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