounters
(which seems reasonable to me atleast from 'struct file' context).

If there was indeed a file object still pointing to B1, then that can
only be true if rcfs_task_count() returns non-zero value. Correct?

This is what I had in mind when I said this above : "In other words, seeing
zero tasks in xxx_rmdir() after taking manage_mutex is permission to kill
nsproxy and associated objects".

OT : In your posting of beancounter patches on top of containers, f_bc
isnt being migrated upon task movements. Is that on intention?

> OK, I've managed to reconstruct my reasoning remembered why it's
> important to have the refcounts associated with the subsystems, and
> why the simple use of the nsproxy count doesn't work.

I didn't mean to have non-task objects add refcounts to nsproxy. See
above.

> 1) Assume the system has a single task T, and two subsystems, A and B
>
> 2) Mount hierarchy H1, with subsystem A and root subsystem state A0,
> and hierarchy H2 with subsystem B and root subsystem state B0. Both
> H1/ and H2/ share a single nsproxy N0, with refcount 3 (including the
> reference from T), pointing at A0 and B0.

Why refcount 3? I can only be 1 (from T) ..

> 3) Create directory H1/foo, which creates subsystem state A1 (nsproxy
> N1, refcount 1, pointing at A1 and B0)

right. At this point A1.count should be 1 (because N1 is pointing to it)

> 4) Create directory H2/bar, which creates subsystem state B1 (nsproxy
> N2, refcount 1, pointing at A0 and B1)

right. B1.count = 1 also.

> 5) Move T into H1/foo/tasks and then H2/bar/tasks. It ends up with
> nsproxy N3, refcount 1, pointing at A1 and B1.

right. A1.count = 2 (N1, N3) and B1.count = 2 (N2, N3)

> 6) T creates an object that is charged to A1 and hence needs to take a
> reference on A1 in order to uncharge it later when it's released. So
> N3 now has a refcount of 2

no ..N3 can continue to have 1 while A1.count becomes 3 (N1, N3 and
file->f_bc)

> 7) Move T back to H1/tasks and H2/tasks; assume it picks up nsproxy N0
> again; N3 has a refcount of 1 now. (Assume that the object created in
> step 6 isn't one that's practical/desirable to relocate when the task
> that created it moves to a different container)

The object was created by the task, so I would expect it should get
migrated too to the new task's context (which should be true in case of
f_bc atleast?). Can you give a practical example where you want to
migrate the task and not the object it created?

Anyway, coming down to the impact of all this for a nsproxy based
solution, I would imagine this is what will happen when T moves back to
H1/tasks and H2/tasks:

 - N3.count becomes zero
 - We invoke free_nsproxy(N3), which drops refcounts on
   all objects it is pointing to i.e

```
 free_nsproxy()
 {
  if (N3->mnt_ns)
   put_mnt_ns(N3->mnt_ns);
  ...
  if (N3->ctlr_data[BC_ID])
   put_bc(N3->ctlr_data[BC_ID]);
 }
```

put/get_bc() manages refcounts on beancounters. It will drop A1.count to 2
(if f_bc wasnt migrated) and not finding it zero will not destroy A1.

Essentially, in the nsproxy based approach, I am having individual
controllers maintain their own refcount mechanism (just like mnt_ns or
uts_ns are doing today).

> In this particular case the extra refcount on N3 is intended to keep
> A1 alive (which prevents H1/foo being deleted), but there's no way to
> tell from the structures in use whether it was taken on A1 or on B1.
> Neither H1/foo nor H2/bar can be deleted, even though nothing is

> intending to have a reference count on H2/bar.
>
> Putting the extra refcount explicitly either in A1, or else in a
> container object associated with H1/foo makes this more obvious.

Hope the above description resolves these points ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 06:39:06 GMT
View Forum Message <> Reply to Message

On Thu, Apr 05, 2007 at 12:09:50PM +0530, Srivatsa Vaddagiri wrote:
> IF (note that /if/ again) the count returned is zero, then my point was
> we can destroy nsproxy behind foo and also B1, not worrying about a
> 'struct file' still pointing to B1. This stems from the fact that you
> cannot have a task's file->f_bc pointing to B1 w/o the task itself
> pointing to B1 also (task->nsproxy->ctlr_data[BC_ID] == B1). I also
> assume f_bc will get migrated with its owner task across beancounters
> (which seems reasonable to me atleast from 'struct file' context).
>
> If there was indeed a file object still pointing to B1, then that can
> only be true if rcfs_task_count() returns non-zero value. Correct?

arghh .. s/B1/A1 pls

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 06:48:57 GMT
View Forum Message <> Reply to Message

On 4/4/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Wed, Apr 04, 2007 at 07:57:40PM -0700, Paul Menage wrote:
> > >Firstly, this is not a unique problem introduced by using ->nsproxy.
> > >Secondly we have discussed this to some extent before
> > >(http://lkml.org/lkml/2007/2/13/122). Essentially if we see zero tasks

> > >sharing a resource object pointed to by ->nsproxy, then we can't be
> > >racing with a function like bc_file_charge(), which simplifies the
> > >problem quite a bit. In other words, seeing zero tasks in xxx_rmdir()
> > >after taking manage_mutex is permission to kill nsproxy and associated
> > >objects. Correct me if I am wrong here.
>
> Let me clarify first that I wasn't proposing an extra ref count in
> nsproxy to account for non-task references to a resource object pointed
> to by nsproxy (say nsproxy->ctlr_data[BC_ID]). Refcounts needed
> on beancounter because a non-task object is pointing to it (like struct
> file) will be put in the beancounter itself.
>
> What I did want to say was this (sorry about the verbose rant):
>
> mount -t container -obeancounter none /dev/bean
> mkdir /dev/bean/foo
> echo some_pid > /dev/bean/foo
>
> Associated with foo is a beancounter object A1 which contains (among other
> things) max files that can be opened by tasks in foo. Also upon
> successful file open, file->f_bc will point to A1.
>
> Now lets say that someone is doing
>
> rmdir /dev/bean/foo
>
> while will lead us to xxx_rmdir() doing this:
>
> mutex_lock(&manage_mutex);
>
> count = rcfs_task_count(foo's dentry);
>
> rcfs_task_count will essentially return number of tasks pointing to A1
> thr' their nsproxy->ctlr_data[BC_ID].

One small issue with the (last posted) version of your patch is that
it doesn't take into account the refcounts from the directories
themselves - I think you probably need to subtract one for each active
subsystem.

>
> IF (note that /if/ again) the count returned is zero, then my point was
> we can destroy nsproxy behind foo and also B1, not worrying about a
> 'struct file' still pointing to B1. This stems from the fact that you
> cannot have a task's file->f_bc pointing to B1 w/o the task itself
> pointing to B1 also (task->nsproxy->ctlr_data[BC_ID] == B1). I also
> assume f_bc will get migrated with its owner task across beancounters
> (which seems reasonable to me atleast from 'struct file' context).

I don't think that's a reasonable assumption. A task can have
thousands of file handles open - having to scan and move every file
that the task has open would make a move operation incredibly
expensive. Additionally, tasks can share many of those file handles
with other tasks. So what happens if one task that has a file open
moves out of the container, but another stays behind? It's cleaner and
more efficient, and conceptually desirable, IMO, just to keep the file
associated with the container.

>
> OT : In your posting of beancounter patches on top of containers, f_bc
> isnt being migrated upon task movements. Is that on intention?

Yes.

>
> > OK, I've managed to reconstruct my reasoning remembered why it's
> > important to have the refcounts associated with the subsystems, and
> > why the simple use of the nsproxy count doesn't work.
>
> I didn't mean to have non-task objects add refcounts to nsproxy. See
> above.
>
> > 1) Assume the system has a single task T, and two subsystems, A and B
> >
> > 2) Mount hierarchy H1, with subsystem A and root subsystem state A0,
> > and hierarchy H2 with subsystem B and root subsystem state B0. Both
> > H1/ and H2/ share a single nsproxy N0, with refcount 3 (including the
> > reference from T), pointing at A0 and B0.
>
> Why refcount 3? I can only be 1 (from T) ..

Plus the refcounts from the two filesystem roots.

>
> > 3) Create directory H1/foo, which creates subsystem state A1 (nsproxy
> > N1, refcount 1, pointing at A1 and B0)
>
> right. At this point A1.count should be 1 (because N1 is pointing to it)
>
> > 4) Create directory H2/bar, which creates subsystem state B1 (nsproxy
> > N2, refcount 1, pointing at A0 and B1)
>
> right. B1.count = 1 also.
>
> > 5) Move T into H1/foo/tasks and then H2/bar/tasks. It ends up with
> > nsproxy N3, refcount 1, pointing at A1 and B1.

>
> right. A1.count = 2 (N1, N3) and B1.count = 2 (N2, N3)
>
> > 6) T creates an object that is charged to A1 and hence needs to take a
> > reference on A1 in order to uncharge it later when it's released. So
> > N3 now has a refcount of 2
>
> no ..N3 can continue to have 1 while A1.count becomes 3 (N1, N3 and
> file->f_bc)
>
> > 7) Move T back to H1/tasks and H2/tasks; assume it picks up nsproxy N0
> > again; N3 has a refcount of 1 now. (Assume that the object created in
> > step 6 isn't one that's practical/desirable to relocate when the task
> > that created it moves to a different container)
>
> The object was created by the task, so I would expect it should get
> migrated too to the new task's context (which should be true in case of
> f_bc atleast?). Can you give a practical example where you want to
> migrate the task and not the object it created?

I gave one above, for files; others could include pages (do you want
to have to migrate every page when a task switches container? what
about shared pages?)

Obviously this fundamental difference of opinion means that we're
going to end up disagreeing on whether the scenario I presented is a
problem or not ...

>
> Anyway, coming down to the impact of all this for a nsproxy based
> solution, I would imagine this is what will happen when T moves back to
> H1/tasks and H2/tasks:
>
> - N3.count becomes zero
> - We invoke free_nsproxy(N3), which drops refcounts on
> all objects it is pointing to i.e
>
> free_nsproxy()
> {
> if (N3->mnt_ns)
> put_mnt_ns(N3->mnt_ns);
> ...
> if (N3->ctlr_data[BC_ID])
> put_bc(N3->ctlr_data[BC_ID]);
> }
>
> put/get_bc() manages refcounts on beancounters. It will drop A1.count to 2
> (if f_bc wasnt migrated) and not finding it zero will not destroy A1.

>
> Essentially, in the nsproxy based approach, I am having individual
> controllers maintain their own refcount mechanism (just like mnt_ns or
> uts_ns are doing today).

The problem with that is that (given the assumption that some
subsystems might not want to migrate objects) you can then end up with
a subsystem state object that has refcounts on it from active objects
like files, but which is unreachable via any container filesystem
mechanism. Better IMO to be able to fail the rmdir() in that situation
so that the subsystem object remains accessible (so you can see where
the resources are being used up).

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 07:06:23 GMT
View Forum Message <> Reply to Message

On 4/4/07, Pavel Emelianov <xemul@sw.ru> wrote:
> Paul Menage wrote:
> > On 4/3/07, Kirill Korotaev <dev@sw.ru> wrote:
> >>
> >> Sounds reasonable.
> >> Pavel is preparing new RSS patches on top of your patches
> >> which take into account other comments and Andrew objections.
> >> Can we cooperate to send it altogher then?
> >
> > Sure. Do you want to send me any core container changes that you have?
>
> Actually I don't have any right now. Everything is built at
> the top of 2.6.20 patches you sent.
>
> Maybe you will send me (give an URL) the preliminary patches
> for more fresh kernel and I'll check what is still needed
> for RSS container?
>
> I remind that I had only two problems with containers:
> 1. fork hook was called to early - before the new task
>   initializes completely

OK, looking at this, it's not clear where the best place to call it
is. "As late as possible" sounds like a reasonable idea, but that
messes up the current support for the nsproxy container subsystem,
which wants to be able to move the current task into a new container
based on nsproxy unsharing, after we've added the new task to its

parent. I can imagine other subsystems maybe wanting to clone the
current container at fork time too.

That sort of implies that we need to split the container fork
mechanism up into two parts, one early to add the refcount to the
parent's container_group, and one late to handle the callbacks if
desired. But that should be pretty straightforward.

> 2. early need for rss containers (earlier than initcalls
>   are called)

Couldn't you copy the way cpuset_init_early() works?

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by xemul on Thu, 05 Apr 2007 07:18:59 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 4/4/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> Paul Menage wrote:
>> > On 4/3/07, Kirill Korotaev <dev@sw.ru> wrote:
>> >>
>> >> Sounds reasonable.
>> >> Pavel is preparing new RSS patches on top of your patches
>> >> which take into account other comments and Andrew objections.
>> >> Can we cooperate to send it altogher then?
>> >
>> > Sure. Do you want to send me any core container changes that you have?
>>
>> Actually I don't have any right now. Everything is built at
>> the top of 2.6.20 patches you sent.
>>
>> Maybe you will send me (give an URL) the preliminary patches
>> for more fresh kernel and I'll check what is still needed
>> for RSS container?
>>
>> I remind that I had only two problems with containers:
>> 1. fork hook was called to early - before the new task
>>   initializes completely
>

> OK, looking at this, it's not clear where the best place to call it
> is. "As late as possible" sounds like a reasonable idea, but that
> messes up the current support for the nsproxy container subsystem,
> which wants to be able to move the current task into a new container
> based on nsproxy unsharing, after we've added the new task to its
> parent. I can imagine other subsystems maybe wanting to clone the
> current container at fork time too.
>
> That sort of implies that we need to split the container fork
> mechanism up into two parts, one early to add the refcount to the
> parent's container_group, and one late to handle the callbacks if
> desired. But that should be pretty straightforward.

Splitting sounds good. I'll try to prepare an appropriate patch.

>> 2. early need for rss containers (earlier than initcalls
>> are called)
>
> Couldn't you copy the way cpuset_init_early() works?

I did this in my previous version of rss container,
but I think it can be generalized. This is initialization
code that must not be nice-looking, but if it can be it
should be.

> Paul
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 08:49:20 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 11:48:57PM -0700, Paul Menage wrote:
> >rcfs_task_count will essentially return number of tasks pointing to A1
> >thr' their nsproxy->ctlr_data[BC_ID].
>
> One small issue with the (last posted) version of your patch is that
> it doesn't take into account the refcounts from the directories
> themselves

You mean dentry->d_fsdata pointing to nsproxy should take a ref count on

nsproxy? afaics it is not needed as long as you first drop the dentry
before freeing associated nsproxy.

> - I think you probably need to subtract one for each active
> subsystem.

I don't understand this.

> I don't think that's a reasonable assumption. A task can have
> thousands of file handles open - having to scan and move every file
> that the task has open would make a move operation incredibly
> expensive.
>
> Additionally, tasks can share many of those file handles
> with other tasks. So what happens if one task that has a file open
> moves out of the container, but another stays behind? It's cleaner and
> more efficient, and conceptually desirable, IMO, just to keep the file
> associated with the container.

I don't have a authoritative view here on whether open file count should
be migrated or not, but from a layman perspective consider this:

- Task T1 is in Container C1, whose max open files can be 100
- T1 opens all of those 100 files
- T1 migrates to Container C2, but its open file count is not
  migrated
- T2 is migrated to container C1 and tries opening a file but is
  denied. T2 looks for "who is in my container who has opened all
  files" and doesn't find anyone.

Isn't that a bit abnormal from an end-user pov?

> >Why refcount 3? I can only be 1 (from T) ..
>
> Plus the refcounts from the two filesystem roots.

Filesystem root dentry's are special case. They will point to
init_nsproxy which is never deleted and hence they need not add
additional ref counts.

For other directories created, say H1/foo, foo's dentry will point to N1
but need not take additional refcount. N1 won't be deleted w/o dropping
foo's dentry first. I think this is very similar to cpuset case, where
dentry->d_fsdata = cs doesnt take additional ref counts on cpuset.

> >The object was created by the task, so I would expect it should get
> >migrated too to the new task's context (which should be true in case of
> >f_bc atleast?). Can you give a practical example where you want to

> >migrate the task and not the object it created?
>
> I gave one above, for files; others could include pages (do you want
> to have to migrate every page when a task switches container? what
> about shared pages?)
>
> Obviously this fundamental difference of opinion means that we're
> going to end up disagreeing on whether the scenario I presented is a
> problem or not ...

Again I am not a VM expert to say whether pages should get migrated or
not. But coming to the impact of this discussion on xxx_rmdir() ..

> The problem with that is that (given the assumption that some
> subsystems might not want to migrate objects) you can then end up with
> a subsystem state object that has refcounts on it from active objects
> like files, but which is unreachable via any container filesystem
> mechanism. Better IMO to be able to fail the rmdir() in that situation
> so that the subsystem object remains accessible (so you can see where
> the resources are being used up).

I agree we shouldn't delete a dir going by just the task count. How abt
a (optional) ->can_destroy callback which will return -EBUSY if additional
non-task objects are pointing to a subsyste's resource object?

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 09:29:32 GMT
View Forum Message <> Reply to Message

On 4/5/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
>
> You mean dentry->d_fsdata pointing to nsproxy should take a ref count on
> nsproxy? afaics it is not needed as long as you first drop the dentry
> before freeing associated nsproxy.

You get the nsproxy object from dup_namespaces(), which will give you
an nsproxy with a refcount of 1. So yes, your d_fsdata is already
holding a refcount.

>
> > - I think you probably need to subtract one for each active

> > subsystem.
>
> I don't understand this.

I meant for each active hierarchy, sorry. But thinking about this
further, I think you're right - since every container directory points
to an nsproxy that contains its own subsystems groups for subsystems
bound to that hierarchy, plus the root subsystem groups for subsystems
not bound to that hierarchy, no other container directory can have a
refcount on an nsproxy that matches this container directories bound
subsystem groups. So subtracting 1 as you do at the moment is fine. It
would be more of a problem if we were trying to rmdir the root
directories, but that's not an issue.

> I don't have a authoritative view here on whether open file count should
> be migrated or not, but from a layman perspective consider this:
>
> - Task T1 is in Container C1, whose max open files can be 100
> - T1 opens all of those 100 files
> - T1 migrates to Container C2, but its open file count is not
> migrated
> - T2 is migrated to container C1 and tries opening a file but is
> denied. T2 looks for "who is in my container who has opened all
> files" and doesn't find anyone.
>
> Isn't that a bit abnormal from an end-user pov?

Possibly. But isn't it equally abnormal if T1 opens a bunch of files,
forks, and its child is moved into a different container. Then C1 has
no open file count? I'm not sure that there's a universally right
answer to this, so we'd need to support both behaviours.

>
> > >Why refcount 3? I can only be 1 (from T) ..
> >
> > Plus the refcounts from the two filesystem roots.
>
> Filesystem root dentry's are special case. They will point to
> init_nsproxy which is never deleted and hence they need not add
> additional ref counts.

But you are taking an extra ref count - the call to
get_task_namespaces(&init_task).

> N1 won't be deleted w/o dropping
> foo's dentry first.

If the container directory were to have no refcount on the nsproxy, so the initial refcount was 0, then surely moving a task in and out of the container would push the refcount up to 1; moving it away would cause the nsproxy to be freed when you call put_nsproxy(oldns) at the end of attach_task(), since that would bring the refcount back down to 0. Similarly, the task exiting would have the same effect. But that's not what's happening, since you are taking a ref count.

> I think this is very similar to cpuset case, where
> dentry->d_fsdata = cs doesnt take additional ref counts on cpuset.

That's different - the refcount on a cpuset falling to 0 doesn't free the cpuset, it just makes it eligible to be freed by someone holding the manage_mutex. the refcount on an nsproxy falling to 0 (via put_nsproxy()) does cause the nsproxy to be freed).

> I agree we shouldn't delete a dir going by just the task count. How abt
> a (optional) ->can_destroy callback which will return -EBUSY if additional
> non-task objects are pointing to a subsyste's resource object?

Possibly - there are two choices:

1) expose a refcount to them directly, and just interrogate the refcount from the generic code to see if it's safe to delete the directory

2) have a can_destroy() callback with well defined semantics about when it's safe to take refcounts again - it's quite possible that one subsystem can return true from can_destroy() but others don't, in which case the subsystem can become active again.

My patches solve this with an exposed refcount; to take a refcount on a subsystem object that you don't already know to be still live, you atomically bump the refcount if it's not -1; if it is -1 you wait for it either to return to 0 or for the "dead" flag to be set by the generic code. (This is all handled by inline functions so the subsystem doesn't have to worry about the complexity).

If we were to say that if at any time the refcount and the task count are both zero, then the refcount isn't allowed to increment until the task count also increments, then it would probably be possible to simplify these semantics to just check that all the refcounts were zero. This probably isn't an unreasonable restriction to make. The analog of this for option 2 would be to require that if ever the task count is 0 and the resut of can_destroy() is true, the subsystem can't get into a state where can_destroy() would return false without the task count being incremented first (by a task moving into the container).

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 12:43:25 GMT
View Forum Message <> Reply to Message

On Thu, Apr 05, 2007 at 02:29:32AM -0700, Paul Menage wrote:
> >I don't understand this.
>
> I meant for each active hierarchy, sorry. But thinking about this
> further, I think you're right - since every container directory points
> to an nsproxy that contains its own subsystems groups for subsystems
> bound to that hierarchy, plus the root subsystem groups for subsystems
> not bound to that hierarchy, no other container directory can have a
> refcount on an nsproxy that matches this container directories bound
> subsystem groups.

Yep.

> So subtracting 1 as you do at the moment is fine. It
> would be more of a problem if we were trying to rmdir the root
> directories, but that's not an issue.
>
> >Isn't that a bit abnormal from an end-user pov?
>
> Possibly. But isn't it equally abnormal if T1 opens a bunch of files,
> forks, and its child is moved into a different container. Then C1 has
> no open file count?

Yep, that would be abnormal too. I think the issues are similar here wrt
shared pages too? So whatever is the consensus on shared page
accounting (is there one?), should apply here maybe ..

> >Filesystem root dentry's are special case. They will point to
> >init_nsproxy which is never deleted and hence they need not add
> >additional ref counts.
>
> But you are taking an extra ref count - the call to
> get_task_namespaces(&init_task).

/me looks at his current patch stack and doesnt find that anymore.

> If the container directory were to have no refcount on the nsproxy, so
> the initial refcount was 0,

No it should be 1.

```
 mkdir H1/foo
  rcfs_create()
     ns = dup_namespaces(parent);

      ....
     dentry->d_fsdata = ns;
```

ns should have a refcount of 1 to begin with.

> then surely moving a task in and out of
> the container would push the refcount up to 1;

it should push the recount to 2.

> moving it away would
> cause the nsproxy to be freed when you call put_nsproxy(oldns) at the
> end of attach_task(), since that would bring the refcount back down to
> 0.

It should bring the refcount back to 1 and hence put_nsproxy() shouldnt
be called.

> Similarly, the task exiting would have the same effect. But that's
> not what's happening, since you are taking a ref count.

Yes, dup_namespaces returns with a refcount of 1 (which should cover
dentry reference to it).

> >I think this is very similar to cpuset case, where
> >dentry->d_fsdata = cs doesnt take additional ref counts on cpuset.
>
> That's different - the refcount on a cpuset falling to 0 doesn't free
> the cpuset, it just makes it eligible to be freed by someone holding
> the manage_mutex. the refcount on an nsproxy falling to 0 (via
> put_nsproxy()) does cause the nsproxy to be freed).

>From what I described above:

- refcount of a nsproxy attached to a directory dentry can never
  fall to zero because of tasks coming in and out. The only
  way for the refcount of such nsproxies to fall to zero and
  hence trigger their destruction is thr' the rmdir i/f.

- New nsproxies derived from the base directory nsproxy
  can have their's refcount go to zero as tasks exit or move
  around and hence they will be destroyed.

Does that sound like correct behavior?

> Possibly - there are two choices:
>
> 1) expose a refcount to them directly, and just interrogate the
> refcount from the generic code to see if it's safe to delete the
> directory
>
> 2) have a can_destroy() callback with well defined semantics about
> when it's safe to take refcounts again - it's quite possible that one
> subsystem can return true from can_destroy() but others don't, in
> which case the subsystem can become active again.

Lets go back to the f_bc example here for a moment. Lets say T1 was in C1 and opened file f1. f1->f_bc points to C1->beancounter.

T1 moves from C1 -> C2, but f1 is not migrated.
C1->beancounter.count stays at 1 (to account for f1->f_bc).

File f1 is closed. C1->beancounter.count becomes zero.

Now user issues rmdir C1. If rmdir finds (after taking manage_mutex that is)

- zero tasks in C1
- zero refcount in C1->beancounter

why is it not safe to assume that C1->beancounter.count will continue to stay zero?

Basically I am struggling to answer "How can a zero refcount (beancounter) object go non-zero when zero tasks are attached to it" ..

If that is the case, ->can_destroy can simply return 0 if it finds ->beancounter.count == 0 else returns -EBUSY. Provided we had checked for zero tasks prior to calling ->can_destroy, it should be safe to assume no new ref counts will take place on ->beancounter.count ??

> My patches solve this with an exposed refcount; to take a refcount on
> a subsystem object that you don't already know to be still live, you
> atomically bump the refcount if it's not -1; if it is -1 you wait for
> it either to return to 0 or for the "dead" flag to be set by the
> generic code. (This is all handled by inline functions so the
> subsystem doesn't have to worry about the complexity).
>
> If we were to say that if at any time the refcount and the task count
> are both zero, then the refcount isn't allowed to increment until the

> task count also increments, then it would probably be possible to
> simplify these semantics to just check that all the refcounts were
> zero. This probably isn't an unreasonable restriction to make. The
> analog of this for option 2 would be to require that if ever the task
> count is 0 and the resut of can_destroy() is true, the subsystem can't
> get into a state where can_destroy() would return false without the
> task count being incremented first (by a task moving into the
> container).

Precisely. This is what I outlined above in the example. I personally
prefer ->can_destroy because it is simpler and only those subsystems
that have this problem need to support it. On a hierarchy where all
subsystems attached don't have a can_destroy callback, then rmdir just
collapses to a "zero task count check" ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 14:13:21 GMT
View Forum Message <> Reply to Message

On Thu, Apr 05, 2007 at 06:13:25PM +0530, Srivatsa Vaddagiri wrote:
> Lets go back to the f_bc example here for a moment. Lets say T1 was in C1 and
> opened file f1. f1->f_bc points to C1->beancounter.
>
> T1 moves from C1 -> C2, but f1 is not migrated.
> C1->beancounter.count stays at 1 (to account for f1->f_bc).

Actually C1->beancounter.count should be at 2 (C1->beancounter and
f1->f_bc are pointing to it).

> File f1 is closed. C1->beancounter.count becomes zero.

C1->beancounter.count should go to 1 ..

> Now user issues rmdir C1. If rmdir finds (after taking manage_mutex that
> is)
>
> - zero tasks in C1
> - zero refcount in C1->beancounter

s/zero refcount in C1->beancounter/exactly 1 refcount in C1->beancounter

> why is it not safe to assume that C1->beancounter.count will continue to

> stay zero?

s/zero/at one

> Basically I am struggling to answer "How can a zero refcount (beancounter)
> object go non-zero when zero tasks are attached to it" ..

s/zero/one and s/non-zero/>1

Essentially bc_subsys->can_attach(struct bean_counter *b) can return
-EBUSY if (atomic_read(&b->count) > 1) ..

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 14:13:37 GMT
View Forum Message <> Reply to Message

On 4/5/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> > If the container directory were to have no refcount on the nsproxy, so
> > the initial refcount was 0,
>
> No it should be 1.
>
> mkdir H1/foo
> rcfs_create()
> ns = dup_namespaces(parent);
>
> ....
> dentry->d_fsdata = ns;
>
> ns should have a refcount of 1 to begin with.

Right - that's my point, you're effectively passing the initial
refcount of the nsproxy to the container directory's d_fsdata
reference.

>
> - refcount of a nsproxy attached to a directory dentry can never
> fall to zero because of tasks coming in and out. The only

> way for the refcount of such nsproxies to fall to zero and
> hence trigger their destruction is thr' the rmdir i/f.
>
> - New nsproxies derived from the base directory nsproxy
> can have their's refcount go to zero as tasks exit or move
> around and hence they will be destroyed.
>
> Does that sound like correct behavior?

Sounds good.

>
> > Possibly - there are two choices:
> >
> > 1) expose a refcount to them directly, and just interrogate the
> > refcount from the generic code to see if it's safe to delete the
> > directory
> >
> > 2) have a can_destroy() callback with well defined semantics about
> > when it's safe to take refcounts again - it's quite possible that one
> > subsystem can return true from can_destroy() but others don't, in
> > which case the subsystem can become active again.
>
> Lets go back to the f_bc example here for a moment. Lets say T1 was in C1 and
> opened file f1. f1->f_bc points to C1->beancounter.
>
> T1 moves from C1 -> C2, but f1 is not migrated.
> C1->beancounter.count stays at 1 (to account for f1->f_bc).
>
> File f1 is closed. C1->beancounter.count becomes zero.
>
> Now user issues rmdir C1. If rmdir finds (after taking manage_mutex that
> is)
>
> - zero tasks in C1
> - zero refcount in C1->beancounter
>
> why is it not safe to assume that C1->beancounter.count will continue to
> stay zero?
>
> Basically I am struggling to answer "How can a zero refcount (beancounter)
> object go non-zero when zero tasks are attached to it" ..

In that case, I think you're fine. Your last posted patches didn't
provide a way to check for that, though, as far as I could see.

Paul

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 14:38:44 GMT
View Forum Message <> Reply to Message

On Thu, Apr 05, 2007 at 07:13:37AM -0700, Paul Menage wrote:
> > ns should have a refcount of 1 to begin with.
>
> Right - that's my point, you're effectively passing the initial
> refcount of the nsproxy to the container directory's d_fsdata
> reference.

sure ..

> > Basically I am struggling to answer "How can a zero refcount (beancounter)
> > object go non-zero when zero tasks are attached to it" ..
>
> In that case, I think you're fine. Your last posted patches didn't
> provide a way to check for that, though, as far as I could see.

Yes, the last patch I posted is very primitive considering all the
discussions we had ..For ex: attach_task() in the last patch has more
bugs [apart from the exit race I found recently with cpusets/containers]
Each time I stopped and stared at attach_task() I could find a bug ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 16:50:52 GMT
View Forum Message <> Reply to Message

On Wed, Apr 04, 2007 at 11:57:18AM -0700, Paul Menage wrote:
> > Very true. That's a bug and can be rectified. Atm however in my patch
> > stack, I have dropped this whole find_nsproxy() and instead create a new
> > nsproxy whenever tasks move ..Not the best I agree on long run.
>
> Or even short term, I think - although it won't hurt too much for
> cases where a single process enters a container and all children stay
> in the container, it'll hurt a lot in cases where you ever move the
> contents of one container into another. (Possibly in an orthogonal
> hierarchy).
>
> BTW, what does rcfs do if a subsystem is registered when there are
> already mounted hierarchies? It looks as though it leaves the relevant

> pointers as NULLs.

The approach I am on currently doesnt deal with dynamically loaded
modules ..Partly because it allows subsystem ids to be compile-time
decided and also init_nsproxy.ctlr_data[] can be initialised to default
values at compile time itself.

For ex: init_nsproxy.ctlr_data[CPUSET_ID] can be set to &top_cpuset
at compile time.

If there is demand later, I agree we need to address the shortcoming you
point out ..

> I think you're only right if you're going to overload the nsproxy
> count field as the total refcount, rather than the number of tasks. If
> you do that then you risk having to allocate way more memory than you
> need in any routine that tries to allocate an array with one pointer
> per task in the container (e.g. when reading the tasks file, or moving
> a task into a new cpuset). My patch separates out the refcounts from
> the tasks counts for exactly that reason - there's a per-subsys_state
> refcount which can be cheaply manipulated via the subsystem. (I have a
> version with lower locking requirements that I've not yet posted).

We have been thr' this in a diff thread. I trust we have discussed all
that need to be discussed here?

> > Again note that I am not hell-bent on avoiding a container-like object
> > to store shared state of a group. My main desire was to avoid additional
> > pointer in task_struct and reuse nsproxy if possible. If the grand scheme of
> > things requires a 'struct container' object in addition to reusing ->nsproxy,
> > to store shared state like 'notify_on_release', 'hierarchical information'
> > then I have absolutely no objection.
>
> OK, but at that point you're basically back at my patches, but using
> nsproxy rather than container_group.

Ok ..by posting rcfs patches, I didn't mean to introduce a "yours" and
"mine" rift ..honestly. In fact you would notice that they have your
(sole) copyright still on them! It took me just two days to convert over the
patches to use nsproxy and come up with the rcfs patches and obviously I
couldnt have done that without your excellent patches to start with.

So if there is is larger interest in using nsproxy at some point, I hope
they serve as some reference patches to do the conversion. I would go ahead
and post what I have now (which incorporates several bug fixes) which
could be used as you deem necessary at a later point if/when considering
moving to nsproxy.

> > Why would it mean duplicating code? A generic function which takes a
> > dentry pointer and returns its vfs path will avoid this code
> > duplication?
>
> It's still adding boilerplate (extra pointers, extra /proc files,
> logic to hook them together) to every subsystem that needs this, that
> could be avoided. But I guess this aspect of it isn't a huge issue.

I agree that is not a big issue now (considering there are larger things
to go after!).

> > > > Did you mean to say "when the number of aggregators sharing the same
> > > > container object are more" ?
> > >
> > > Yes. Although having thought about the possibility of null groupings
> > > that I described above, I'm no longer convinced that argument is
> > > valid.
> >
> > I think the null grouping as defined so far is very fuzzy ..How would
> > the kernel use this grouping, which would require reserving N pointers
> > in 'struct container_group'/'struct nsproxy'?
>
> The kernel wouldn't use it, other than to act as an inescapable task
> grouping whose members could always be easily listed from userspace.

I am still trying to come to terms with this null groupings and how they
would be used in real life.

- Can you list a real world use of it?

- If they are "inescapable" task groups, how does the first task enter
  such a group, using just the filesystem interface?

- If there is no real kernel use for such groups, can this be
  implemented in userspace (user ids, session ids etc)

- If userspace soln is not feasible, why would you want such null
  groupings in multiple hierarchies and not in one hierarchy?
  If having them in one hierarchy satisfies the requirements,
  then we could create empty cpusets at top level (or at an
  appropriate subcpuset level) and use them as null groups?
  By defining permissionss of the null-group cpuset directories,
    we could restrict movement of tasks across groups ?

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Srivatsa Vaddagiri on Thu, 05 Apr 2007 17:14:13 GMT
View Forum Message <> Reply to Message

On Thu, Apr 05, 2007 at 10:27:52PM +0530, Srivatsa Vaddagiri wrote:
> Ok ..by posting rcfs patches, I didn't mean to introduce a "yours" and
> "mine" rift ..honestly. In fact you would notice that they have your
> (sole) copyright still on them! It took me just two days to convert over the
> patches to use nsproxy and come up with the rcfs patches and obviously I
> couldnt have done that without your excellent patches to start with.

By doing rcfs, I have been able to do a much better review of your
patches than by just browsing thr' the code. I hope you would find usefull some
of the feedback I have sent so far on the container patches ..

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Fri, 06 Apr 2007 21:54:48 GMT
View Forum Message <> Reply to Message

On 4/5/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
>
> The approach I am on currently doesnt deal with dynamically loaded
> modules ..Partly because it allows subsystem ids to be compile-time
> decided

Yes, that part is definitely a good idea, since it removes one of the
potential performance complaints that people have compared to
hard-coded pointers in a structure.

I've reworked my patches to require subsystems to be declared at compile time.

> and also init_nsproxy.ctlr_data[] can be initialised to default
> values at compile time itself.

> Ok ..by posting rcfs patches, I didn't mean to introduce a "yours" and
> "mine" rift ..honestly. In fact you would notice that they have your
> (sole) copyright still on them! It took me just two days to convert over the
> patches to use nsproxy and come up with the rcfs patches and obviously I
> couldnt have done that without your excellent patches to start with.

OK, sorry if I came across as possessive :-) There are definitely some

great ideas in your patches, some of which I've incorporated in my
patches as you'll see when I send them out later this afternoon

>
> I am still trying to come to terms with this null groupings and how they
> would be used in real life.
>
> - Can you list a real world use of it?

As a simple job tracking mechanism, without any other implications.

>
> - If they are "inescapable" task groups, how does the first task enter
> such a group, using just the filesystem interface?

Root would be able to move tasks around between containers, as normal.

>
> - If there is no real kernel use for such groups, can this be
> implemented in userspace (user ids, session ids etc)

Not cleanly. (Multiple jobs with the same user, session ids can be
changed by the user). Currently in the job-control system I'm working
on here, I was tagging any processes introduced in a job with a
job-unique extra gid, so we could identify which job a process was in
by looking at its group list. But that's a bit ugly.

In a more modern kernel we can just use cpusets without bothering to
make distinctions between the memory and cpus in different cpusets,
but it seems ugly to have to use a more heavyweight solution than you
really need.

In practice, this would be more of a toy/example, since anyone doing
job control probably is interested in at least some rudimentary kind
of resource tracking/control.

Paul