Subject: [PATCH 05/29] task containersv11 add container_clone interface
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:44 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Add support for cgroup_clone(), a way to create new cgroups 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
cgroups.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

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

diff -puN Documentation/cgroups.txt~task-cgroupsv11-add-cgroup_clone-interface Documentation/cgroups.txt
--- a/Documentation/cgroups.txt~task-cgroupsv11-add-cgroup_clone-interface
+++ a/Documentation/cgroups.txt
@@ -504,6 +504,13 @@ include/linux/cgroup.h for details).
method can return an error code, the error code is currently not
always handled well.

+void post_clone(struct cgroup_subsys *ss, struct cgroup *cont)
+
+Called at the end of cgroup_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 cgroup_subsys *ss, struct cgroup *root)
LL=callback_mutex

diff -puN include/linux/cgroup.h~task-cgroupsv11-add-cgroup_clone-interface

```
include/linux/cgroup.h
--- a/include/linux/cgroup.h~task-cgroupsv11-add-cgroup_clone-interface
+++ a/include/linux/cgroup.h
@@ -174,6 +174,7 @@ struct cgroup_subsys {
  void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
  int (*populate)(struct cgroup_subsys *ss,
    struct cgroup *cont);
+ void (*post_clone)(struct cgroup_subsys *ss, struct cgroup *cont);
  void (*bind)(struct cgroup_subsys *ss, struct cgroup *root);
  int subsys_id;
  int active;
@@ -213,6 +214,8 @@ static inline struct cgroup* task_con

 int cgroup_path(const struct cgroup *cont, char *buf, int buflen);

+int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *ss);
+
 #else /* !CONFIG_CGROUPS */

 static inline int cgroup_init_early(void) { return 0; }
diff -puN kernel/cgroup.c~task-cgroupsv11-add-cgroup_clone-interface kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-add-cgroup_clone-interface
+++ a/kernel/cgroup.c
@@ -1684,3 +1684,138 @@ void cgroup_exit(struct task_struct *
  tsk->cgroups = init_task.cgroups;
  task_unlock(tsk);
 }
+
+/**
+ * cgroup_clone - duplicate the current cgroup in the hierarchy
+ * that the given subsystem is attached to, and move this task into
+ * the new child
+ */
+int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *subsys)
+{
+ struct dentry *dentry;
+ int ret = 0;
+ char nodename[MAX_CGROUP_TYPE_NAMELEN];
+ struct cgroup *parent, *child;
+ struct inode *inode;
+ struct css_set *cg;
+ struct cgroupfs_root *root;
+ struct cgroup_subsys *ss;
+
+ /* We shouldn't be called by an unregistered subsystem */
+ BUG_ON(!subsys->active);
+
+ /* First figure out what hierarchy and cgroup we're dealing
```

```
+  * with, and pin them so we can drop cgroup_mutex */
+ mutex_lock(&cgroup_mutex);
+ again:
+ root = subsys->root;
+ if (root == &rootnode) {
+  printk(KERN_INFO
+         "Not cloning cgroup for unused subsystem %s\n",
+         subsys->name);
+  mutex_unlock(&cgroup_mutex);
+  return 0;
+ }
+ cg = &tsk->cgroups;
+ parent = task_cgroup(tsk, subsys->subsys_id);
+
+ snprintf(nodename, MAX_CGROUP_TYPE_NAMELEN, "node_%d", tsk->pid);
+
+ /* Pin the hierarchy */
+ atomic_inc(&parent->root->sb->s_active);
+
+ mutex_unlock(&cgroup_mutex);
+
+ /* Now do the VFS work to create a cgroup */
+ inode = parent->dentry->d_inode;
+
+ /* Hold the parent directory mutex across this operation to
+  * stop anyone else deleting the new cgroup */
+ mutex_lock(&inode->i_mutex);
+ dentry = cgroup_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 cgroup directory, which also creates the cgroup */
+ ret = vfs_mkdir(inode, dentry, S_IFDIR | 0755);
+ child = __d_cont(dentry);
+ dput(dentry);
+ if (ret) {
+  printk(KERN_INFO
+         "Failed to create cgroup %s: %d\n", nodename,
+         ret);
+  goto out_release;
+ }
+
+ if (!child) {
```

```
+  printk(KERN_INFO
+         "Couldn't find new cgroup %s\n", nodename);
+  ret = -ENOMEM;
+  goto out_release;
+ }
+
+ /* The cgroup now exists. Retake cgroup_mutex and check
+  * that we're still in the same state that we thought we
+  * were. */
+ mutex_lock(&cgroup_mutex);
+ if ((root != subsys->root) ||
+     (parent != task_cgroup(tsk, subsys->subsys_id))) {
+  /* Aargh, we raced ... */
+  mutex_unlock(&inode->i_mutex);
+
+  deactivate_super(parent->root->sb);
+  /* The cgroup is still accessible in the VFS, but
+   * we're not going to try to rmdir() it at this
+   * point. */
+  printk(KERN_INFO
+         "Race in cgroup_clone() - leaking cgroup %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 cgroup */
+ ret = attach_task(child, tsk);
+ mutex_unlock(&cgroup_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 cgroup in
+ * the appropriate hierarchy
+ *
+ * If we are sending in dummytop, then presumably we are creating
+ * the top cgroup in the subsystem.
+ *
```

```
+ * Called only by the ns (nsproxy) cgroup.
+ */
+int cgroup_is_descendant(const struct cgroup *cont)
+{
+ int ret;
+ struct cgroup *target;
+ int subsys_id;
+
+ if (cont == dummytop)
+  return 1;
+
+ get_first_subsys(cont, NULL, &subsys_id);
+ target = task_cgroup(current, subsys_id);
+ while (cont != target && cont!= cont->top_cgroup)
+  cont = cont->parent;
+ ret = (cont == target);
+ return ret;
+}
```

_

--

_____

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