Subject: Re: [PATCH 1/9] Containers (V9): Basic container framework
Posted by Paul Menage on Tue, 01 May 2007 17:46:10 GMT
View Forum Message <> Reply to Message

On 5/1/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> menage@google.com wrote:
> > This patch adds the main containers framework - the container
> > filesystem, and the basic structures for tracking membership and
> > associating subsystem state objects to tasks.
>
> [snip]
>
> > +*** notify_on_release is disabled in the current patch set. It may be
> > +*** reactivated in a future patch in a less-intrusive manner
> > +
>
> Won't this break user space tools for cpusets?

Yes, so it's a must-fix before this gets anywhere near a real distribution.

>
> [snip]
>
> > +See kernel/container.c for more details.
> > +
> > +Subsystems can take/release the container_mutex via the functions
> > +container_lock()/container_unlock(), and can
> > +take/release the callback_mutex via the functions
> > +container_lock()/container_unlock().
> > +
>
> Hmm.. looks like a documentation error. Both mutex's are obtained through
> container_lock/container_unlock ?

The second half of that sentence is obsolete and should have been deleted.

>
> > +Accessing a task's container pointer may be done in the following ways:
> > +- while holding container_mutex
> > +- while holding the task's alloc_lock (via task_lock())
> > +- inside an rcu_read_lock() section via rcu_dereference()
> > +
>
> container_mutex() and task_lock() can be used for changing the pointer?

No, these are all for read operations. (Actually, this is a bit of
documentation that's bit-rotted - there's no longer a per-task
"container" pointer). I'll update this.

For write operations, only the container system should be modifying
those pointers (under the protection of both container_mutex and
alloc_lock).

>
> We needed the equivalent of container_remove_file() to be called
> if container_add_file() failed.
>

Yes, this is some incomplete behaviour that I inherited from cpusets.
Needs tidying up.

>
> Can't we derive the top_container from containerfs_root?

Yes, we could for the cost of an extra dereference. Not sure it's a
big deal either way.

```
> > +     ssize_t (*read) (struct container *cont, struct cftype *cft,
> > +                        struct file *file,
> > +                        char __user *buf, size_t nbytes, loff_t *ppos);
> > +     u64 (*read_uint) (struct container *cont, struct cftype *cft);
>
```
> Is this a new callback, a specialization of the read() callback?

Yes. It's to simplify the common case of reporting a number in a
control file. (Not yet well documented :-( )

Paul