Subject: [patch 3/4] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Tue, 24 Jun 2008 13:58:26 GMT
View Forum Message <> Reply to Message

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. It provides a way to stop and resume
execution of all tasks in a cgroup by writing in the cgroup
filesystem.

This is the basic mechanism which should do the right thing for
user space tasks in a simple scenario. This will require more work
to get the freezing right (cf. try_to_freeze_tasks()) for ptraced
tasks.

It's important to note that freezing can be incomplete. In that case we return
EBUSY. This means that some tasks in the cgroup are busy doing something that
prevents us from completely freezing the cgroup at this time. After EBUSY,
the cgroup will remain partially frozen -- reflected by freezer.state reporting
"FREEZING" when read. The state will remain "FREEZING" until one of these
things happens:

 1) Userspace cancels the freezing operation by writing "RUNNING" to
  the freezer.state file
 2) Userspace retries the freezing operation by writing "FROZEN" to
  the freezer.state file (writing "FREEZING" is not legal
  and returns EIO)
 3) The tasks that blocked the cgroup from entering the "FROZEN"
  state disappear from the cgroup's set of tasks.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
Tested-by: Matt Helsley <matthltc@us.ibm.com>

TODO:
 Check that we handle ptrace'd and vfork-ing tasks correctly.

Changelog:
v2:
 Moved the "kill" file into a separate cgroup subsystem (signal) and
  it's own patch.
 Changed the name of the file from freezer.freeze to freezer.state.
 Switched from taking 1 and 0 as input to the strings "FROZEN" and
  "RUNNING", respectively. This helps keep the interface
  human-usable if/when we need to more states.
 Checked that stopped or interrupted is "frozen enough"

Since try_to_freeze() is called upon wakeup of these tasks
this should be fine. This idea comes from recent changes to
the freezer.
Checked that if (task == current) whilst freezing cgroup we're ok
Fixed bug where -EBUSY would always be returned when freezing
Added code to handle userspace retries for any remaining -EBUSY

include/linux/cgroup_freezer.h | 73 +++++++++
include/linux/cgroup_subsys.h | 6
init/Kconfig | 7
kernel/Makefile | 1
kernel/cgroup_freezer.c | 316 ++
5 files changed, 403 insertions(+)

Index: linux-2.6.26-rc5-mm2/include/linux/cgroup_freezer.h
===
--- /dev/null
+++ linux-2.6.26-rc5-mm2/include/linux/cgroup_freezer.h
@@ -0,0 +1,73 @@
+#ifndef _LINUX_CGROUP_FREEZER_H
+#define _LINUX_CGROUP_FREEZER_H
+/*
+ * cgroup_freezer.h - control group freezer subsystem interface
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#include <linux/cgroup.h>
+
+#ifdef CONFIG_CGROUP_FREEZER
+
+enum freezer_state {
+ STATE_RUNNING = 0,
+ STATE_FREEZING,
+ STATE_FROZEN,
+};
+
+struct freezer {

```
+ struct cgroup_subsys_state css;
+ enum freezer_state state;
+ spinlock_t lock; /* protects writes to state -- reads are RCU protected
+       because that also helps us when we need to call
+       task_cgroup() */
+};
+
+static inline struct freezer *cgroup_freezer(
+  struct cgroup *cgroup)
+{
+ return container_of(
+  cgroup_subsys_state(cgroup, freezer_subsys_id),
+  struct freezer, css);
+}
+
+static inline struct freezer *task_freezer(struct task_struct *task)
+{
+ return container_of(task_subsys_state(task, freezer_subsys_id),
+       struct freezer, css);
+}
+
+static inline int cgroup_frozen(struct task_struct *task)
+{
+ struct freezer *freezer;
+ enum freezer_state state;
+
+ rcu_read_lock(); /* protects use of task's cgroup ptr */
+ freezer = task_freezer(task);
+ state = freezer->state;
+ rcu_read_unlock();
+
+ return state == STATE_FROZEN;
+}
+
+#else /* !CONFIG_CGROUP_FREEZER */
+
+static inline int cgroup_frozen(struct task_struct *task)
+{
+ return 0;
+}
+
+#endif /* !CONFIG_CGROUP_FREEZER */
+
+#endif /* _LINUX_CGROUP_FREEZER_H */
Index: linux-2.6.26-rc5-mm2/include/linux/cgroup_subsys.h
===================================================================
--- linux-2.6.26-rc5-mm2.orig/include/linux/cgroup_subsys.h
+++ linux-2.6.26-rc5-mm2/include/linux/cgroup_subsys.h
```

```
@@ -50,5 +50,11 @@ SUBSYS(devices)
 #ifdef CONFIG_CGROUP_MEMRLIMIT_CTLR
 SUBSYS(memrlimit_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_FREEZER
+SUBSYS(freezer)
+#endif
+
+/* */
Index: linux-2.6.26-rc5-mm2/init/Kconfig
===================================================================
--- linux-2.6.26-rc5-mm2.orig/init/Kconfig
+++ linux-2.6.26-rc5-mm2/init/Kconfig
@@ -329,10 +329,17 @@ config GROUP_SCHED
  default n
  help
    This feature lets CPU scheduler recognize task groups and control CPU
    bandwidth allocation to such task groups.

+config CGROUP_FREEZER
+       bool "control group freezer subsystem"
+       depends on CGROUPS
+       help
+         Provides a way to freeze and unfreeze all tasks in a
+   cgroup
+
 config FAIR_GROUP_SCHED
  bool "Group scheduling for SCHED_OTHER"
  depends on GROUP_SCHED
  default GROUP_SCHED

Index: linux-2.6.26-rc5-mm2/kernel/Makefile
===================================================================
--- linux-2.6.26-rc5-mm2.orig/kernel/Makefile
+++ linux-2.6.26-rc5-mm2/kernel/Makefile
@@ -49,10 +49,11 @@ obj-$(CONFIG_PM) += power/
 obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
 obj-$(CONFIG_KEXEC) += kexec.o
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
+obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
 obj-$(CONFIG_UTS_NS) += utsname.o
```

```
 obj-$(CONFIG_USER_NS) += user_namespace.o
 obj-$(CONFIG_PID_NS) += pid_namespace.o
Index: linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
===================================================================
--- /dev/null
+++ linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
@@ -0,0 +1,316 @@
+/*
+ * cgroup_freezer.c -  control group freezer subsystem
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
+#include <linux/freezer.h>
+#include <linux/cgroup_freezer.h>
+
+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
+
+
+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 struct cgroup_subsys_state *freezer_create(
+ struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ struct freezer *freezer;
+
+ freezer = kzalloc(sizeof(struct freezer), GFP_KERNEL);
+ if (!freezer)
+  return ERR_PTR(-ENOMEM);
+
+ spin_lock_init(&freezer->lock);
+ freezer->state = STATE_RUNNING;
+ return  &freezer->css;
+}
+
+static void freezer_destroy(struct cgroup_subsys *ss,
+       struct cgroup *cgroup)
+{
+ kfree(cgroup_freezer(cgroup));
+}
+
+
+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;
+ rcu_read_unlock();
+
+ return retval;
+}
+
+static void freezer_fork(struct cgroup_subsys *ss, struct task_struct *task)
+{
+ struct freezer *freezer;
+
+ rcu_read_lock(); /* needed to fetch task's cgroup
+       can't use task_lock() here because
+       freeze_task() grabs that */
+ freezer = task_freezer(task);
+ if (freezer->state == STATE_FREEZING)
+  freeze_task(task, 1);
+ rcu_read_unlock();
+}
+
+static int freezer_check_if_frozen(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ unsigned int nfrozen = 0, ntotal = 0;
+
+ cgroup_iter_start(cgroup, &it);
+
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+  ntotal++;
+  if (frozen(task))
+   nfrozen++;
+ }
+ cgroup_iter_end(cgroup, &it);
+
+ return nfrozen == ntotal;
+}
+
+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]));
+}
+
+static int freezer_freeze_tasks(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ unsigned int num_cant_freeze_now = 0;
+
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+  if (!freeze_task(task, 1))
+   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.
+    */
+   continue;
+  if (!freezing(task) && !freezer_should_skip(task))
+   num_cant_freeze_now++;
+ }
+ cgroup_iter_end(cgroup, &it);
+
```

```
+ return num_cant_freeze_now ? -EBUSY : 0;
+}
+
+static int freezer_unfreeze_tasks(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it)))
+  thaw_process(task);
+
+ cgroup_iter_end(cgroup, &it);
+ return 0;
+}
+
+static int freezer_freeze(struct cgroup *cgroup, enum freezer_state goal_state)
+{
+ struct freezer *freezer;
+ int retval = 0;
+
+ rcu_read_lock();
+ freezer = cgroup_freezer(cgroup);
+retry:
+ if (goal_state == freezer->state)
+  goto unlock;
+ spin_lock_irq(&freezer->lock);
+ switch (freezer->state) {
+ case STATE_RUNNING:
+  if (goal_state == STATE_FROZEN) {
+   freezer->state = STATE_FREEZING;
+   retval = freezer_freeze_tasks(cgroup);
+   if (retval == 0)
+    freezer->state = STATE_FROZEN;
+  }
+  break;
+ case STATE_FREEZING:
+  if (freezer_check_if_frozen(cgroup)) {
+   freezer->state = STATE_FROZEN;
+   spin_unlock_irq(&freezer->lock);
+   goto retry;
+  }
+
+  if (goal_state == STATE_FROZEN) {
+   /* Userspace is retrying after
+    * "/bin/echo FROZEN > freezer.state" returned -EBUSY */
+   retval = freezer_freeze_tasks(cgroup);
+   if (retval == 0)
```

```
+    freezer->state = STATE_FROZEN;
+   break;
+  }
+  /* state == FREEZING and goal_state == RUNNING, so unfreeze */
+ case STATE_FROZEN:
+  if (goal_state == STATE_RUNNING) {
+   freezer->state = STATE_RUNNING;
+   retval = freezer_unfreeze_tasks(cgroup);
+  }
+  break;
+ default:
+  break;
+ }
+ spin_unlock_irq(&freezer->lock);
+unlock:
+ rcu_read_unlock();
+
+ return retval;
+}
+
+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[STATE_MAX_STRLEN + 1];
+ int retval = 0;
+ enum freezer_state goal_state;
+
+ if (nbytes >= PATH_MAX)
+  return -E2BIG;
+ nbytes = min(sizeof(buffer) - 1, nbytes);
+ if (copy_from_user(buffer, userbuf, nbytes))
+  return -EFAULT;
+ buffer[nbytes + 1] = 0; /* nul-terminate */
+ strstrip(buffer); /* remove any trailing whitespace */
+ if (strcmp(buffer, freezer_state_strs[STATE_RUNNING]) == 0)
+  goal_state = STATE_RUNNING;
+ else if (strcmp(buffer, freezer_state_strs[STATE_FROZEN]) == 0)
+  goal_state = STATE_FROZEN;
+ else
+  return -EIO;
+
+ cgroup_lock();
+
+ if (cgroup_is_removed(cgroup)) {
+  retval = -ENODEV;
```

```
+  goto unlock;
+ }
+
+ retval = freezer_freeze(cgroup, goal_state);
+ if (retval == 0)
+  retval = nbytes;
+unlock:
+ cgroup_unlock();
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "state",
+  .read = freezer_read,
+  .write = freezer_write,
+ },
+};
+
+static int freezer_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ return cgroup_add_files(cgroup, ss, files, ARRAY_SIZE(files));
+}
+
+struct cgroup_subsys freezer_subsys = {
+ .name  = "freezer",
+ .create  = freezer_create,
+ .destroy = freezer_destroy,
+ .populate = freezer_populate,
+ .subsys_id = freezer_subsys_id,
+ .can_attach = freezer_can_attach,
+ .attach  = NULL,
+ .fork  = freezer_fork,
+ .exit  = NULL,
+};

--
```

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

Subject: Re: [patch 3/4] Container Freezer: Implement freezer cgroup subsystem
Posted by Paul Menage on Tue, 24 Jun 2008 21:27:10 GMT
View Forum Message <> Reply to Message

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.

```
> +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).

```
> +
> +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?

```
> +       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.

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

```
> +       freezer = task_freezer(task);
```

Maybe BUG_ON(freezer->state == STATE_FROZEN) ?

```
> +
```

```
> +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.

```
> +               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()

Paul

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

---

Subject: Re: [patch 3/4] Container Freezer: Implement freezer cgroup subsystem

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

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