
Subject: Re: [PATCH v2 4/5] don't take cgroup_mutex in destroy()

Posted by [Li Zefan](#) on Wed, 25 Apr 2012 08:01:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

Glauber Costa wrote:

> On 04/23/2012 11:31 PM, KAMEZAWA Hiroyuki wrote:

>> (2012/04/24 4:37), Glauber Costa wrote:

>>

>>> Most of the destroy functions are only doing very simple things

>>> like freeing memory.

>>>

>>> The ones who goes through lists and such, already use its own

>>> locking for those.

>>>

>>> * The cgroup itself won't go away until we free it, (after destroy)

>>> * The parent won't go away because we hold a reference count

>>> * There are no more tasks in the cgroup, and the cgroup is declared

>>> dead (cgroup_is_removed() == true)

>>>

>>> [v2: don't cgroup_lock the freezer and blkcg]

>>>

>>> Signed-off-by: Glauber Costa<glommer@parallels.com>

>>> CC: Tejun Heo<tj@kernel.org>

>>> CC: Li Zefan<lizefan@huawei.com>

>>> CC: Kamezawa Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>

>>> CC: Vivek Goyal<vgoyal@redhat.com>

>>> ---

>>> kernel/cgroup.c | 9 +++++-----

>>> 1 files changed, 4 insertions(+), 5 deletions(-)

>>>

>>> diff --git a/kernel/cgroup.c b/kernel/cgroup.c

>>> index 932c318..976d332 100644

>>> --- a/kernel/cgroup.c

>>> +++ b/kernel/cgroup.c

>>> @@ -869,13 +869,13 @@ static void cgroup_diput(struct dentry *dentry, struct inode *inode)

>>> * agent */

>>> synchronize_rcu();

>>>

>>> - mutex_lock(&cgroup_mutex);

>>> /*

>>> * Release the subsystem state objects.

>>> */

>>> for_each_subsys(cgrp->root, ss)

>>> ss->destroy(cgrp);

>>>

>>> + mutex_lock(&cgroup_mutex);

>>> cgrp->root->number_of_cgroups--;

```

>>> mutex_unlock(&cgroup_mutex);
>>>
>>> @@ -3994,13 +3994,12 @@ static long cgroup_create(struct cgroup *parent, struct dentry
*dentry,
>>>
>>> err_destroy:
>>>
>>> + mutex_unlock(&cgroup_mutex);
>>> for_each_subsys(root, ss) {
>>> if (cgrp->subsys[ss->subsys_id])
>>> ss->destroy(cgrp);
>>> }
>>>
>>> - mutex_unlock(&cgroup_mutex);
>>> -
>>> /* Release the reference count that we took on the superblock */
>>> deactivate_super(sb);
>>>
>>> @@ -4349,9 +4348,9 @@ int __init_or_module cgroup_load_subsys(struct cgroup_subsys
*ss)
>>> int ret = cgroup_init_idr(ss, css);
>>> if (ret) {
>>> dummytop->subsys[ss->subsys_id] = NULL;
>>> + mutex_unlock(&cgroup_mutex);
>>> ss->destroy(dummytop);
>>> subsys[i] = NULL;
>>> - mutex_unlock(&cgroup_mutex);
>>> return ret;
>>> }
>>> }
>>> @@ -4447,10 +4446,10 @@ void cgroup_unload_subsys(struct cgroup_subsys *ss)
>>> * pointer to find their state. note that this also takes care of
>>> * freeing the css_id.
>>> */
>>> + mutex_unlock(&cgroup_mutex);
>>> ss->destroy(dummytop);
>>> dummytop->subsys[ss->subsys_id] = NULL;
>>>
>>
>> I'm not fully sure but...dummytop->subsys[] update can be done without locking ?
>>
> I don't see a reason why updates to subsys[] after destruction shouldn't
> be safe. But maybe I am wrong.
>
> Tejun? Li?
>

```

It's safe for dummpytop->subsys[], but it makes the code a bit subtle.

The worst part is, it's not safe to NULLify subsys[i] without cgroup_mutex. It should be ok to do that before calling ->destroy(), but again the code becomes a bit subtler.
