Subject: Re: [ckrm-tech] [PATCH 7/7] containers (V7): Container interface to nsproxy subsystem
Posted by Paul Menage on Thu, 05 Apr 2007 01:35:28 GMT
View Forum Message <> Reply to Message

On 4/4/07, Eric W. Biederman <ebiederm@xmission.com> wrote:
>
> 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?

The current code creates such arrays when it needs an atomic snapshot of the set of tasks in the container (e.g. for reporting them to userspace or updating the mempolicies of all the tasks in the case of cpusets). It may be possible to do it by traversing tasklist and dropping the lock to operate on each task where necessary - I'll take a look at that.

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

I'm working on a paper for OLS that sets out a lot of these issues - I'll send you a draft copy hopefully tonight.

>
> 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.

One of the things that I'd really like to be able to do is allow userspace to mount one filesystem instance with several controllers/subsystems attached to it, so that they can define a single task->container mapping to be used for multiple different subsystems.

I think that would be hard to do without a central registration of subsystems.

I'm not particularly expecting modules to register as subsystems, since as you say they'd probably need to be somewhat embedded in the kernel in order to operate. And there are definite performance

advantages to be had from allowing in-kernel subsystems to have a constant subsystem id.

It would be nice if subsystems could optionally have a dynamic subsystem id so that new subsystem patches didn't have to clash with one another in some global enum, but maybe that's not worth worrying about at this point.

>
> 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.

There's not a whole lot of difference between the two patch sets, other than the names, and whether they piggyback on nsproxy or on a separate object (container_group) that could be merged with nsproxy later if it seemed appropriate. Plus mine have the ability to support subsystem callbacks on fork/exit, with no significant overhead in the event that no configured subsystems actually want those callbacks.

Paul