Subject: [RFC/PATCH 0/8]: CGroup Files: Clean up locking and boilerplate
Posted by Paul Menage on Tue, 13 May 2008 06:37:07 GMT
View Forum Message <> Reply to Message

Most of the remaining cgroup control files that implement raw file handlers (those where the userspace pointer/length/ppos are passed through unchanged by cgroups) appear to do so in order to have some control over the kind of locking that their handler uses. In particular, some handlers need to call cgroup_lock() after copying data from userspace and others don't. They also often dynamically allocate memory when it's not strictly needed.

This patchset provides three main features:

1) A new "lockmode" field that indicates what kind of stability/locking guarantees the handler needs from cgroups while it is running. This can be used to reduce the dependency on cgroup_lock(), and prevent its use as a BKL from becoming a contention point as the number of cgroups subsystems grow.

An alternative to this would be to not have cgroups do the locking internally, but to export similar functionality via functions such as

cgroup_lock_rmdir(struct cgroup*)
cgroup_lock_hierarchy(struct cgroup*)
cgroup_lock_attach(struct cgroup*)

that would perform any necessary locking, and recheck !cgroup_removed(). This results in clearer code from the locking point of view, as you can see within the function exactly what locking it relies upon. The downside is that it makes the code more complex in that you can't simply return from the middle of the function, but instead need to save the return code and goto the point in the function that releases the lock. E.g. the difference would be:

```
int some_handler(struct cgroup *cgrp, struct cftype *cft)
{
 if (!do_locked_operation1(cgrp))
  return -EINVAL;
 return do_locked_operation2(cgrp);
}
```

(and setting CFT_LOCK_RMDIR in the cftype descriptor)

versus

```
int some_handler(struct cgroup *cgrp, struct cftype *cft)
{
 int retval = cgroup_lock_rmdir(cgrp);
```

```
 if (retval)
  return retval;
 if (!do_locked_operation1(cgrp)) {
  retval = -EINVAL;
  goto out_unlock;
 }
 retval = do_locked_operation2(cgrp);
out_unlock:
 cgroup_unlock_rmdir(cgrp);
 return retval;
}
```

The latter style is common in the kernel, but my feeling is that the former style is easier to code and to comprehend. I'm interested in what others think.

2) A new "write_string" method which copies the user's data to kernel space and ensures it's nul-terminated, performs any necessary locking/checks, and invokes the handler with the kernelspace buffer

3) Conversion of several raw read/write handlers in cgroup, cpuset, devcgroup and res_counter to use typed handlers and the new locking specifications.

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

--