Subject: [PATCH v8 0/3] cgroups: implement moving a threadgroup's threads atomically with cgroup.procs
Posted by Ben Blum on Tue, 08 Feb 2011 01:35:42 GMT
View Forum Message <> Reply to Message

On Sun, Dec 26, 2010 at 07:09:19AM -0500, Ben Blum wrote:
> On Fri, Dec 24, 2010 at 03:22:26AM -0500, Ben Blum wrote:
> > On Wed, Aug 11, 2010 at 01:46:04AM -0400, Ben Blum wrote:
> > > On Fri, Jul 30, 2010 at 07:56:49PM -0400, Ben Blum wrote:
> > > > This patch series is a revision of http://lkml.org/lkml/2010/6/25/11 .
> > > >
> > > > This patch series implements a write function for the 'cgroup.procs'
> > > > per-cgroup file, which enables atomic movement of multithreaded
> > > > applications between cgroups. Writing the thread-ID of any thread in a
> > > > threadgroup to a cgroup's procs file causes all threads in the group to
> > > > be moved to that cgroup safely with respect to threads forking/exiting.
> > > > (Possible usage scenario: If running a multithreaded build system that
> > > > sucks up system resources, this lets you restrict it all at once into a
> > > > new cgroup to keep it under control.)
> > > >
> > > > Example: Suppose pid 31337 clones new threads 31338 and 31339.
> > > >
> > > > # cat /dev/cgroup/tasks
> > > > ...
> > > > 31337
> > > > 31338
> > > > 31339
> > > > # mkdir /dev/cgroup/foo
> > > > # echo 31337 > /dev/cgroup/foo/cgroup.procs
> > > > # cat /dev/cgroup/foo/tasks
> > > > 31337
> > > > 31338
> > > > 31339
> > > >
> > > > A new lock, called threadgroup_fork_lock and living in signal_struct, is
> > > > introduced to ensure atomicity when moving threads between cgroups. It's
> > > > taken for writing during the operation, and taking for reading in fork()
> > > > around the calls to cgroup_fork() and cgroup_post_fork().
> >
> > Well this time everything here is actually safe and correct, as far as
> > my best efforts and keen eyes can tell. I dropped the per_thread call
> > from the last series in favour of revising the subsystem callback
> > interface. It now looks like this:
> >
> > ss->can_attach()
> > - Thread-independent, possibly expensive/sleeping.
> >
> > ss->can_attach_task()

> - Called per-thread, run with rcu_read so must not sleep.
>
> ss->pre_attach()
> - Thread independent, must be atomic, happens before attach_task.
>
> ss->attach_task()
> - Called per-thread, run with tasklist_lock so must not sleep.
>
> ss->attach()
> - Thread independent, possibly expensive/sleeping, called last.

Okay, so.

I've revamped the cgroup_attach_proc implementation a bunch and this version should be a lot easier on the eyes (and brains). Issues that are addressed:

1) cgroup_attach_proc now iterates over leader->thread_group once, at the very beginning, and puts each task_struct that we want to move into an array, using get_task_struct to make sure they stick around.
 - threadgroup_fork_lock ensures no threads not in the array can appear, and allows us to use signal->nr_threads to determine the size of the array when kmallocing it.
 - This simplifies the rest of the function a bunch, since now we never need to do rcu_read_lock after building the array. All the subsystem callbacks are the same as described just above, but the "can't sleep" restriction is gone, so it's nice and clean.
 - Checking for a race with de_thread (the manoeuvre I refer to as "double-double-toil-and-trouble-check locking") now needs to be done only once, at the beginning (before building the array).

2) The nodemask allocation problem in cpuset is fixed the same way as before - the masks are shared between the three attach callbacks, so are made as static global variables.

3) The introduction of threadgroup_fork_lock in sched.h (specifically, in signal_struct) requires rwsem.h; the new include appears in the first patch. (An alternate plan would be to make it a struct pointer with an incomplete forward declaration and kmalloc/kfree it during housekeeping, but adding an include seems better than that particular complication.) In light of this, the definitions for threadgroup_fork_{read,write}_{un,}lock are also in sched.h.

-- Ben

 Documentation/cgroups/cgroups.txt | 39 ++-
 block/blk-cgroup.c | 18 -

```
 include/linux/cgroup.h          |   10
 include/linux/init_task.h       |    9
 include/linux/sched.h           |   37 +++
 kernel/cgroup.c                 |  454 ++++++++++++++++++++++++++++++++++++-----
 kernel/cgroup_freezer.c         |   26 --
 kernel/cpuset.c                 |  105 +++++-----
 kernel/fork.c                   |   10
 kernel/ns_cgroup.c              |   23 -
 kernel/sched.c                  |   38 ---
 mm/memcontrol.c                 |   18 -
 security/device_cgroup.c        |    3
 13 files changed, 575 insertions(+), 215 deletions(-)
```

_____

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