
Subject: [PATCH 2/2] cgroup: Remove call to synchronize_rcu in cgroup_attach_task

Posted by [Colin Cross](#) on Wed, 24 Nov 2010 05:37:04 GMT

[View Forum Message](#) <> [Reply to Message](#)

synchronize_rcu can be very expensive, averaging 100 ms in some cases. In cgroup_attach_task, it is used to prevent a task->cgroups pointer dereferenced in an RCU read side critical section from being invalidated, by delaying the call to put_css_set until after an RCU grace period.

To avoid the call to synchronize_rcu, make the put_css_set call rcu-safe by moving the deletion of the css_set links into free_css_set_work, scheduled by the rcu callback free_css_set_rcu.

The decrement of the cgroup refcount is no longer synchronous with the call to put_css_set, which can result in the cgroup refcount staying positive after the last call to cgroup_attach_task returns. To allow the cgroup to be deleted with cgroup_rmdir synchronously after cgroup_attach_task, have rmdir check the refcount of all associated css_sets. If cgroup_rmdir is called on a cgroup for which the css_sets all have refcount zero but the cgroup refcount is nonzero, reuse the rmdir waitqueue to block the rmdir until free_css_set_work is called.

Signed-off-by: Colin Cross <ccross@android.com>

```
include/linux/cgroup.h | 1 +
kernel/cgroup.c        | 120 ++++++-----
2 files changed, 74 insertions(+), 47 deletions(-)
```

```
diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
```

```
index 9e13078..49fdff0 100644
```

```
--- a/include/linux/cgroup.h
```

```
+++ b/include/linux/cgroup.h
```

```
@@ -279,6 +279,7 @@ struct css_set {
```

```
    /* For RCU-protected deletion */
    struct rcu_head rcu_head;
+ struct work_struct work;
};
```

```
/*
```

```
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
```

```
index 34e855e..e752c83 100644
```

```
--- a/kernel/cgroup.c
```

```

+++ b/kernel/cgroup.c
@@ -267,6 +267,33 @@ static void cgroup_release_agent(struct work_struct *work);
static DECLARE_WORK(release_agent_work, cgroup_release_agent);
static void check_for_release(struct cgroup *cgrp);

+/*
+ * A queue for waiters to do rmdir() cgroup. A tasks will sleep when
+ * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
+ * reference to css->refcnt. In general, this refcnt is expected to goes down
+ * to zero, soon.
+ *
+ * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
+ */
+DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
+
+static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
+{
+ if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
+ wake_up_all(&cgroup_rmdir_waitq);
+}
+
+void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
+{
+ css_get(css);
+}
+
+void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
+{
+ cgroup_wakeup_rmdir_waiter(css->cgroup);
+ css_put(css);
+}
+
+/* Link structure for associating css_set objects with cgroups */
+struct cg_cgroup_link {
+/*
@@ -326,10 +353,35 @@ static struct hlist_head *css_set_hash(struct cgroup_subsys_state
*css[])
return &css_set_table[index];
}

+static void free_css_set_work(struct work_struct *work)
+{
+ struct css_set *cg = container_of(work, struct css_set, work);
+ struct cg_cgroup_link *link;
+ struct cg_cgroup_link *saved_link;
+
+ write_lock(&css_set_lock);
+ list_for_each_entry_safe(link, saved_link, &cg->cg_links,

```

```

+   cg_link_list) {
+   struct cgroup *cgrp = link->cgrp;
+   list_del(&link->cg_link_list);
+   list_del(&link->cgrp_link_list);
+   if (atomic_dec_and_test(&cgrp->count)) {
+   check_for_release(cgrp);
+   cgroup_wakeup_rmdir_waiter(cgrp);
+   }
+   kfree(link);
+ }
+ write_unlock(&css_set_lock);
+
+ kfree(cg);
+}
+
static void free_css_set_rcu(struct rcu_head *obj)
{
    struct css_set *cg = container_of(obj, struct css_set, rcu_head);
- kfree(cg);
+
+ INIT_WORK(&cg->work, free_css_set_work);
+ schedule_work(&cg->work);
}

/* We don't maintain the lists running through each css_set to its
@@ -348,8 +400,6 @@ static inline void get_css_set(struct css_set *cg)

static void put_css_set(struct css_set *cg)
{
- struct cg_cgroup_link *link;
- struct cg_cgroup_link *saved_link;
/*
 * Ensure that the refcount doesn't hit zero while any readers
 * can see it. Similar to atomic_dec_and_lock(), but for an
@@ -363,21 +413,9 @@ static void put_css_set(struct css_set *cg)
return;
}

- /* This css_set is dead. unlink it and release cgroup refcounts */
hlist_del(&cg->hlist);
css_set_count--;

- list_for_each_entry_safe(link, saved_link, &cg->cg_links,
-   cg_link_list) {
-   struct cgroup *cgrp = link->cgrp;
-   list_del(&link->cg_link_list);
-   list_del(&link->cgrp_link_list);
-   if (atomic_dec_and_test(&cgrp->count))

```

```

- check_for_release(cgrp);
-
- kfree(link);
- }
-
- write_unlock(&css_set_lock);
- call_rcu(&cgrp->rcu_head, free_css_set_rcu);
- }
@@ -711,9 +749,9 @@ static struct cgroup *task_cgroup_from_root(struct task_struct *task,
 * cgroup_attach_task(), which overwrites one task's cgroup pointer with
 * another. It does so using cgroup_mutex, however there are
 * several performance critical places that need to reference
- * task->cgroup without the expense of grabbing a system global
+ * task->cgroups without the expense of grabbing a system global
 * mutex. Therefore except as noted below, when dereferencing or, as
- * in cgroup_attach_task(), modifying a task's cgroup pointer we use
+ * in cgroup_attach_task(), modifying a task's cgroups pointer we use
 * task_lock(), which acts on a spinlock (task->alloc_lock) already in
 * the task_struct routinely used for such matters.
 *
@@ -895,33 +933,6 @@ static void cgroup_d_remove_dir(struct dentry *dentry)
}

/*
- * A queue for waiters to do rmdir() cgroup. A task will sleep when
- * cgroup->count == 0 && list_empty(&cgroup->children) && subsys has some
- * reference to css->refcnt. In general, this refcnt is expected to go down
- * to zero, soon.
- *
- * CGRP_WAIT_ON_RMDIR flag is set under cgroup's inode->i_mutex;
- */
-DECLARE_WAIT_QUEUE_HEAD(cgroup_rmdir_waitq);
-
-static void cgroup_wakeup_rmdir_waiter(struct cgroup *cgrp)
-{
- if (unlikely(test_and_clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags)))
- wake_up_all(&cgroup_rmdir_waitq);
-}
-
-void cgroup_exclude_rmdir(struct cgroup_subsys_state *css)
-{
- css_get(css);
-}
-
-void cgroup_release_and_wakeup_rmdir(struct cgroup_subsys_state *css)
-{
- cgroup_wakeup_rmdir_waiter(css->cgroup);
- css_put(css);

```

```

-}
-
-/*
 * Call with cgroup_mutex held. Drops reference counts on modules, including
 * any duplicate ones that parse_cgroupfs_options took. If this function
 * returns an error, no reference counts are touched.
@@ -1788,7 +1799,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
    ss->attach(ss, cgrp, oldcgrp, tsk, false);
}
set_bit(CGRP_RELEASABLE, &cgrp->flags);
- synchronize_rcu();
+ /* put_css_set will not destroy cg until after an RCU grace period */
put_css_set(cg);

/*
@@ -3546,6 +3557,21 @@ static int cgroup_clear_css_refs(struct cgroup *cgrp)
    return !failed;
}

+/* checks if all of the css_sets attached to a cgroup have a refcount of 0.
+ * Must be called with css_set_lock held */
+static int cgroup_css_sets_empty(struct cgroup *cgrp)
+{
+ struct cg_cgroup_link *link;
+
+ list_for_each_entry(link, &cgrp->css_sets, cgrp_link_list) {
+ struct css_set *cg = link->cg;
+ if (atomic_read(&cg->refcount) > 0)
+ return 0;
+ }
+
+ return 1;
+}
+
static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
{
    struct cgroup *cgrp = dentry->d_fsdata;
@@ -3558,7 +3584,7 @@ static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
    /* the vfs holds both inode->i_mutex already */
    again:
    mutex_lock(&cgroup_mutex);
- if (atomic_read(&cgrp->count) != 0) {
+ if (!cgroup_css_sets_empty(cgrp)) {
    mutex_unlock(&cgroup_mutex);
    return -EBUSY;
}
@@ -3591,7 +3617,7 @@ again:

```

```
mutex_lock(&cgroup_mutex);
parent = cgrp->parent;
- if (atomic_read(&cgrp->count) || !list_empty(&cgrp->children)) {
+ if (!cgroup_css_sets_empty(cgrp) || !list_empty(&cgrp->children)) {
    clear_bit(CGRP_WAIT_ON_RMDIR, &cgrp->flags);
    mutex_unlock(&cgroup_mutex);
    return -EBUSY;
--
```

1.7.3.1

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
