Subject: Re: [RFC] libcg: design and plans
Posted by Paul Menage on Wed, 05 Mar 2008 11:51:01 GMT
View Forum Message <> Reply to Message

On Wed, Mar 5, 2008 at 3:07 AM, Dhaval Giani <dhaval@linux.vnet.ibm.com> wrote:
>
> OK. Hmm, I've not really thought about it. At first thought, it should
> not be very difficult. Only thing I am not sure is the arbitrary
> grouping of the groups (ok, a bit confusing).

I suspect that the main form of composite grouping is going to be
between parents and children. E.g. you might want to say things like:

create_group(A, memory=1G, cpu=100)
create_group(B, parent=A, memory=inherit, cpu=20)
create_group(C, parent=A, memory=inherit, cpu=30)

i.e. both B and C inherit/share their memory limit from their parent,
but have their own CPU groups (child groups of their parent?)

So this would result in a single group A in the memory hierarchy and a
top-level group A and child groups B and C in the cpu hierarchy. libcg
would abstract away the fact that when you moved a process into an
abstract group, it actually had to be moved into multiple real groups.

I think this kind of sharing is fairly easy to specify. Now, there's
no reason that it shouldn't support more complex group sharing as
well, but that might require the user to use lower-level operations,
such as creating resource groups in particular hierarchies, and
associating abstract groups with those resource groups.

The model above (children sharing resource groups with their parents
for some resources) is actually something that I figured could be
supported relatively straightforwardly in the kernel - essentially:

- each subsystem "foo" would have a foo.inherit file provided by
cgroups in each group directory

- setting the foo.inherit flag (i.e. writing 1 to it) would cause
tasks in that cgroup to share the "foo" subsystem state with the
parent cgroup

- from the subsystem's point of view, it would only need to worry
about its own foo_cgroup objects and which task was associated with
each object; the subsystem wouldn't need to care about which tasks
were part of each cgroup, and which cgroups were sharing state; that
would all be taken care of by the cgroup framework

I'd sketched out a fairly nice design for how it would all work in my head when I realised that it could actually all be done via multiple hierarchies in userspace with something like the libcg operations I suggested above.

Paul

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers