Subject: Re: [PATCH 1/2] rcfs core patch
Posted by Herbert Poetzl on Fri, 09 Mar 2007 00:38:19 GMT
View Forum Message <> Reply to Message

On Thu, Mar 08, 2007 at 01:10:24AM -0800, Paul Menage wrote:
> On 3/7/07, Eric W. Biederman <ebiederm@xmission.com> wrote:
> >
> > Please next time this kind of patch is posted add a description of
> > what is happening and why. I have yet to see people explain why
> > this is a good idea. Why the current semantics were chosen.
>
> OK. I thought that the descriptions in my last patch 0/7 and
> Documentation/containers.txt gave a reasonable amount of "why", but I
> can look at adding more details.
>
> >
> > I have a question? What does rcfs look like if we start with
> > the code that is in the kernel? That is start with namespaces
> > and nsproxy and just build a filesystem to display/manipulate them?
> > With the code built so it will support adding resource controllers
> > when they are ready?
>
> There's at least one resource controller that's already in the kernel - cpusets.
>
> > We probably want to rename this struct task_proxy....
> > And then we can rename most of the users things like:
> > dup_task_proxy, clone_task_proxy, get_task_proxy, free_task_proxy,
> > put_task_proxy, exit_task_proxy, init_task_proxy....
>
> That could be a good start.
>
> >
> > This extra list of nsproxy's is unneeded and a performance problem the
> > way it is used. In general we want to talk about the individual resource
> > controllers not the nsproxy.
>
> There's one important reason why it's needed, and highlights one of
> the ways that "resource controllers" are different from the way that
> "namespaces" have currently been used.
>
> Currently with a namespace, you can only unshare, either by
> sys_unshare() or clone() - you can't "reshare" a namespace with some
> other task. But resource controllers tend to have the concept a lot
> more of being able to move between resource classes. If you're going
> to have an ns_proxy/container_group object that gathers together a
> group of pointers to namespaces/subsystem-states, then either:
>
> 1) you only allow a task to reshare *all* namespaces/subsystems with

>   another task, i.e. you can update current->task_proxy to point to
>   other->task_proxy. But that restricts flexibility of movement.
>   It would be impossible to have a process that could enter, say,
>   an existing process' network namespace without also entering its
>   pid/ipc/uts namespaces and all of its resource limits.
>
> 2) you allow a task to selectively reshare namespaces/subsystems with
>   another task, i.e. you can update current->task_proxy to point to
>   a proxy that matches your existing task_proxy in some ways and the
>   task_proxy of your destination in others. In that case a trivial
>   implementation would be to allocate a new task_proxy and copy some
>   pointers from the old task_proxy and some from the new. But then
>   whenever a task moves between different groupings it acquires a
>   new unique task_proxy. So moving a bunch of tasks between two
>   groupings, they'd all end up with unique task_proxy objects with
>   identical contents.

this is exactly what Linux-VServer does right now, and I'm
still not convinced that the nsproxy really buys us anything
compared to a number of different pointers to various spaces
(located in the task struct)

> So it would be much more space efficient to be able to locate an
> existing task_proxy with an identical set of namespace/subsystem
> pointers in that event. The linked list approach that I put in my last
> containers patch was a simple way to do that, and Vatsa's reused it
> for his patches. My intention is to replace it with a more efficient
> lookup (maybe using a hash of the desired pointers?) in a future
> patch.

IMHO that is getting quite complicated and probably very
inefficient, especially if you think hundreds of guests
with a dozent spaces each ... and still we do not know if
the nsproxy is a real benefit either memory or performance
wise ...

> > > +   void *ctlr_data[CONFIG_MAX_RC_SUBSYS];
> >
> > I still don't understand why these pointers are so abstract,
> > and why we need an array lookup into them?
> >
>
> For the same reason that we have:
>
> - generic notifier chains rather than having a big pile of #ifdef'd
>   calls to the various notification sites
>
> - linker sections to define initcalls and per-cpu variables, rather

>   than hard-coding all init calls into init/main.c and having a big
>   per-cpu structure (both of which would again be full of #ifdefs)
>
> It makes the code much more readable, and makes patches much simpler
> and less likely to stomp on one another.
>
> OK, so my current approaches have involved an approach like notifier
> chains, i.e. have a generic list/array, and do something to all the
> objects on that array.

I'd prefer to do accounting (and limits) in a very simple
and especially performant way, and the reason for doing
so is quite simple:

 nobody actually cares about a precise accounting and
 calculating shares or partitions of whatever resource,
 all that matters is that you have a way to prevent a
 potential hostile environment from sucking up all your
 resources (or even a single one) resulting in a DoS

so the main purpose of a resource limit (or accounting)
is to get an idea how much a certain guest uses up, not
more and not less ...

> How about a radically different approach based around the
> initcall/percpu way (linker sections)? Something like:
>
> - each namespace or subsystem defines itself in its own code, via a
> macro such as:
>
> struct task_subsys {
>   const char *name;
>   ...
> };
>
> #define DECLARE_TASKGROUP_SUBSYSTEM(ss) \
>   __attribute__((__section__(".data.tasksubsys"))) struct
> task_subsys *ss##_ptr = &ss
>
>
> It would be used like:
>
> struct taskgroup_subsys uts_ns = {
>   .name = "uts",
>   .unshare = uts_unshare,
> };
>
> DECLARE_TASKGROUP_SUBSYSTEM(uts_ns);

>
> ...
>
> struct taskgroup_subsys cpuset_ss {
>   .name = "cpuset",
>   .create = cpuset_create,
>   .attach = cpuset_attach,
> };
>
> DECLARE_TASKGROUP_SUBSYSTEM(cpuset_ss);
>
> At boot time, the task_proxy init code would figure out from the size
> of the task_subsys section how many pointers had to be in the
> task_proxy object (maybe add a few spares for dynamically-loaded
> modules?). The offset of the subsystem pointer within the task_subsys
> data section would also be the offset of that subsystem's
> per-task-group state within the task_proxy object, which should allow
> accesses to be pretty efficient (with macros providing user-friendly
> access to the appropriate locations in the task_proxy)
>
> The loops in container.c in my patch that iterate over the subsys
> array to perform callbacks, and the code in nsproxy.c that performs
> the same action for each namespace type, would be replaced with
> iterations over the task_subsys data section; possibly some
> pre-processing of the various linked-in subsystems could be done to
> remove unnecessary iterations. The generic code would handle things
> like reference counting.
>
> The existing unshare()/clone() interface would be a way to create a
> child "container" (for want of a better term) that shared some
> subsystem pointers with its parent and had cloned versions of others
> (perhaps only for the namespace-like subsystems?); the filesystem
> interface would allow you to create new "containers" that weren't
> explicitly associated with processes, and to move processes between
> "containers". Also, the filesystem interface would allow you to bind
> multiple subsystems together to allow easier manipulation from
> userspace, in a similar way to my current containers patch.
>
> So in summary, it takes the concepts that resource controllers and
> namespaces share (that of grouping tasks) and unifies them, while
> not forcing them to behave exactly the same way. I can envisage some
> other per-task pointers that are generally inherited by children
> being possibly moved into this in the same way, e.g. task->user and
> task->mempolicy, if we could come up with a solution that handles
> groupings with sufficiently different lifetimes.
>
> Thoughts?

sounds quite complicated and fragile to me ...

but I guess I have to go through that one again
before I can give a final statement ...

best,
Herbert

>
> Paul
> ____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers