Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem

Posted by serue on Mon, 28 Apr 2008 04:03:58 GMT

View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):

```
> >+static const char *freezer_state_strs[] = {
> >+ "RUNNING\n",
> >+ "FREEZING\n" ,
> >+ "FROZEN\n"
> >+};
>
> I think it might be cleaner to not include the \n characters in this array.
>
> >+static inline int cgroup_frozen(struct task_struct *task)
> >+{
> >+ struct cgroup *cgroup = task_cgroup(task, freezer_subsys_id);
> >+ struct freezer *freezer = cgroup_freezer(cgroup);
> >+ enum freezer_state state;
> >+
> >+ spin_lock(&freezer->lock);
> >+ state = freezer->state;
> >+ spin_unlock(&freezer->lock);
> >+
> >+ return (state == STATE_FROZEN);
> >+}
>
> You need to be in an RCU critical section or else hold task_lock() in
> order to dereference the cgroup returned from task_cgroup()
>
> I'm not sure that you need to take freezer->lock here - you're just
> reading a single word.
>
> >+
> >+ if (!capable(CAP_SYS_ADMIN))
> >+  return ERR_PTR(-EPERM);
> >+
>
> Why does everyone keep throwing calls to check CAP_SYS_ADMIN into
> their cgroup create callbacks? You have to be root in order to mount a
> cgroups hierarchy in the first place, and filesystem permissions will
> control who can create new cgroups.
```

The scourge of cut-n-paste :) Except I'm thinking that the check should be taken out of even kernel/ns_cgroup.c:ns_create(), which I think is where that all began.

The reason why tossing these in is bad is that it requires us to give *away* extra privilege.

```
> >+static int freezer_can_attach(struct cgroup_subsys *ss,
> >+          struct cgroup *new_cgroup,
> >+          struct task_struct *task)
> >+{
> >+ struct freezer *freezer = cgroup_freezer(new_cgroup);
> >+ int retval = 0;
> >+
> >+ if (freezer->state == STATE_FROZEN)
> >+  retval = -EBUSY;
> >+
> >+ return retval;
> >+}
>
> You should comment here that the call to cgroup_lock() in the
> freezer.state write method prevents a write to that file racing
> against an attach, and hence the can_attach() result will remain valid
> until the attach completes.
>
> >+static ssize_t freezer_write(struct cgroup *cgroup,
> >+         struct cftype *cft,
> >+         struct file *file,
> >+         const char __user *userbuf,
> >+         size_t nbytes, loff_t *unused_ppos)
> >+{
> >+ char *buffer;
> >+ int retval = 0;
> >+ enum freezer_state goal_state;
> >+
> >+ if (nbytes >= PATH_MAX)
> >+  return -E2BIG;
> >+
> >+ /* +1 for nul-terminator */
> >+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> >+ if (buffer == NULL)
> >+  return -ENOMEM;
>
> Given that you're copying a string whose maximum valid length is
> "FREEZING" you don't really need to use a dynamically-allocated
> buffer.
>
> But I really ought to provide a write_string() method that handles
> this kind of copying on behalf of cgroup subsystems, the way it
> already does for 64-bit ints.
>
> >+ if (strcmp(buffer, "RUNNING") == 0)
> >+  goal_state = STATE_RUNNING;
```

> >+ else if (strcmp(buffer, "FROZEN") == 0)
> >+ goal_state = STATE_FROZEN;
>
> Would it make sense to compare against the strings you already have in
> the array earlier in the file?
>
> Paul
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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