Subject: [PATCH 2/3] don't take cgroup_mutex in destroy()
Posted by Glauber Costa on Thu, 19 Apr 2012 22:49:17 GMT
View Forum Message <> Reply to Message

Most of the destroy functions are only doing very simple things
like freeing memory.

The ones who goes through lists and such, already use its own
locking for those.

* The cgroup itself won't go away until we free it, (after destroy)
* The parent won't go away because we hold a reference count
* There are no more tasks in the cgroup, and the cgroup is declared
  dead (cgroup_is_removed() == true)

For the blk-cgroup and the cpusets, I got the impression that the mutex
is still necessary.

For those, I grabbed it from within the destroy function itself.

If the maintainer for those subsystems consider it safe to remove
it, we can discuss it separately.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Tejun Heo <tj@kernel.org>
CC: Li Zefan <lizefan@huawei.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Vivek Goyal <vgoyal@redhat.com>
---
 block/blk-cgroup.c | 2 ++
 kernel/cgroup.c | 9 ++++++-----
 kernel/cpuset.c | 2 ++
 3 files changed, 8 insertions(+), 5 deletions(-)

diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
index 126c341..477463f 100644
--- a/block/blk-cgroup.c
+++ b/block/blk-cgroup.c
@@ -1527,6 +1527,7 @@ static void blkiocg_destroy(struct cgroup *cgroup)
  struct blkio_policy_type *blkiop;
  struct blkio_policy_node *pn, *pntmp;

+ cgroup_lock();
  rcu_read_lock();
  do {
   spin_lock_irqsave(&blkcg->lock, flags);
@@ -1566,6 +1567,7 @@ static void blkiocg_destroy(struct cgroup *cgroup)
  rcu_read_unlock();

```
  if (blkcg != &blkio_root_cgroup)
   kfree(blkcg);
+ cgroup_unlock();
 }

 static struct cgroup_subsys_state *blkiocg_create(struct cgroup *cgroup)
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 932c318..976d332 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -869,13 +869,13 @@ static void cgroup_diput(struct dentry *dentry, struct inode *inode)
    * agent */
   synchronize_rcu();

-  mutex_lock(&cgroup_mutex);
   /*
    * Release the subsystem state objects.
    */
   for_each_subsys(cgrp->root, ss)
    ss->destroy(cgrp);

+  mutex_lock(&cgroup_mutex);
   cgrp->root->number_of_cgroups--;
   mutex_unlock(&cgroup_mutex);

@@ -3994,13 +3994,12 @@ static long cgroup_create(struct cgroup *parent, struct dentry
*dentry,

  err_destroy:

+ mutex_unlock(&cgroup_mutex);
  for_each_subsys(root, ss) {
   if (cgrp->subsys[ss->subsys_id])
    ss->destroy(cgrp);
  }

- mutex_unlock(&cgroup_mutex);
-
  /* Release the reference count that we took on the superblock */
  deactivate_super(sb);

@@ -4349,9 +4348,9 @@ int __init_or_module cgroup_load_subsys(struct cgroup_subsys *ss)
   int ret = cgroup_init_idr(ss, css);
   if (ret) {
    dummytop->subsys[ss->subsys_id] = NULL;
+   mutex_unlock(&cgroup_mutex);
    ss->destroy(dummytop);
    subsys[i] = NULL;
```

```
-   mutex_unlock(&cgroup_mutex);
    return ret;
   }
  }
@@ -4447,10 +4446,10 @@ void cgroup_unload_subsys(struct cgroup_subsys *ss)
   * pointer to find their state. note that this also takes care of
   * freeing the css_id.
   */
+ mutex_unlock(&cgroup_mutex);
  ss->destroy(dummytop);
  dummytop->subsys[ss->subsys_id] = NULL;

- mutex_unlock(&cgroup_mutex);
 }
 EXPORT_SYMBOL_GPL(cgroup_unload_subsys);

diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index 8c8bd65..3cd4916 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
@@ -1862,10 +1862,12 @@ static void cpuset_destroy(struct cgroup *cont)
 {
  struct cpuset *cs = cgroup_cs(cont);

+ cgroup_lock();
  if (is_sched_load_balance(cs))
   update_flag(CS_SCHED_LOAD_BALANCE, cs, 0);

  number_of_cpusets--;
+ cgroup_unlock();
  free_cpumask_var(cs->cpus_allowed);
  kfree(cs);
 }
--
1.7.7.6
```