Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children
Posted by Glauber Costa on Mon, 25 Jun 2012 22:36:27 GMT
View Forum Message <> Reply to Message

```
On 06/25/2012 10:29 PM, Tejun Heo wrote:
> Feeling like a nit pervert but..
>
> On Mon, Jun 25, 2012 at 06:15:26PM +0400, Glauber Costa wrote:
>> @@ -287,7 +287,11 @@ struct mem_cgroup {
>>      * Should the accounting and control be hierarchical, per subtree?
>>      */
>>     bool use_hierarchy;
>> - bool kmem_accounted;
>> + /*
>> +  * bit0: accounted by this cgroup
>> +  * bit1: accounted by a parent.
>> +  */
>> + volatile unsigned long kmem_accounted;
>
> Is the volatile declaration really necessary?  Why is it necessary?
> Why no comment explaining it?

Seems to be required by set_bit and friends. gcc will complain if it is
not volatile (take a look at the bit function headers)

>> +
>> +  for_each_mem_cgroup_tree(iter, memcg) {
>> +   struct mem_cgroup *parent;
>
> Blank line between decl and body please.
ok.

>
>> +   if (iter == memcg)
>> +    continue;
>> +   /*
>> +    * We should only have our parent bit cleared if none of
>> +    * ouri parents are accounted. The transversal order of
>
>                                  ^ type
>
>> +    * our iter function forces us to always look at the
>> +    * parents.
>
> Also, it's okay here but the text filling in comments and patch
> descriptions tend to be quite inconsistent.  If you're on emacs, alt-q
> is your friend and I'm sure vim can do text filling pretty nicely too.
```

```
>
>> +    */
>> +   parent = parent_mem_cgroup(iter);
>> +   while (parent && (parent != memcg)) {
>> +    if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
>> +     goto noclear;
>> +
>> +    parent = parent_mem_cgroup(parent);
>> +   }
>
> Better written in for (;;)?  Also, if we're breaking on parent ==
> memcg, can we ever hit NULL parent in the above loop?
```

I can simplify to test parent != memcg only, indeed it is not expected
to be NULL (but if it happens to be due to any kind of bug, we protect
against NULL-dereference, that is why I like to write this way)

```
>> +   continue;
>> +  }
>> + }
>> +out:
>> + mutex_unlock(&set_limit_mutex);
>
> Can we please branch on val != RECOURSE_MAX first?  I'm not even sure
> whether the above conditionals are correct.  If the user updates an
> existing kmem limit, the first test_and_set_bit() returns non-zero, so
> the code proceeds onto clearing KMEM_ACCOUNTED_THIS, which succeeds
> but val == RESOURCE_MAX fails so it doesn't do anything.  If the user
> changes it again, it will set ACCOUNTED_THIS again.  So, changing an
> existing kmem limit toggles KMEM_ACCOUNTED_THIS, which just seems
> wacky to me.
>
```

I will take a look at that tomorrow as well.