Subject: Re: [RFC] cpuset update_cgroup_cpus_allowed
Posted by David Rientjes on Mon, 15 Oct 2007 18:49:02 GMT
View Forum Message <> Reply to Message

On Mon, 15 Oct 2007, Paul Jackson wrote:

> --- 2.6.23-mm1.orig/kernel/cpuset.c 2007-10-14 22:24:56.268309633 -0700
> +++ 2.6.23-mm1/kernel/cpuset.c 2007-10-14 22:34:52.645364388 -0700
> @@ -677,6 +677,64 @@ done:
> }
>
> /*
> + * update_cgroup_cpus_allowed(cont, cpus)
> + *
> + * Keep looping over the tasks in cgroup 'cont', up to 'ntasks'
> + * tasks at a time, setting each task->cpus_allowed to 'cpus',
> + * until all tasks in the cgroup have that cpus_allowed setting.
> + *
> + * The 'set_cpus_allowed()' call cannot be made while holding the
> + * css_set_lock lock embedded in the cgroup_iter_* calls, so we stash
> + * some task pointers, in the tasks[] array on the stack, then drop
> + * that lock (cgroup_iter_end) before looping over the stashed tasks
> + * to update their cpus_allowed fields.
> + *
> + * Making the const 'ntasks' larger would use more stack space (bad),
> + * and reduce the number of cgroup_iter_start/cgroup_iter_end calls
> + * (good). But perhaps more importantly, it could allow any bugs
> + * lurking in the 'need_repeat' looping logic to remain hidden longer.
> + * So keep ntasks rather small, to ensure any bugs in this loop logic
> + * are exposed quickly.
> + */
> +static void update_cgroup_cpus_allowed(struct cgroup *cont, cpumask_t *cpus)
> +{
> + int need_repeat = true;
> +
> + while (need_repeat) {
> + struct cgroup_iter it;
> + const int ntasks = 10;
> + struct task_struct *tasks[ntasks];
> + struct task_struct **p, **q;
> +
> + need_repeat = false;
> + p = tasks;
> +
> + cgroup_iter_start(cont, &it);
> + while (1) {
> +   struct task_struct *t;
> +

```
> +   t = cgroup_iter_next(cont, &it);
> +   if (!t)
> +    break;
> +   if (cpus_equal(*cpus, t->cpus_allowed))
> +    continue;
```

By making this cpus_equal() and not cpus_intersects(), you're trying to make sure that t->cpus_allowed is always equal to *cpus for each task in the iterator.

```
> +   if (p == tasks + ntasks) {
> +    need_repeat = true;
> +    break;
> +   }
> +   get_task_struct(t);
> +   *p++ = t;
> +  }
> +  cgroup_iter_end(cont, &it);
> +
> +  for (q = tasks; q < p; q++) {
> +   set_cpus_allowed(*q, *cpus);
> +   put_task_struct(*q);
> +  }
> + }
> +}
```

Yet by not doing any locking here to prevent a cpu from being hot-unplugged, you can race and allow the hot-unplug event to happen before calling set_cpus_allowed(). That makes this entire function a no-op with set_cpus_allowed() returning -EINVAL for every call, which isn't caught, and no error is reported to userspace.

Now all the tasks in the cpuset have an inconsistent state with respect to their p->cpuset->cpus_allowed, because that was already updated in update_cpumask(). When userspace checks that value via the 'cpus' file, this is the value returned which is actually not true at all for any of the tasks in 'tasks'.

David