

---

Subject: [PATCH 05/10] Containers(V10): Add container\_clone() interface

Posted by [Paul Menage](#) on Tue, 29 May 2007 13:01:09 GMT

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

---

This patch adds support for container\_clone(), a speculative interface to creating new containers intended to be used for systems such as namespace unsharing.

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

---  
include/linux/container.h | 2  
kernel/container.c | 123 +++  
2 files changed, 125 insertions(+)

Index: container-2.6.22-rc2-mm1/kernel/container.c

```
=====
--- container-2.6.22-rc2-mm1.orig/kernel/container.c
+++ container-2.6.22-rc2-mm1/kernel/container.c
@@ -1616,3 +1616,126 @@ void container_exit(struct task_struct *
     tsk->containers = init_task.containers;
     task_unlock(tsk);
 }
+
+static atomic_t namecnt;
+static void get_unused_name(char *buf)
+{
+    sprintf(buf, "node%d", atomic_inc_return(&namecnt));
+}
+
+/**
+ * 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[32];
+    struct container *parent, *child;
+    struct inode *inode;
+    struct css_group *cg;
+    struct containerfs_root *root;
+
+    /* 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);
+ /* Pin the hierarchy */
+ atomic_inc(&parent->root->sb->s_active);
+
+ mutex_unlock(&container_mutex);
+
+ /* Now do the VFS work to create a container */
+ get_unused_name(nodename);
+ 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;
+ }
+
+ /* 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 */
+
+int container_is_descendant(const struct container *cont)
+{
+ int ret;
+ struct container *target;
+ int subsys_id;
+ 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-rc2-mm1/include/linux/container.h
=====
--- container-2.6.22-rc2-mm1.orig/include/linux/container.h
+++ container-2.6.22-rc2-mm1/include/linux/container.h
@@ -197,6 +197,8 @@ 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; }

--

```

---

Subject: Re: [PATCH 05/10] Containers(V10): Add container\_clone() interface  
 Posted by [Andrew Morton](#) on Wed, 30 May 2007 07:16:10 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 29 May 2007 06:01:09 -0700 menage@google.com wrote:

```

> This patch adds support for container_clone(), a speculative interface
> to creating new containers intended to be used for systems such as
> namespace unsharing.
>
> ...
>
> +
> +static atomic_t namecnt;
> +static void get_unused_name(char *buf)
> +{
> + sprintf(buf, "node%d", atomic_inc_return(&namecnt));
> +}

```

A stupid thing, but a sufficiently determined attacker could cause this to wrap.

```

> +/**
> + * 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[32];
> + struct container *parent, *child;
> + struct inode *inode;
> + struct css_group *cg;
> + struct containerfs_root *root;
> +
> + /* 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);
> + /* Pin the hierarchy */
> + atomic_inc(&parent->root->sb->s_active);
> +
> + mutex_unlock(&container_mutex);
> +
> + /* Now do the VFS work to create a container */
> + get_unused_name(nodename);
> + 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;

```

Which I guess could cause a failure here.

Perhaps we could go back and rerun the `get_unused_name()` if `EEXIST`, if

we're bothered.

I hope this is the biggest problem ;)

```
> + }
> +
> + /* 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;
> + }
> +
> + /* 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;
> +}
```

---

---

Subject: Re: [PATCH 05/10] Containers(V10): Add container\_clone() interface  
Posted by [serue](#) on Thu, 31 May 2007 19:56:31 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Andrew Morton ([akpm@linux-foundation.org](mailto:akpm@linux-foundation.org)):

> On Tue, 29 May 2007 06:01:09 -0700 [menage@google.com](mailto:menage@google.com) wrote:

>  
>> This patch adds support for container\_clone(), a speculative interface  
>> to creating new containers intended to be used for systems such as  
>> namespace unsharing.

>>

>> ...

>>

>> +

>> +static atomic\_t namecnt;

>> +static void get\_unused\_name(char \*buf)

>> +{

>> + sprintf(buf, "node%d", atomic\_inc\_return(&namecnt));

>> +}

>

> A stupid thing, but a sufficiently determined attacker could cause this to  
> wrap.

Yeah, this was very consciously done as a "just make it work for now" naming system. If we want to stick with this naming, then I suppose we could do a global bitmap.

But imo this naming is not very convenient - it would be nicer if we

- a) allowed users to specify a name (not sure how that would work logistically)
- b) made the namecnt variable for automatically named containers be per-directory. I'd much rather see

/containers/node1/node1

/containers/node2

than

/containers/node1/node3

/containers/node2

(assuming /node2 was created between /node1 and /node1/node1 or /node1/node3)

thanks,  
-serge

---