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
