Subject: Re: [PATCH 1/6] containers: Generic container system abstracted from cpusets code
Posted by ebiederm on Sat, 30 Dec 2006 13:10:32 GMT
View Forum Message <> Reply to Message

Paul Menage <menage@google.com> writes:

> This patch creates a generic process container system based on (and
> parallel top) the cpusets code. At a coarse level it was created by
> copying kernel/cpuset.c, doing s/cpuset/container/g, and stripping out any
> code that was cpuset-specific rather than applicable to any process
> container subsystem.

First thank you for bring the conversation here. Given what you are implementing I rather object to the term containers as that is what we have been using to refer to the aggregate whole and not the individual pieces.

I'm still digesting this but do you think you could make the code pid namespace safe before moving it all over creation.

i.e. pid_nr(task_pid(task)) not task->pid.

I hadn't realized we had any users like the one below left.

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. All it currently needs is the rcu_read_lock. Holding the tasklist_lock looks like a good way to kill performance on a big box. Even hold the cpu for an indefinite duration I find a little worrying but no where near as bad as taking a global lock for an indefinite period of time. Although I am curious why this is even needed when we have /proc/<pid>/cpuset which gets us the information in another way.

This interface really belongs in /proc as it is about managing processes.

The filesystem operations to manage cpusets are a little non-intuitive but once you see what they are they appear usable.

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. For most of the unix API we have avoided things for precisely this reason. Leaving that set of races to the debugging commands in sys_ptrace.

You are putting a pointer into the task_struct for each class
of resource you want to count. Ouch. Andi Kleen was sufficiently
paranoid about the space bloat that we were obliged to introduce
struct nsproxy.

The more I look at this the more this appears to be completely
overkill for process resource control, and currently I am horrified at what
currently looks like huge piles of unnecessary complexity in the
cpuset implementation.

I still need to do some research but at the moment my feeling that
this approach is so wrong that cpusets need to get fixed and nothing
should ever look at cloning them.

Process resource control that looks like a good reason to add some
more unshare flags or some separate syscalls whichever is simpler.
At least that has a simple user interface that is easy to audit.

If nothing else the code needs to find a way to be refactored so
it isn't scary too look at.

Please also next time explain the mechanism you are talking about
using to track processes and don't grandfather it in with oh
this is just a slightly enhanced cpuset. The insanity of this
interface would have been a lot easier to have been spotted
if it had been described more clearly.

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 so you can't implicitly free these things
on the exit of the last task that uses them. That isn't the
unix way and I don't like it. Way over complicated.

Eric

```
> +/*
> + * Load into 'pidarray' up to 'npids' of the tasks using container 'cont'.
> + * Return actual number of pids loaded. No need to task_lock(p)
> + * when reading out p->container, as we don't really care if it changes
> + * on the next cycle, and we are not going to try to dereference it.
> + */
> +static int pid_array_load(pid_t *pidarray, int npids, struct container *cont)
> +{
> + int n = 0;
> + struct task_struct *g, *p;
> +
> + read_lock(&tasklist_lock);
> +
> + do_each_thread(g, p) {
> +  if (p->container == cont) {
> +   pidarray[n++] = p->pid;
> +   if (unlikely(n == npids))
> +    goto array_full;
> +  }
> + } while_each_thread(g, p);
> +
> +array_full:
> + read_unlock(&tasklist_lock);
> + return n;
> +}
```

____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers