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