Subject: Re: [PATCH 3/9] Containers (V9): Add tasks file interface
Posted by Balbir Singh on Tue, 01 May 2007 18:12:33 GMT
View Forum Message <> Reply to Message

> +static int attach_task_by_pid(struct container *cont, char *pidbuf)
> +{
> + pid_t pid;
> + struct task_struct *tsk;
> + int ret;
> +
> + if (sscanf(pidbuf, "%d", &pid) != 1)
> + return -EIO;
> +
> + if (pid) {
> + read_lock(&tasklist_lock);

You could just use rcu_read_lock() and rcu_read_unlock() instead
of read_lock(&tasklist_lock) and read_unlock(&tasklist_lock).

> +
> + tsk = find_task_by_pid(pid);
> + if (!tsk || tsk->flags & PF_EXITING) {
> + read_unlock(&tasklist_lock);
> + return -ESRCH;
> + }
> +
> + get_task_struct(tsk);
> + read_unlock(&tasklist_lock);
> +
> + if ((current->euid) && (current->euid != tsk->uid)
> + && (current->euid != tsk->suid)) {
> + put_task_struct(tsk);
> + return -EACCES;
> + }
> + } else {
> + tsk = current;
> + get_task_struct(tsk);
> + }
> +
> + ret = attach_task(cont, tsk);
> + put_task_struct(tsk);
> + return ret;
> +}
> +
> /* The various types of files and directories in a container file system */
>
> typedef enum {
> @@ -684,6 +789,54 @@ typedef enum {

```
>   FILE_TASKLIST,
>  } container_filetype_t;
>
> +static ssize_t container_common_file_write(struct container *cont,
> +        struct cftype *cft,
> +        struct file *file,
> +        const char __user *userbuf,
> +        size_t nbytes, loff_t *unused_ppos)
> +{
> + container_filetype_t type = cft->private;
> + char *buffer;
> + int retval = 0;
> +
> + if (nbytes >= PATH_MAX)
> +  return -E2BIG;
> +
> + /* +1 for nul-terminator */
> + if ((buffer = kmalloc(nbytes + 1, GFP_KERNEL)) == 0)
> +  return -ENOMEM;
> +
> + if (copy_from_user(buffer, userbuf, nbytes)) {
> +  retval = -EFAULT;
> +  goto out1;
> + }
> + buffer[nbytes] = 0; /* nul-terminate */
> +
> + mutex_lock(&container_mutex);
> +
> + if (container_is_removed(cont)) {
> +  retval = -ENODEV;
> +  goto out2;
> + }
```

Can't we make this check prior to kmalloc() and copy_from_user()?

```
> +int container_task_count(const struct container *cont) {
> + int count = 0;
> + struct task_struct *g, *p;
> + struct container_subsys_state *css;
> + int subsys_id;
> + get_first_subsys(cont, &css, &subsys_id);
> +
> + read_lock(&tasklist_lock);
```

Can be replaced with rcu_read_lock() and rcu_read_unlock()

```
> + do_each_thread(g, p) {
> +  if (task_subsys_state(p, subsys_id) == css)
> +   count ++;
> + } while_each_thread(g, p);
> + read_unlock(&tasklist_lock);
> + return count;
> +}
> +
> +static int pid_array_load(pid_t *pidarray, int npids, struct container *cont)
> +{
> + int n = 0;
> + struct task_struct *g, *p;
> + struct container_subsys_state *css;
> + int subsys_id;
> + get_first_subsys(cont, &css, &subsys_id);
> + rcu_read_lock();
> + read_lock(&tasklist_lock);
```

The read_lock() and read_unlock() are redundant

```
> +
> + do_each_thread(g, p) {
> +  if (task_subsys_state(p, subsys_id) == css) {
> +   pidarray[n++] = pid_nr(task_pid(p));
> +   if (unlikely(n == npids))
> +    goto array_full;
> +  }
> + } while_each_thread(g, p);
> +
> +array_full:
> + read_unlock(&tasklist_lock);
> + rcu_read_unlock();
> + return n;
> +}
> +
```

[snip]

```
> +static int container_tasks_open(struct inode *unused, struct file *file)
> +{
> + struct container *cont = __d_cont(file->f_dentry->d_parent);
> + struct ctr_struct *ctr;
> + pid_t *pidarray;
> + int npids;
> + char c;
> +
> + if (!(file->f_mode & FMODE_READ))
> +  return 0;
> +
```

```
> + ctr = kmalloc(sizeof(*ctr), GFP_KERNEL);
> + if (!ctr)
> +  goto err0;
> +
> + /*
> +  * If container gets more users after we read count, we won't have
> +  * enough space - tough.  This race is indistinguishable to the
> +  * caller from the case that the additional container users didn't
> +  * show up until sometime later on.
> +  */
> + npids = container_task_count(cont);
> + pidarray = kmalloc(npids * sizeof(pid_t), GFP_KERNEL);
> + if (!pidarray)
> +  goto err1;
> +
> + npids = pid_array_load(pidarray, npids, cont);
> + sort(pidarray, npids, sizeof(pid_t), cmppid, NULL);
> +
> + /* Call pid_array_to_buf() twice, first just to get bufsz */
> + ctr->bufsz = pid_array_to_buf(&c, sizeof(c), pidarray, npids) + 1;
> + ctr->buf = kmalloc(ctr->bufsz, GFP_KERNEL);
> + if (!ctr->buf)
> +  goto err2;
> + ctr->bufsz = pid_array_to_buf(ctr->buf, ctr->bufsz, pidarray, npids);
> +
> + kfree(pidarray);
> + file->private_data = ctr;
> + return 0;
> +
> +err2:
> + kfree(pidarray);
> +err1:
> + kfree(ctr);
> +err0:
> + return -ENOMEM;
> +}
> +
```

Any chance we could get a per-container task list? It will
help subsystem writers as well. Alternatively, subsystems
could use the attach_task() callback to track all tasks,
but a per-container list will avoid duplication.

--
Warm Regards,
Balbir Singh