Subject: [PATCH 04/10] Task Containers(V11): Add container_clone() interface.
Posted by Paul Menage on Fri, 20 Jul 2007 18:31:56 GMT
View Forum Message <> Reply to Message

This patch adds support for container_clone(), a way to create new
containers intended to be used for systems such as namespace
unsharing. A new subsystem callback, post_clone(), is added to allow
subsystems to automatically configure cloned containers.

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

 Documentation/containers.txt | 7 ++
 include/linux/container.h | 3
 kernel/container.c | 135 ++
 3 files changed, 145 insertions(+)

Index: container-2.6.22-rc6-mm1/include/linux/container.h
 === =======
--- container-2.6.22-rc6-mm1.orig/include/linux/container.h
+++ container-2.6.22-rc6-mm1/include/linux/container.h
@@ -174,6 +174,7 @@ struct container_subsys {
  void (*exit)(struct container_subsys *ss, struct task_struct *task);
  int (*populate)(struct container_subsys *ss,
    struct container *cont);
+ void (*post_clone)(struct container_subsys *ss, struct container *cont);
  void (*bind)(struct container_subsys *ss, struct container *root);
  int subsys_id;
  int active;
@@ -213,6 +214,8 @@ static inline struct container* task_con

 int container_path(const struct container *cont, char *buf, int buflen);

+int container_clone(struct task_struct *tsk, struct container_subsys *ss);
+
 #else /* !CONFIG_CONTAINERS */

 static inline int container_init_early(void) { return 0; }
Index: container-2.6.22-rc6-mm1/kernel/container.c
 === =======
--- container-2.6.22-rc6-mm1.orig/kernel/container.c
+++ container-2.6.22-rc6-mm1/kernel/container.c
@@ -1675,3 +1675,138 @@ void container_exit(struct task_struct *
  tsk->containers = init_task.containers;
  task_unlock(tsk);
 }
+
+/**

```
+ * container_clone - duplicate the current container in the hierarchy
+ * that the given subsystem is attached to, and move this task into
+ * the new child
+ */
+int container_clone(struct task_struct *tsk, struct container_subsys *subsys)
+{
+ struct dentry *dentry;
+ int ret = 0;
+ char nodename[MAX_CONTAINER_TYPE_NAMELEN];
+ struct container *parent, *child;
+ struct inode *inode;
+ struct css_group *cg;
+ struct containerfs_root *root;
+ struct container_subsys *ss;
+
+ /* We shouldn't be called by an unregistered subsystem */
+ BUG_ON(!subsys->active);
+
+ /* First figure out what hierarchy and container we're dealing
+  * with, and pin them so we can drop container_mutex */
+ mutex_lock(&container_mutex);
+ again:
+ root = subsys->root;
+ if (root == &rootnode) {
+  printk(KERN_INFO
+         "Not cloning container for unused subsystem %s\n",
+         subsys->name);
+  mutex_unlock(&container_mutex);
+  return 0;
+ }
+ cg = &tsk->containers;
+ parent = task_container(tsk, subsys->subsys_id);
+
+ snprintf(nodename, MAX_CONTAINER_TYPE_NAMELEN, "node_%d", tsk->pid);
+
+ /* Pin the hierarchy */
+ atomic_inc(&parent->root->sb->s_active);
+
+ mutex_unlock(&container_mutex);
+
+ /* Now do the VFS work to create a container */
+ inode = parent->dentry->d_inode;
+
+ /* Hold the parent directory mutex across this operation to
+  * stop anyone else deleting the new container */
+ mutex_lock(&inode->i_mutex);
+ dentry = container_get_dentry(parent->dentry, nodename);
+ if (IS_ERR(dentry)) {
```

```
+  printk(KERN_INFO
+         "Couldn't allocate dentry for %s: %ld\n", nodename,
+         PTR_ERR(dentry));
+  ret = PTR_ERR(dentry);
+  goto out_release;
+ }
+
+ /* Create the container directory, which also creates the container */
+ ret = vfs_mkdir(inode, dentry, S_IFDIR | 0755);
+ child = __d_cont(dentry);
+ dput(dentry);
+ if (ret) {
+  printk(KERN_INFO
+         "Failed to create container %s: %d\n", nodename,
+         ret);
+  goto out_release;
+ }
+
+ if (!child) {
+  printk(KERN_INFO
+         "Couldn't find new container %s\n", nodename);
+  ret = -ENOMEM;
+  goto out_release;
+ }
+
+ /* The container now exists. Retake container_mutex and check
+  * that we're still in the same state that we thought we
+  * were. */
+ mutex_lock(&container_mutex);
+ if ((root != subsys->root) ||
+     (parent != task_container(tsk, subsys->subsys_id))) {
+  /* Aargh, we raced ... */
+  mutex_unlock(&inode->i_mutex);
+
+  deactivate_super(parent->root->sb);
+  /* The container is still accessible in the VFS, but
+   * we're not going to try to rmdir() it at this
+   * point. */
+  printk(KERN_INFO
+         "Race in container_clone() - leaking container %s\n",
+         nodename);
+  goto again;
+ }
+
+ /* do any required auto-setup */
+ for_each_subsys(root, ss) {
+  if (ss->post_clone)
+   ss->post_clone(ss, child);
```

```
+ }
+
+ /* All seems fine. Finish by moving the task into the new container */
+ ret = attach_task(child, tsk);
+ mutex_unlock(&container_mutex);
+
+ out_release:
+ mutex_unlock(&inode->i_mutex);
+ deactivate_super(parent->root->sb);
+ return ret;
+}
+
+/*
+ * See if "cont" is a descendant of the current task's container in
+ * the appropriate hierarchy
+ *
+ * If we are sending in dummytop, then presumably we are creating
+ * the top container in the subsystem.
+ *
+ * Called only by the ns (nsproxy) container.
+ */
+int container_is_descendant(const struct container *cont)
+{
+ int ret;
+ struct container *target;
+ int subsys_id;
+
+ if (cont == dummytop)
+  return 1;
+
+ get_first_subsys(cont, NULL, &subsys_id);
+ target = task_container(current, subsys_id);
+ while (cont != target && cont!= cont->top_container)
+  cont = cont->parent;
+ ret = (cont == target);
+ return ret;
+}
Index: container-2.6.22-rc6-mm1/Documentation/containers.txt
 ===================================================================
--- container-2.6.22-rc6-mm1.orig/Documentation/containers.txt
+++ container-2.6.22-rc6-mm1/Documentation/containers.txt
@@ -504,6 +504,13 @@ include/linux/container.h for details).
 method can return an error code, the error code is currently not
 always handled well.

+void post_clone(struct container_subsys *ss, struct container *cont)
+
+Called at the end of container_clone() to do any paramater
```

+initialization which might be required before a task could attach. For
+example in cpusets, no task may attach before 'cpus' and 'mems' are set
+up.
+
 void bind(struct container_subsys *ss, struct container *root)
 LL=callback_mutex

--