Subject: Re: [PATCH v8 0/3] cgroups: implement moving a threadgroup's threads atomically with cgroup.procs
Posted by akpm on Wed, 09 Feb 2011 23:10:46 GMT
View Forum Message <> Reply to Message

On Mon, 7 Feb 2011 20:35:42 -0500
Ben Blum <bblum@andrew.cmu.edu> wrote:

> 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().

The above six month old text is the best (and almost the only)
explanation of the rationale for the entire patch series. Is
it still correct and complete?

Assuming "yes", then... how do we determine whether the feature is

sufficiently useful to justify merging and maintaining it? Will people actually use it?

Was there some particular operational situation which led you to think that the kernel should have this capability? If so, please help us out here and lavishly describe it.

_____

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