Subject: Re: [PATCH v5 04/14] kmem accounting basic infrastructure
Posted by Tejun Heo on Thu, 18 Oct 2012 19:37:17 GMT
View Forum Message <> Reply to Message

Hello, David.

On Wed, Oct 17, 2012 at 03:08:04PM -0700, David Rientjes wrote:
> > +static int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
> > +{
> > + int ret = -EINVAL;
> > +#ifdef CONFIG_MEMCG_KMEM
> > + struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
> > + /*
> > + * For simplicity, we won't allow this to be disabled. It also can't
> > + * be changed if the cgroup has children already, or if tasks had
> > + * already joined.
> > + *
> > + * If tasks join before we set the limit, a person looking at
> > + * kmem.usage_in_bytes will have no way to determine when it took
> > + * place, which makes the value quite meaningless.
> > + *
> > + * After it first became limited, changes in the value of the limit are
> > + * of course permitted.
> > + *
> > + * Taking the cgroup_lock is really offensive, but it is so far the only
> > + * way to guarantee that no children will appear. There are plenty of
> > + * other offenders, and they should all go away. Fine grained locking
> > + * is probably the way to go here. When we are fully hierarchical, we
> > + * can also get rid of the use_hierarchy check.
>
> Not sure it's so offensive, it's a pretty standard way of ensuring that
> cont->children doesn't get manipulated in a race.

cgroup_lock is inherently one of the outermost locks as it protects
cgroup hierarchy and modifying cgroup hierarchy involves invoking
subsystem callbacks which may grab subsystem locks. Grabbing it
directly from subsystems thus creates high likelihood of creating a
dependency loop and it's nasty to break.

And I'm unsure whether making cgroup locks finer grained would help as
cpuset grabs cgroup_lock to perform actual task migration which would
require the outermost cgroup locking anyway. This one already has
showed up in a couple lockdep warnings involving static_key usages.

A couple days ago, I posted a patchset to remove cgroup_lock usage
from cgroup_freezer and at least cgroup_freezer seems like it was
aiming for the wrong behavior which led to the wrong locking behavior
requiring grabbing cgroup_lock. I can't say whether others will be

that easy tho.

Anyways, so, cgroup_lock is in the process of being unexported and I'd
really like another usage isn't added but maybe that requires larger
changes to memcg and not something which can be achieved here. Dunno.
Will think more about it.

Thanks.

--
tejun