Subject: [RFC][PATCH] Container Freezer: Don't Let Frozen Stuff Change
Posted by Matt Helsley on Thu, 10 Jul 2008 02:18:29 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 09:42 +0900, KAMEZAWA Hiroyuki wrote:
> On Wed, 09 Jul 2008 14:58:43 -0700
> Matt Helsley <matthltc@us.ibm.com> wrote:
>
> >
> > On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> > > On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> > > > On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> > > >>
> > > >> One is to try and disallow users from moving frozen tasks. That doesn't
> > > >> seem like a good approach since it would require a new cgroups interface
> > > >> "can_detach()".
> > > >
> > > > Detaching from the old cgroup happens at the same time as attaching to
> > > > the new cgroup, so can_attach() would work here.
> >
> > Update: I've made a patch implementing this. However it might be better
> > to just modify attach() to thaw the moving task rather than disallow
> > moving the frozen task. Serge, Cedric, Kame-san, do you have any
> > thoughts on which is more useful and/or intuitive?
> >
>
> Thank you for explanation in previous mail.
>
> Hmm, just thawing seems atractive but it will confuse people (I think).
>
> I think some kind of process-group is freezed by this freezer and "moving
> freezed task" is wrong(unexpected) operation in general. And there will
> be no demand to do that from users.
> I think just taking "moving freezed task" as error-operation and returning
> -EBUSY is better.

Kame-san,

 I've been working on changes to the can_attach() code so it was pretty
easy to try this out.

 Don't let frozen tasks or cgroups change. This means frozen tasks can't
leave their current cgroup for another cgroup. It also means that tasks
cannot be added to or removed from a cgroup in the FROZEN state. We
enforce these rules by checking for frozen tasks and cgroups in the
can_attach() function.

Signed-off-by: Matt Helsley <matthltc@us.ibm.com>

```
---
Builds, boots, passes testing against 2.6.26-rc5-mm2

 kernel/cgroup_freezer.c |   42 +++++++++++++++++++++++++-----------------
 1 file changed, 25 insertions(+), 17 deletions(-)

Index: linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
===================================================================
--- linux-2.6.26-rc5-mm2.orig/kernel/cgroup_freezer.c
+++ linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
@@ -89,26 +89,43 @@ static void freezer_destroy(struct cgrou
        struct cgroup *cgroup)
 {
  kfree(cgroup_freezer(cgroup));
 }

+/* Task is frozen or will freeze immediately when next it gets woken */
+static bool is_task_frozen_enough(struct task_struct *task)
+{
+ return (frozen(task) || (task_is_stopped_or_traced(task) && freezing(task)));
+}

+/*
+ * 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 int freezer_can_attach(struct cgroup_subsys *ss,
         struct cgroup *new_cgroup,
         struct task_struct *task)
 {
  struct freezer *freezer;
- int retval = 0;
+ int retval;
+
+ /* Anything frozen can't move or be moved to/from */
+
+ if (is_task_frozen_enough(task))
+  return -EBUSY;

- /*
-  * 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.
-  */
  freezer = cgroup_freezer(new_cgroup);
  if (freezer->state == STATE_FROZEN)
+  return -EBUSY;
```

```
+
+ retval = 0;
+ task_lock(task);
+ freezer = task_freezer(task);
+ if (freezer->state == STATE_FROZEN)
   retval = -EBUSY;
+ task_unlock(task);
  return retval;
 }

 static void freezer_fork(struct cgroup_subsys *ss, struct task_struct *task)
 {
@@ -139,16 +156,11 @@ static void check_if_frozen(struct cgrou
  unsigned int nfrozen = 0, ntotal = 0;

  cgroup_iter_start(cgroup, &it);
  while ((task = cgroup_iter_next(cgroup, &it))) {
   ntotal++;
-  /*
-   * Task is frozen or will freeze immediately when next it gets
-   * woken
-   */
-  if (frozen(task) ||
-      (task_is_stopped_or_traced(task) && freezing(task)))
+  if (is_task_frozen_enough(task))
    nfrozen++;
  }

  /*
   * Transition to FROZEN when no new tasks can be added ensures
@@ -195,15 +207,11 @@ static int try_to_freeze_cgroup(struct c
  freezer->state = STATE_FREEZING;
  cgroup_iter_start(cgroup, &it);
  while ((task = cgroup_iter_next(cgroup, &it))) {
   if (!freeze_task(task, true))
    continue;
-  if (task_is_stopped_or_traced(task) && freezing(task))
-   /*
-    * The freeze flag is set so these tasks will
-    * immediately go into the fridge upon waking.
-    */
+  if (is_task_frozen_enough(task))
    continue;
   if (!freezing(task) && !freezer_should_skip(task))
    num_cant_freeze_now++;
  }
  cgroup_iter_end(cgroup, &it);
```