Subject: Re: [PATCH] task containersv11 add tasks file interface fix for cpusets
Posted by Paul Menage on Sat, 06 Oct 2007 21:09:52 GMT
View Forum Message <> Reply to Message

On 10/6/07, Paul Jackson <pj@sgi.com> wrote:
> David wrote:
> > It would probably be better to just save references to the tasks.
> >
> > struct cgroup_iter it;
> > struct task_struct *p, **tasks;
> > int i = 0;
> >
> > cgroup_iter_start(cs->css.cgroup, &it);
> > while ((p = cgroup_iter_next(cs->css.cgroup, &it))) {
> > get_task_struct(p);
> > tasks[i++] = p;
> > }
> > cgroup_iter_end(cs->css.cgroup, &it);
>
> Hmmm ... guess I'd have to loop over the cgroup twice, once to count
> them (the 'count' field is not claimed to be accurate) and then again,
> after I've kmalloc'd the tasks[] array, filling in the tasks[] array.
>
> On a big cgroup on a big system, this could easily be thousands of
> iteration loops.

But if userspace has to do it, the effect will be far more expensive.

>
> If I need to close the window all the way, completely solving the race
> condition, then I have the code in kernel/cpuset.c:update_nodemask(),
> which builds an mmarray[] using two loops and some retries if newly
> forked tasks are showing up too rapidly at the same time. The first of
> the two loops is hidden in the cgroup_task_count() call.

In general, the loop inside cgroup_task_count() will only have a
single iteration. (It's iterating across the shared css_set objects,
not across member tasks.

What's wrong with:

```
  allocate a page of task_struct pointers
again:
  need_repeat = false;
  cgroup_iter_start();
  while (cgroup_iter_next()) {
    if (p->cpus_allowed != new_cpumask) {
      store p;
```

```
      if (page is full) {
        need_repeat = true;
        break;
      }
    }
  }
  for each saved task p {
    set_cpus_allowed(p, new_cpumask);
    release p;
  }
  if (need_repeat)
    goto again;
```

Advantages:

- no vmalloc needed
- just one iteration in the case where a cgroup has fewer than 512 members
- additional iterations only need to deal with tasks that don't have
the right cpu mask
- automatically handles fork races

Another option would be to have a cpuset fork callback that forces
p->cpus_allowed to its cpuset's cpus_allowed if another thread is in
the middle of update_cpumask(). Then you don't need to worry about
fork races at all, since all new threads will get the right cpumask.

> Or, if there is a good reason that must remain a spinlock, then the
> smallest amount of new code, and the easiest code to write, would
> perhaps be adding another cgroup callback, called only by cgroup attach
> () requests back to the same group. Then code that wants to do
> something odd, such as cpusets, for what seems like a no-op, can do so.

I'd much rather not perpetuate that broken API requirement. The fact
that cpusets wants this odd behaviour is based on a nasty hack.

Paul