Subject: Re: [patch 3/4] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Mon, 07 Jul 2008 22:42:08 GMT
View Forum Message <> Reply to Message

On Tue, 2008-06-24 at 14:27 -0700, Paul Menage wrote:
> On Tue, Jun 24, 2008 at 6:58 AM, Matt Helsley <matthltc@us.ibm.com> wrote:
> > From: Cedric Le Goater <clg@fr.ibm.com>
> > Subject: [patch 3/4] Container Freezer: Implement freezer cgroup subsystem
> >
> > This patch implements a new freezer subsystem for Paul Menage's
> > control groups framework.
>
> You can s/Paul Menage's// now that it's in mainline.

OK. Incidentally sorry for the delayed reply. Got so caught up in making changes in response to your email that I neglected to reply sooner. I'll be posting the changes shortly but first I want to address your earlier mail.

> > +static const char *freezer_state_strs[] = {
> > +	"RUNNING",
> > +	"FREEZING",
> > +	"FROZEN",
> > +};
> > +
> > +/* Check and update whenever adding new freezer states. Currently is:
> > +   strlen("FREEZING") */
> > +#define STATE_MAX_STRLEN 8
> > +
>
> That's a bit nasty ...
>
> But hopefully it could go away when the write_string() method is
> available in cgroups? (See my patchset from earlier this week).

I've looked at this and I like it. I've changed the patches to use this interface.

> > +
> > +struct cgroup_subsys freezer_subsys;
> > +
> > +/* Locking and lock ordering:
> > + *
> > + * can_attach(), cgroup_frozen():
> > + * rcu (task->cgroup, freezer->state)
> > + *
> > + * freezer_fork():
> > + * rcu (task->cgroup, freezer->state)

```
> > + *  freezer->lock
> > + *   task_lock
> > + *    sighand->siglock
> > + *
> > + * freezer_read():
> > + * rcu (freezer->state)
> > + *  freezer->lock (upgrade to write)
> > + *   read_lock css_set_lock
> > + *
> > + * freezer_write()
> > + * cgroup_lock
> > + *  rcu
> > + *   freezer->lock
> > + *    read_lock css_set_lock
> > + *     task_lock
> > + *      sighand->siglock
> > + *
> > + * freezer_create(), freezer_destroy():
> > + * cgroup_lock [ by cgroup core ]
> > + */
>
>
> > +static int freezer_can_attach(struct cgroup_subsys *ss,
> > +                              struct cgroup *new_cgroup,
> > +                              struct task_struct *task)
> > +{
> > +       struct freezer *freezer;
> > +       int retval = 0;
> > +
> > +       /*
> > +        * 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.
> > +        */
> > +       rcu_read_lock();
> > +       freezer = cgroup_freezer(new_cgroup);
> > +       if (freezer->state == STATE_FROZEN)
> > +               retval = -EBUSY;
>
> Is it meant to be OK to move a task into a cgroup that's currently in
> the FREEZING state but not yet fully frozen?

Yes.

> > +       struct freezer *freezer;
> > +
> > +       rcu_read_lock(); /* needed to fetch task's cgroup
> > +                           can't use task_lock() here because
```

> > +                       freeze_task() grabs that */
>
> I'm not sure that RCU is the right thing for this. All that the RCU
> lock will guarantee is that the freezer structure you get a pointer to
> doesn't go away. It doesn't guarantee that the task doesn't move
> cgroup, or that the cgroup doesn't get a freeze request via a write.
> But in this case, the fork callback is called before the task is added
> to the task_list/pidhash, or to its cgroups' linked lists. So it
> shouldn't be able to change groups. Racing against a concurrent write
> to the cgroup's freeze file may be more of an issue.

I think you're right. The problem is it could change state between the
test of the state and the call to freeze_task(). If we're changing from
FROZEN to running that would leave us with a frozen task even though
we're in the running state. Thanks for spotting this one.

> Can you add a __freeze_task() that has to be called with task_lock(p)
> already held?

task_lock() is no longer acquired in freeze_task(). So I've updated the
patches to drop RCU in favor of acquiring the task_lock() here. It's
still taken in thaw_process() however, so something like this is still
needed.

> > +       freezer = task_freezer(task);
>
> Maybe BUG_ON(freezer->state == STATE_FROZEN) ?

Seems appropriate.

> > +
> > +static ssize_t freezer_read(struct cgroup *cgroup,
> > +                       struct cftype *cft,
> > +                       struct file *file, char __user *buf,
> > +                       size_t nbytes, loff_t *ppos)
> > +{
> > +       struct freezer *freezer;
> > +       enum freezer_state state;
> > +
> > +       rcu_read_lock();
> > +       freezer = cgroup_freezer(cgroup);
> > +       state = freezer->state;
> > +       if (state == STATE_FREEZING) {
> > +               /* We change from FREEZING to FROZEN lazily if the cgroup was
> > +                * only partially frozen when we exitted write. */
> > +               spin_lock_irq(&freezer->lock);
> > +               if (freezer_check_if_frozen(cgroup)) {
> > +                       freezer->state = STATE_FROZEN;

```
> > +                       state = STATE_FROZEN;
> > +               }
> > +               spin_unlock_irq(&freezer->lock);
> > +       }
> > +       rcu_read_unlock();
> > +
> > +       return simple_read_from_buffer(buf, nbytes, ppos,
> > +                                      freezer_state_strs[state],
> > +                                      strlen(freezer_state_strs[state]));
> > +}
>
> Technically this could return weird results if someone read it
> byte-by-byte and the status changed between reads. If you used
> read_seq_string rather than read you'd avoid that.
```

Good point. I've made that change as well.

```
> > +               return -EIO;
> > +
> > +       cgroup_lock();
>
> If you're taking cgroup_lock() here in freezer_write(), there's no
> need for the rcu_read_lock() in freezer_freeze()
```

Yup. Fixed since I'll no longer be using RCU.

Cheers,
-Matt Helsley