Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by ebiederm on Wed, 04 Apr 2007 23:02:39 GMT
View Forum Message <> Reply to Message

Next time I have a moment I will try and take a closer look. However
currently these approaches feel like there is some unholy coupling going
on between different things.

In addition there appear to be some weird assumptions (an array with
one member per task_struct) in the group. The pid limit allows
us millions of task_structs if the user wants it. A several megabyte
array sounds like a completely unsuitable data structure. What
is wrong with just traversing task_list at least initially?

What happened to the unix philosophy of starting with simple and
correct designs?

Further we have several different questions that are all mixed up
in this thread.

- What functionality do we want to provide.
- How do we want to export that functionality to user space.

You can share code by having an embedded structure instead of a magic
subsystem things have to register with, and I expect that would be
preferable. Libraries almost always are easier to work with then
a subsystem with strict rules that doesn't give you choices.

Why do we need a subsystem id? Do we expect any controllers to
be provided as modules? I think the code is so in the core that
modules of any form are a questionable assumption.

.....
There is a real issue to be solved here that we can't add
controls/limits for a group of processes if we don't have a user space
interface for it.

Are the issues of building a user space interface so very hard,
and the locking so very nasty or is this a case of over engineering?

I'm inclined to the rcfs variant and using nsproxy (from what I have
seen in passing) because it is more of an exercise in minimalism, and I
am strongly inclined to be minimalistic. The straight cpuset
derivative seems to start with everything but the kitchen sink and
then add on to it. Which at first glance seems unhealthy.

Eric