Subject: Re: [PATCH v3 16/28] memcg: kmem controller charge/uncharge infrastructure
Posted by Frederic Weisbecker on Wed, 30 May 2012 13:37:39 GMT
View Forum Message <> Reply to Message

On Wed, May 30, 2012 at 05:06:22PM +0400, Glauber Costa wrote:
> On 05/30/2012 05:04 PM, Frederic Weisbecker wrote:
> >Do you think it's possible that this memcg can be destroyed (like ss->destroy())
> >concurrently?
> >
> >Probably not because there is a synchronize_rcu() in cgroup_diput() so as long
> >as we are in rcu_read_lock() we are fine.
> >
> >OTOH current->mm->owner can exit() right after we fetched its memcg and thus the css_set
> >can be freed concurrently? And then the cgroup itself after we call rcu_read_unlock()
> >due to cgroup_diput().
> >And yet we are doing the mem_cgroup_get() below unconditionally assuming it's
> >always fine to get a reference to it.
> >
> >May be I'm missing something?
> When a cache is created, we grab a reference to the memcg. So after
> the cache is created, no.
>
> When destroy is called, we flush the create queue, so if the cache
> is not created yet, it will just disappear.
>
> I think the only problem that might happen is in the following scenario:
>
> * cache gets created, but ref count is not yet taken
> * memcg disappears
> * we try to inc refcount for a non-existent memcg, and crash.
>
> This would be trivially solvable by grabing the reference earlier.
> But even then, I need to audit this further to make sure it is
> really an issue.

Right. __mem_cgroup_get_kmem_cache() fetches the memcg of the owner
and calls memcg_create_cache_enqueue() which does css_tryget(&memcg->css).
After this tryget I think you're fine. And in-between you're safe against
css_set removal due to rcu_read_lock().

I'm less clear with __mem_cgroup_new_kmem_page() though...