Subject: Re: [PATCH v8 0/3] cgroups: implement moving a threadgroup's threads atomically with cgroup.procs
Posted by Ben Blum on Thu, 10 Feb 2011 01:36:06 GMT
View Forum Message <> Reply to Message

```
On Thu, Feb 10, 2011 at 10:02:10AM +0900, KAMEZAWA Hiroyuki wrote:
> On Wed, 9 Feb 2011 15:10:46 -0800
> Andrew Morton <akpm@linux-foundation.org> wrote:
>
> > 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?

Yep, it's still fresh. (That's why I kept it around!)

> >
> >
> > 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.
> >
>
> In these months, I saw following questions as
> ==
> Q. I think I put qemu to xxxx cgroup but it never works!
> A. You need to put all threads in qemu to cgroup.
> ==
>
> 'tasks' file is not useful interface for users, I think.
> (Even if users tend to use put-task-before-exec scheme.)
>
>
> IMHO, from user's side of view, 'tasks' file is a mystery.
>
> TID(thread-ID) is one of secrets in Linux + pthread library. For example,
> on RHEL6, to use gettid(), users has to use syscall() directly. And end-user
> may not know about thread-ID which is hidden under pthreads.

I think glibc in general is to blame for the fact that you need to
syscall(__NR_gettid)? Regardless - yes, exposing an interface dealing
with task_structs can be less than perfect for a world that deals in
userland applications.

> IIRC, there are no interface other than /proc/<pid>/tasks which shows all
> thread IDs of a process. But it's not atomic.

I tend to use pgrep, which is a bit of a hassle.

Also, like in the six-month-old-text, many resource-sucking programs
nowadays (web browsers) are multithreaded.

> So, I think it's ok to have 'procs' interface for cgroup if
> overhead/impact of patch is not heavy.
>
> Thanks,

> -Kame

Thanks for the reasoning. ;)

-- Ben