Subject: [PATCH 8/9] Containers (V9): Share css_group arrays between tasks with same container memberships
Posted by Paul Menage on Fri, 27 Apr 2007 10:46:15 GMT
View Forum Message <> Reply to Message

This patch replaces the struct css_group embedded in task_struct with
a pointer; all tasks with the same set of memberships across all
hierarchies will share a css_group object.

The css_group used by init isn't refcounted, since it can't ever be
freed; this speeds up fork/exit for any systems that have containers
compiled in but haven't actually created any containers other than the
default one.

With more than one registered subsystem, this reduces the number of
atomic inc/dec operations required when tasks fork/exit;

Assuming that many tasks share the same container assignments, this
reduces overall space usage and keeps the size of the task_struct down
(only one pointer added to task_struct compared to a non-containers
kernel).

Signed-off-by: Paul Menage <menage@google.com>

---
 include/linux/container.h | 35 ++++++
 include/linux/sched.h | 30 -----
 kernel/container.c | 248 ++++++++++++++++++++++++++++++++++++---------
 3 files changed, 238 insertions(+), 75 deletions(-)

Index: container-2.6.21-rc7-mm1/include/linux/container.h
 == =======
--- container-2.6.21-rc7-mm1.orig/include/linux/container.h
+++ container-2.6.21-rc7-mm1/include/linux/container.h
@@ -29,6 +29,14 @@ extern void container_unlock(void);

 struct containerfs_root;

+/* Define the enumeration of all container subsystems */
+#define SUBSYS(_x) _x ## _subsys_id,
+enum container_subsys_id {
+#include <linux/container_subsys.h>
+ CONTAINER_SUBSYS_COUNT
+};
+#undef SUBSYS
+
 /* Per-subsystem/per-container state maintained by the system. */
 struct container_subsys_state {

```
  /* The container that this subsystem is attached to. Useful
@@ -87,6 +95,31 @@ struct container {
  struct container *top_container;
 };

+/* A css_group is a structure holding pointers to a set of
+ * container_subsys_state objects. This saves space in the task struct
+ * object and speeds up fork()/exit(), since a single inc/dec can bump
+ * the reference count on the entire container set for a task.
+ */
+
+struct css_group {
+
+ /* Reference count */
+ struct kref ref;
+
+ /* List running through all container groups */
+ struct list_head list;
+
+ /* Set of subsystem states, one for each subsystem. NULL for
+  * subsystems that aren't part of this hierarchy. These
+  * pointers reduce the number of dereferences required to get
+  * from a task to its state for a given container, but result
+  * in increased space usage if tasks are in wildly different
+  * groupings across different hierarchies. This array is
+  * immutable after creation */
+ struct container_subsys_state *subsys[CONTAINER_SUBSYS_COUNT];
+
+};
+
 /* struct cftype:
  *
  * The files in the container filesystem mostly have a very simple read/write
@@ -178,7 +211,7 @@ static inline struct container_subsys_st
 static inline struct container_subsys_state *task_subsys_state(
  struct task_struct *task, int subsys_id)
 {
- return rcu_dereference(task->containers.subsys[subsys_id]);
+ return rcu_dereference(task->containers->subsys[subsys_id]);
 }

 static inline struct container* task_container(struct task_struct *task,
Index: container-2.6.21-rc7-mm1/include/linux/sched.h
 ===================================================================== =======
--- container-2.6.21-rc7-mm1.orig/include/linux/sched.h
+++ container-2.6.21-rc7-mm1/include/linux/sched.h
@@ -818,34 +818,6 @@ struct uts_namespace;
```

```
 struct prio_array;

-#ifdef CONFIG_CONTAINERS
-
-#define SUBSYS(_x) _x ## _subsys_id,
-enum container_subsys_id {
-#include <linux/container_subsys.h>
- CONTAINER_SUBSYS_COUNT
-};
-#undef SUBSYS
-
-/* A css_group is a structure holding pointers to a set of
- * container_subsys_state objects.
- */
-
-struct css_group {
-
- /* Set of subsystem states, one for each subsystem. NULL for
-  * subsystems that aren't part of this hierarchy. These
-  * pointers reduce the number of dereferences required to get
-  * from a task to its state for a given container, but result
-  * in increased space usage if tasks are in wildly different
-  * groupings across different hierarchies. This array is
-  * immutable after creation */
- struct container_subsys_state *subsys[CONTAINER_SUBSYS_COUNT];
-
-};
-
-#endif /* CONFIG_CONTAINERS */
-
 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
  struct thread_info *thread_info;
@@ -1098,7 +1070,7 @@ struct task_struct {
  int cpuset_mem_spread_rotor;
 #endif
 #ifdef CONFIG_CONTAINERS
- struct css_group containers;
+ struct css_group *containers;
 #endif
  struct robust_list_head __user *robust_list;
 #ifdef CONFIG_COMPAT
Index: container-2.6.21-rc7-mm1/kernel/container.c
 ===================================================================== =======
--- container-2.6.21-rc7-mm1.orig/kernel/container.c
+++ container-2.6.21-rc7-mm1/kernel/container.c
@@ -132,12 +132,29 @@ list_for_each_entry(_ss, &_root->subsys_
 #define for_each_root(_root) \
```

```
 list_for_each_entry(_root, &roots, root_list)

-/* Each task_struct has an embedded css_group, so the get/put
- * operation simply takes a reference count on all the containers
- * referenced by subsystems in this css_group. This can end up
- * multiple-counting some containers, but that's OK - the ref-count is
- * just a busy/not-busy indicator; ensuring that we only count each
- * container once would require taking a global lock to ensure that no
+/* The default css_group - used by init and its children prior to any
+ * hierarchies being mounted. It contains a pointer to the root state
+ * for each subsystem. Also used to anchor the list of css_groups. Not
+ * reference-counted, to improve performance when child containers
+ * haven't been created.
+ */
+
+static struct css_group init_css_group;
+
+/*
+ * This lock nests inside tasklist lock, which can be taken by
+ * interrupts, and hence always needs to be taken with
+ * spin_lock_irq*
+ */
+static DEFINE_SPINLOCK(css_group_lock);
+static int css_group_count;
+
+/* When we create or destroy a css_group, the operation simply
+ * takes/releases a reference count on all the containers referenced
+ * by subsystems in this css_group. This can end up multiple-counting
+ * some containers, but that's OK - the ref-count is just a
+ * busy/not-busy indicator; ensuring that we only count each container
+ * once would require taking a global lock to ensure that no
  * subsystems moved between hierarchies while we were doing so.
  *
  * Possible TODO: decide at boot time based on the number of
@@ -146,18 +163,140 @@ list_for_each_entry(_root, &roots, root_
  * take a global lock and only add one ref count to each hierarchy.
  */

-static void get_css_group(struct css_group *cg) {
+/*
+ * unlink a css_group from the list and free it
+ */
+static void release_css_group(struct kref *k) {
+ struct css_group *cg =
+  container_of(k, struct css_group, ref);
+ unsigned long flags;
  int i;
+ spin_lock_irqsave(&css_group_lock, flags);
```

```
+ list_del(&cg->list);
+ css_group_count--;
+ spin_unlock_irqrestore(&css_group_lock, flags);
  for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
-  atomic_inc(&cg->subsys[i]->container->count);
+  atomic_dec(&cg->subsys[i]->container->count);
  }
+ kfree(cg);
+}
+
+/*
+ * refcounted get/put for css_group objects
+ */
+static inline void get_css_group(struct css_group *cg) {
+ if (cg != &init_css_group)
+  kref_get(&cg->ref);
+}
+
+static inline void put_css_group(struct css_group *cg) {
+ if (cg != &init_css_group)
+  kref_put(&cg->ref, release_css_group);
 }

-static void put_css_group(struct css_group *cg) {
+/*
+ * find_existing_css_group() is a helper for
+ * find_css_group(), and checks to see whether an existing
+ * css_group is suitable. This currently walks a linked-list for
+ * simplicity; a later patch will use a hash table for better
+ * performance
+ *
+ * oldcg: the container group that we're using before the container
+ * transition
+ *
+ * cont: the container that we're moving into
+ *
+ * template: location in which to build the desired set of subsystem
+ * state objects for the new container group
+ */
+
+static struct css_group *find_existing_css_group(
+ struct css_group *oldcg,
+ struct container *cont,
+ struct container_subsys_state *template[])
+{
  int i;
+ struct containerfs_root *root = cont->root;
+ struct list_head *l = &init_css_group.list;
```

```
+
+ /* Built the set of subsystem state objects that we want to
+  * see in the new css_group */
  for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
-  atomic_dec(&cg->subsys[i]->container->count);
+  if (root->subsys_bits & (1ull << i)) {
+   /* Subsystem is in this hierarchy. So we want
+    * the subsystem state from the new
+    * container */
+   template[i] = cont->subsys[i];
+  } else {
+   /* Subsystem is not in this hierarchy, so we
+    * don't want to change the subsystem state */
+   template[i] = oldcg->subsys[i];
+  }
+ }
+
+ /* Look through existing container groups to find one to reuse */
+ do {
+  struct css_group *cg =
+   list_entry(l, struct css_group, list);
+
+  if (!memcmp(template, oldcg->subsys, sizeof(oldcg->subsys))) {
+   /* All subsystems matched */
+   return cg;
+  }
+  /* Try the next container group */
+  l = l->next;
+ } while (l != &init_css_group.list);
+
+ /* No existing container group matched */
+ return NULL;
+}
+
+/*
+ * find_css_group() takes an existing container group and a
+ * container object, and returns a css_group object that's
+ * equivalent to the old group, but with the given container
+ * substituted into the appropriate hierarchy. Must be called with
+ * container_mutex held
+ */
+
+static struct css_group *find_css_group(
+ struct css_group *oldcg, struct container *cont)
+{
+ struct css_group *res;
+ struct container_subsys_state *template[CONTAINER_SUBSYS_COUNT];
+ int i;
```

```
+
+ /* First see if we already have a container group that matches
+  * the desired set */
+ spin_lock_irq(&css_group_lock);
+ res = find_existing_css_group(oldcg, cont, template);
+ if (res)
+  get_css_group(res);
+ spin_unlock_irq(&css_group_lock);
+
+ if (res)
+  return res;
+
+ res = kmalloc(sizeof(*res), GFP_KERNEL);
+ if (!res)
+  return NULL;
+
+ kref_init(&res->ref);
+ /* Copy the set of subsystem state objects generated in
+  * find_existing_css_group() */
+ memcpy(res->subsys, template, sizeof(res->subsys));
+ /* Add reference counts from the new css_group */
+ for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
+  atomic_inc(&res->subsys[i]->container->count);
  }
+
+ /* Link this container group into the list */
+ spin_lock_irq(&css_group_lock);
+ list_add(&res->list, &init_css_group.list);
+ css_group_count++;
+ spin_unlock_irq(&css_group_lock);
+
+ return res;
 }

 /*
@@ -717,9 +856,9 @@ static int attach_task(struct container
  int retval = 0;
  struct container_subsys *ss;
  struct container *oldcont;
- struct css_group *cg = &tsk->containers;
+ struct css_group *cg = tsk->containers;
+ struct css_group *newcg;
  struct containerfs_root *root = cont->root;
- int i;

  int subsys_id;
  get_first_subsys(cont, NULL, &subsys_id);
@@ -738,24 +877,20 @@ static int attach_task(struct container
```

```
   }
  }

+ /* Locate or allocate a new css_group for this task,
+  * based on its final set of containers */
+ newcg = find_css_group(cg, cont);
+ if (!newcg) {
+  return -ENOMEM;
+ }
+
  task_lock(tsk);
  if (tsk->flags & PF_EXITING) {
   task_unlock(tsk);
+  put_css_group(newcg);
   return -ESRCH;
  }
- /* Update the css_group pointers for the subsystems in this
-  * hierarchy */
- for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
-  if (root->subsys_bits & (1ull << i)) {
-   /* Subsystem is in this hierarchy. So we want
-    * the subsystem state from the new
-    * container. Transfer the refcount from the
-    * old to the new */
-   atomic_inc(&cont->count);
-   atomic_dec(&cg->subsys[i]->container->count);
-   rcu_assign_pointer(cg->subsys[i], cont->subsys[i]);
-  }
- }
+ rcu_assign_pointer(tsk->containers, newcg);
  task_unlock(tsk);

  for_each_subsys(root, ss) {
@@ -765,6 +900,7 @@ static int attach_task(struct container
  }

  synchronize_rcu();
+ put_css_group(cg);
  return 0;
 }

@@ -1431,8 +1567,9 @@ static int container_rmdir(struct inode

 static void container_init_subsys(struct container_subsys *ss) {
  int retval;
- struct task_struct *g, *p;
  struct container_subsys_state *css;
+ unsigned long flags;
```

```
+ struct list_head *l;
  printk(KERN_ERR "Initializing container subsys %s\n", ss->name);

  /* Create the top container state for this subsystem */
@@ -1443,26 +1580,32 @@ static void container_init_subsys(struct
  init_container_css(ss, dummytop);
  css = dummytop->subsys[ss->subsys_id];

- /* Update all tasks to contain a subsys pointer to this state
-  * - since the subsystem is newly registered, all tasks are in
-  * the subsystem's top container. */
+ /* Update all container groups to contain a subsys
+  * pointer to this state - since the subsystem is
+  * newly registered, all tasks and hence all container
+  * groups are in the subsystem's top container. */
+ spin_lock_irqsave(&css_group_lock, flags);
+ l = &init_css_group.list;
+ do {
+  struct css_group *cg =
+   list_entry(l, struct css_group, list);
+  cg->subsys[ss->subsys_id] = dummytop->subsys[ss->subsys_id];
+  l = l->next;
+ } while (l != &init_css_group.list);
+ spin_unlock_irqrestore(&css_group_lock, flags);

   /* If this subsystem requested that it be notified with fork
    * events, we should send it one now for every process in the
    * system */
+  if (ss->fork) {
+   struct task_struct *g, *p;

- read_lock(&tasklist_lock);
- init_task.containers.subsys[ss->subsys_id] = css;
- if (ss->fork)
-  ss->fork(ss, &init_task);
-
- do_each_thread(g, p) {
-  printk(KERN_INFO "Setting task %p css to %p (%d)\n", css, p, p->pid);
-  p->containers.subsys[ss->subsys_id] = css;
-  if (ss->fork)
-   ss->fork(ss, p);
- } while_each_thread(g, p);
- read_unlock(&tasklist_lock);
+   read_lock(&tasklist_lock);
+   do_each_thread(g, p) {
+    ss->fork(ss, p);
+   } while_each_thread(g, p);
+   read_unlock(&tasklist_lock);
```

```
+  }

  need_forkexit_callback |= ss->fork || ss->exit;

@@ -1478,7 +1621,12 @@ static void container_init_subsys(struct
 int __init container_init_early(void)
 {
  int i;
+ kref_init(&init_css_group.ref);
+ kref_get(&init_css_group.ref);
+ INIT_LIST_HEAD(&init_css_group.list);
+ css_group_count = 1;
  init_container_root(&rootnode);
+ init_task.containers = &init_css_group;

  for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
   struct container_subsys *ss = subsys[i];
@@ -1635,6 +1783,7 @@ static int proc_containerstats_show(stru
   seq_printf(m, "%d: name=%s hierarchy=%p\n",
       i, ss->name, ss->root);
  }
+ seq_printf(m, "Container groups: %d\n", css_group_count);
  mutex_unlock(&container_mutex);
  return 0;
 }
@@ -1671,7 +1820,7 @@ void container_fork(struct task_struct *
 {
  rcu_read_lock();
  child->containers = rcu_dereference(current->containers);
- get_css_group(&child->containers);
+ get_css_group(child->containers);
  rcu_read_unlock();
 }

@@ -1733,6 +1882,7 @@ void container_fork_callbacks(struct tas
 void container_exit(struct task_struct *tsk, int run_callbacks)
 {
  int i;
+ struct css_group *cg = NULL;
  if (run_callbacks && need_forkexit_callback) {
   for (i = 0; i < CONTAINER_SUBSYS_COUNT; i++) {
    struct container_subsys *ss = subsys[i];
@@ -1743,9 +1893,13 @@ void container_exit(struct task_struct *
  }
  /* Reassign the task to the init_css_group. */
  task_lock(tsk);
- put_css_group(&tsk->containers);
- tsk->containers = init_task.containers;
```

```
+ if (tsk->containers != &init_css_group) {
+  cg = tsk->containers;
+  tsk->containers = &init_css_group;
+ }
  task_unlock(tsk);
+ if (cg)
+  put_css_group(cg);
 }

 static atomic_t namecnt;
@@ -1783,11 +1937,13 @@ int container_clone(struct task_struct *
   mutex_unlock(&container_mutex);
   return 0;
  }
- cg = &tsk->containers;
+ cg = tsk->containers;
  parent = task_container(tsk, subsys->subsys_id);
  /* Pin the hierarchy */
  atomic_inc(&parent->root->sb->s_active);

+ /* Keep the container alive */
+ get_css_group(cg);
  mutex_unlock(&container_mutex);

  /* Now do the VFS work to create a container */
@@ -1832,6 +1988,7 @@ int container_clone(struct task_struct *
      (parent != task_container(tsk, subsys->subsys_id))) {
   /* Aargh, we raced ... */
   mutex_unlock(&inode->i_mutex);
+  put_css_group(cg);

   deactivate_super(parent->root->sb);
   /* The container is still accessible in the VFS, but
@@ -1849,6 +2006,7 @@ int container_clone(struct task_struct *

  out_release:
  mutex_unlock(&inode->i_mutex);
+ put_css_group(cg);
  deactivate_super(parent->root->sb);
  return ret;
 }

--
```