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