Subject: Re: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 08:29:08 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-24 at 23:01 -0700, Paul Menage wrote:
> I don't think you need cgroup_signal.h. It's only included in
> cgroup_signal.c, and doesn't really contain any useful definitions
> anyway. You should just use a cgroup_subsys_state object as your state
> object, since you'll never need to do anything with it anyway.
>
> >+static struct cgroup_subsys_state *signal_create(
> >+ struct cgroup_subsys *ss, struct cgroup *cgroup)
> >+{
> >+ struct stateless *dummy;
> >+
> >+ if (!capable(CAP_SYS_ADMIN))
> >+ return ERR_PTR(-EPERM);
>
> This is unnecessary.

OK, removed.

> >+
> + dummy = kzalloc(sizeof(struct stateless), GFP_KERNEL);
> + if (!dummy)
> + return ERR_PTR(-ENOMEM);
> + return &dummy->css;
> +}
>
> This function could be simplified to:
>
> struct cgroup_subsys_state *css;
> css = kzalloc(sizeof(*css), GFP_KERNEL);
> return css ?: ERR_PTR(-ENOMEM);

I kept the if() syntax but used cgroup_subsys_state as suggested. I kept the name "dummy" too to emphasize that except for following the currently-required form we don't really need the state information.

As a side note, I don't think adding state (which signal was/to sent/send) or a fork handling function prevents races. I tend to agree that there are lots of races involving adding/removing tasks to the cgroup where we may not signal "everything". I think that from userspace's perspective we can't solve the races because there will always be a window between releasing whatever lock we use and returning to userspace where new tasks can be added.

IMHO the only way to prevent such races would be to allow userspace to

"lock" a cgroup so that no new tasks may be added. That would block/fail
new forks and prevent writing new pids to the tasks file. Otherwise
userspace must always recheck the list to ensure it didn't get any new
entries...

> >+static int signal_can_attach(struct cgroup_subsys *ss,
> >+ struct cgroup *new_cgroup,
> >+ struct task_struct *task)
> >+{
> >+ return 0;
> >+}
>
> No need for a can_attach() method if it just returns 0 - that's the default.

Removed.

> >+static int signal_kill(struct cgroup *cgroup, int signum)
> >+{
> >+ struct cgroup_iter it;
> >+ struct task_struct *task;
> >+ int retval = 0;
> >+
> >+ cgroup_iter_start(cgroup, &it);
> >+ while ((task = cgroup_iter_next(cgroup, &it))) {
> >+ retval = send_sig(signum, task, 1);
> >+ if (retval)
> >+ break;
> >+ }
> >+ cgroup_iter_end(cgroup, &it);
> >+
> >+ return retval;
> >+}
>
> cgroup_iter_start() takes a read lock - is send_sig() guaranteed not to sleep?

send_sig() -> send_sig_info() hold the tasklist_lock for read and the
task's sighand->siglock spinlock.

> >+static ssize_t signal_write(struct cgroup *cgroup,
> >+ struct cftype *cft,
> >+ struct file *file,
> >+ const char __user *userbuf,
> >+ size_t nbytes, loff_t *unused_ppos)
>
> This should just be a write_u64() method - cgroups will handle the
> copying/parsing for you. See e.g.
> kernel/sched.c:cpu_shares_write_u64()

Sure.

> >+static struct cftype kill_file = {
> >+ .name = "kill",
> >+ .write = signal_write,
> >+ .private = 0,
> >+};
>
> I agree with PaulJ that "signal.send" would be a nicer name for this
> than "signal.kill"

OK.

Cheers,
 -Matt Helsley

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