Subject: Re: [PATCH 1/6] containers: Generic container system abstracted from cpusets code
Posted by Paul Jackson on Sun, 31 Dec 2006 05:17:56 GMT
View Forum Message <> Reply to Message

Eric wrote:
> The whole interface that reads out the processes in your task
> grouping looks scary. It takes the tasklist_lock and holds
> it for an indefinite duration.

It doesn't look "indefinite" to me. It reads the 'container'
field of each task struct, and then is done, dropping the lock.

That's got to be one of the lowest cost, most definite duration,
invocations of "do_each_thread(g, p)" in the kernel.

> Although I am curious why this is even needed when
> we have /proc/<pid>/cpuset which gets us the information
> in another way.

The /proc/<pid>/cpuset interface lets you map one pid to its cpuset.

That's the opposite of mapping a cpuset to the set of all pids that
are attached to it.

I suppose one could get all the tasks in a cpuset by doing whatever it
takes for an opendir/readdir/closedir loop over the pid entries
of /proc, and then each pid in the system doing an open/read/close on
its /proc/<pid>/cpuset, and doing a strcmp on its path with the cpuset
path of interest, to see if they match.

What kind of locking is done in the kernel when a user task does an
opendir/readdir/closedir loop over /proc?

In any case, this would be hecka more expensive than the current
quite -definite- "do_each_thread(g, p)" over the task list, with
three system calls per pid. And the 'tasks' file in cpusets is an
existing and valuable feature, which we can't just remove without
serious cause.

> I hate attach_task. Allowing movement of a process from
> one set to another by another process looks like a great way
> to create subtle races. The very long and exhaustive locking
> comments seem to verify this.

The ability to move tasks between cpusets a valued feature for my
customers. Sorry you hate it.

I'll try to make my comments shorter and less exhausting next
time </sarcarsm>.

The locking is difficult, because:
 1) yes, as you note, attach_task() isn't easy,
 2) the cpu and memory placement of a whole set of tasks can be
    changed by a single write system call on some cpuset file,
 2) cpusets is on the critical code path for both the memory
    allocator and task scheduler (controlling where one can
    allocate and schedule), but needs to avoid putting any
    sigificant locks on either of these paths.

> currently I am horrified at what
> currently looks like huge piles of unnecessary complexity in the
> cpuset implementation.

Not much I can do to help you with your horror, sorry.

If you could be more specific on ways to trim the code while
maintaining the API's that we use, then that might be useful.

> that cpusets need to get fixed

Let me know when you have patches.

> Why does any of this code need a user mode helper? I guess
> because of the complicated semantics this doesn't do proper
> reference counting

The cpuset reference counting is just fine, thank-you.

Removing nodes from the bottom of a vfs file system, when one got there
by an unexpected code path, such as task exit, is not easy. Well, for
someone of my limited vfs talents, quite impossible. I had no desire
(nor ability) to replicate in the kernel/cpuset.c code whatever voodoo
it takes to get the vfs locking correct for a rmdir(2) system call.

Using a user mode helper lets this be handled using the ordinary
rmdir(2) system call, with no special vfs locking awareness, from
a separate thread.

... Hope you had a Merry Christmas.

--
                  I won't rest till it's the best ...
                  Programmer, Linux Scalability
                  Paul Jackson <pj@sgi.com> 1.925.600.0401