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