Subject: Re: Thoughts on virtualizing task containers
Posted by Paul Menage on Tue, 28 Aug 2007 21:11:28 GMT
View Forum Message <> Reply to Message

On 8/27/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> Agreed, although at k-s last year it was agreed that it's ok if
> processes in a guest know that they are in a guest. That was in
> relation specifically to /proc virtualization.

If that's been already settled on then it simplifies things a fair bit.

>
> So for instance if some vserver is locked to cpus 3 and 4, I think it
> would be nice if that vserver thought it was on a 2-cpu system and had
> full access to cpus 0 and 1, but it's not necessary.

Well, assuming that userspace would normally handle CPU hotplug, then it ought to be able to handle a non-contiguous CPU map, as a real system might appear like that.

>
> So long as the available resources are always a subset of the resources
> assigned to the container, are there situations where it would be
> desirable to not allow the container to sub-partition among it's tasks?

No, but it will probably take a bit of reorganization in the container system.

E.g. suppose that the host doesn't mount subsystem X, and hence all processes in the system are in the root container for subsystem X. If the guest then mounts subsystem X and tries to change the root container parameters for X, that's going to be a problem.

We could get around that by saying that a guest has no control over the parameters of its root container, in the same way that e.g. even root can't add/subtract cpus from the root cpuset, since it represents what's available in hardware. So the guest would have to mount subsystem X, create a subcontainer, and move processes into it in order to change their X limits.

This then brings up the awkward issue of what happens if the host later wants to mount subsystem X itself and apply its own process division (or what if X is already mounted but the host has divided processes in a way that's different from the host/guest process distinction?

These are all reasons why it might be simpler for the host to decide exactly which subsystems are available to the guest, and to require

that all the guest processes be in the same sub-tree for all the
available subsystems.

>
> Since the controls are implemented as vfs operations, perhaps a natural
> way to split these up could be by mounts namespaces.
>
> So you clone(CLONE_NEWNS), then when the child does a mount of
> containerfs filesystem, the container hierarchy applies to the child but
> not the parent.
>
> That's probably not quite right, since we probably don't want every
> CLONE_NEWNS to have that effect.  So instead we could introduce
> CLONE_CONTAINER_LEVEL, which always implies CLONE_NEWNS, and which means
> that any containerfs mounts by the child are not applied to the parent.

How do you define "not applied"?

> > - how much visibility/control does the host have into any child task
> > containers created by the guest?
>
>  2. All containerfs mounts in a child after
>  clone(CLONE_CONTAINER_LEVEL) could appear in the parent's
>  hierarchy mounting the 'virtualization' subsystem.

It's not so much the containerfs mounts that matter, I think - it's
the actual creation of containers and movement of tasks between them.
If we can manage to share the superblocks between the parent and
child, and just use mount tricks to only let the child mount a subtree
of the hierarchy, then this information would be shared automatically.

The only drawback with that is that a superblock defines a container
hierarchy for a specified set of subsystems, which is why we come back
to the issue of different hierarchy compositions in the parent and the
child causing problems.

Paul