Subject: Thoughts on Namespace / Subsystem unification
Posted by Paul Menage on Mon, 03 Sep 2007 12:22:07 GMT
View Forum Message <> Reply to Message

Today at the mini-summit I think that the fact that I was only connected via Skype made it way too difficult for me to get across the idea of my proposals for exploring the potential benefits to be gained from unifying namespaces and "task container subsystems", hereafter just referred to mainly as "subsystems" to avoid confusion over the term container. (Yes, the name may well be changing to something like "task sets" ...) So I'll flesh them out a bit in an email instead.
This should be regarded more as a partially-formed concept/vision than a complete design proposal.

The idea is based on that fact that subsystems and namespaces have a bunch of similarities:

- associate each process with a piece of state (where that state may be resource limits/usage, object translation table, etc)

- allow multiple processes to share the same piece of state in aggregate (e.g. multiple processes allocate resources from the same limit, or use the same ipc lookup table)

- aren't generally changeable/escapable (except by users with root or delegated privileges)

- have a shared aggregator object (nsproxy or css_group) that allows multiple tasks that share the same namespaces/subsystems to cheaply add/remove refcounts from a whole bunch of objects at once.

- are used as state for code that may have hooks scattered throughout the kernel code (e.g. namespace indirection, resource checking).

And they also have a few differences:

1) "subsystems" have a generic and flexible control/monitoring API via the "containerfs" filesystem. Namespaces are viewable internally via existing Unix/Linux APIs, and may potentially have additional custom control/monitoring set up as special-purpose code. (But I believe most don't).

I think that it could be very useful for namespaces to have the same support for control/monitoring. For example, consider the IPC namespace. This has a shm_ctlmni field that controls how many shm ids can be created in total in that namespace. Currently only the root IPC namespace can have its shm_ctlmni updated via sysctl; child namespaces

aren't configurable in the same way. It could be plausible to have the shm_ctlmni in other namespaces be updateable too, assuming that the relevant /proc file was virtualized. But then there are issues such as:

-how does a process in the parent namespace read/write the shmmni value in the child namespace? Having to fork a child into the namespace via something like sys_hijack() seems overly expensive.

- should a namespace' shmmni value be writeable only by its parent, or writeable by the child too (in which case, how does the parent limit the child's IPC id creation?)

If the IPC namespace had the concept of an "internal" view (the shmmni value seen and writeable by the child via normal IPC interfaces) and an "external" view (the shmmni value seen and writeable by the parent, via a control file in containerfs) these problems could be resolved. The child could control its own shmmni value, and the parent could impose an additional limit to control the child's resources. (If it turns out that I've misunderstood the IPC namespace and this was actually a bad example, I hope that you can still appreciate the generic argument that I'm trying to make here).

2) entering the "container" associated with a subsystem is well supported since subsystems are expecting the relevant state pointers to be somewhat volatile; entering namespaces is tricky since lots of existing code doesn't expect the namespace pointer to be volatile, and can't necessarily be updated to allow such volatility since they're performance-critical structures.

But the fact that this is a distinction between namespaces and subsystems is a bit artificial. I think it's quite possible to imagine some namespaces whose implementation can quite easily handle tasks changing their namespace pointer unexpectedly, since they're written to handle the tricky issues this introduces, and aren't so performance critical that they can't do locking when necessary.

3) "subsystems" have new instances created via a mkdir in "containerfs", namespaces have new instances created via clone() or unshare(). But this could just be considered two different ways of creating the same kind of object. The container_clone() call already exists to support the clone/unshare approach used by namespaces. The choice of which was appropriate (or even both?) could be made by the kernel code for the subsystem/namespace in question.

4) "namespaces" expire as soon as no tasks are using them; "subsystems" persist until explicitly deleted. But containerfs already

has "notify on release" support; extending this to include "delete on release" wouldn't be hard for people who wanted their resource controllers and other subsystems cleaned up as soon as they weren't in use, and the same code could support the expected behaviour for namespaces. And in the opposite direction, some users might want to be able to set up some kind of namespace environment and have it persist even when there were no active processes in the nsproxy. (Perhaps pre-allocating environments, or reusing them across multiple operations).

5) There's no straightforward way to view/list namespaces from userspace, since the nsproxy is regarded as purely an in-kernel convenience/performance feature, whereas "subsystems" can be easily viewed and listed via containerfs directories. But this seems like it would be useful behaviour for namespaces too.

I hope this demonstrates that the distinction between namespaces and "subsystems" is at least partially arbitrary, and that namespaces could benefit from a lot of the support that subsystems get automatically from the "task containers" framework.

The ns_container subsystem is a first step towards linking subsystems and namespaces - it associates an entire set of namespaces (via an nsproxy) with a "task container", so the nsproxy is on the same level with other subsystems. But based on the similarities/differences explored above, my argument is that we should explore the idea that subsystems and namespaces should be considered on the same level, rather than subsystems be considered as being on the same level as the nsproxy aggregate. If we could come up with a single abstraction that captures the similarities and differences between namespaces and subsystems, this could give the benefits of both.

I'll call the aggregation of multiple such abstractions a "container" for brevity, although in practice it's somewhere between the concept of my "task container" and the full vision of containers as self-contained virtualised environments.

The abstraction (I'm not sure I have an elegant name for it yet) would have the properties listed as the similarities above; it would be tied to some kind of aggregator that would be similar to an nsproxy or a "task container". It would have a generic filesystem-base control/monitoring API. It would be parameterizable with options such as:

- should a process be allowed to enter this "container" (a property specified by the code itself)

- whether it can be created via mkdir and/or clone/unshare (specified by the code itself)

- what action should be taken if this "container" becomes empty (probably user-specifiable, with options such as "ignore", "notify", "delete")

(I think these three options capture the essential differences between "subsystems" and namespaces as they exist currently).

It's a bit different from the arguments of "everything's a namespace" that have been made in the past, since the new abstraction resembles more a "task container subsystem" than it does the existing definition of a namespace.

In a way it would incorporate some of the ideas of the "rcfs" subsystem that Vatsa proposed a while ago, but with differences such as not having separate arrays for subsystems and namespaces, and having the "container" be a much more first-class object, both in terms of kernel support and in terms of visibility from userspace (compared to the current situation where an nsproxy is purely an in-kernel convenience that's not visible from userspace). There would also be more focus on adding control/monitoring APIs to namespaces.

Paul