Subject: [RFC] alternative mechanism to skip memcg kmem allocations
Posted by Glauber Costa on Tue, 08 May 2012 03:37:18 GMT
View Forum Message <> Reply to Message

Since Kame expressed the wish to see a context-based method to skip
accounting for caches, I came up with the following proposal for
your appreciation.

It basically works in the same way as preempt_disable()/preempt_enable():
By marking a region under which all allocations will be accounted
to the root memcg.

I basically see two main advantages of it:

 * No need to clutter the code with *_noaccount functions; they could
   become specially widespread if we needed to skip accounting for
   kmalloc variants like track, zalloc, etc.
 * Works with other caches, not only kmalloc; specially interesting
   since during cache creation we touch things like cache_cache,
   that could very well we wrapped inside a noaccount region.

However:

 * It touches task_struct
 * It is harder to keep drivers away from using it. With
   kmalloc_no_account we could simply not export it. Here, one can
   always set this in the task_struct...

Let me know what you think of it.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: Michal Hocko <mhocko@suse.cz>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Suleiman Souhlal <suleiman@google.com>
---
 include/linux/sched.h | 1 +
 mm/memcontrol.c | 34 ++++++++++++++++++++++++++++++++++
 2 files changed, 35 insertions(+), 0 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 81a173c..516a9fe 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1613,6 +1613,7 @@ struct task_struct {
   unsigned long nr_pages; /* uncharged usage */

```
   unsigned long memsw_nr_pages; /* uncharged mem+swap usage */
  } memcg_batch;
+ int memcg_kmem_skip_account;
 #endif
 #ifdef CONFIG_HAVE_HW_BREAKPOINT
  atomic_t ptrace_bp_refcnt;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 8c7c404..833f4cd 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -479,6 +479,33 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 EXPORT_SYMBOL(tcp_proto_cgroup);
 #endif /* CONFIG_INET */

+static void memcg_stop_kmem_account(void)
+{
+ struct task_struct *p;
+
+ if (!current->mm)
+  return;
+
+ p = rcu_dereference(current->mm->owner);
+ if (p) {
+  task_lock(p);
+  p->memcg_kmem_skip_account = true;
+ }
+}
+
+static void memcg_start_kmem_account(void)
+{
+ struct task_struct *p;
+
+ if (!current->mm)
+  return;
+
+ p = rcu_dereference(current->mm->owner);
+ if (p) {
+  p->memcg_kmem_skip_account = false;
+  task_unlock(p);
+ }
+}
 char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct kmem_cache *cachep)
 {
  char *name;
@@ -541,7 +568,9 @@ static struct kmem_cache *memcg_create_kmem_cache(struct
mem_cgroup *memcg,
  if (new_cachep)
   goto out;
```

```
+ memcg_stop_kmem_account();
  new_cachep = kmem_cache_dup(memcg, cachep);
+ memcg_start_kmem_account();

  if (new_cachep == NULL) {
   new_cachep = cachep;
@@ -634,7 +663,9 @@ static void memcg_create_cache_enqueue(struct mem_cgroup *memcg,
  if (!css_tryget(&memcg->css))
   return;

+ memcg_stop_kmem_account();
  cw = kmalloc(sizeof(struct create_work), GFP_NOWAIT);
+ memcg_start_kmem_account();
  if (cw == NULL) {
   css_put(&memcg->css);
   return;
@@ -678,6 +709,9 @@ struct kmem_cache *__mem_cgroup_get_kmem_cache(struct
kmem_cache *cachep,
  VM_BUG_ON(idx == -1);

  p = rcu_dereference(current->mm->owner);
+ if (p->memcg_kmem_skip_account)
+  return cachep;
+
  memcg = mem_cgroup_from_task(p);

  if (!mem_cgroup_kmem_enabled(memcg))
--
1.7.7.6
```

Subject: Re: [RFC] alternative mechanism to skip memcg kmem allocations
Posted by Suleiman Souhlal on Tue, 08 May 2012 20:47:02 GMT
View Forum Message <> Reply to Message

On Mon, May 7, 2012 at 8:37 PM, Glauber Costa <glommer@parallels.com> wrote:
> Since Kame expressed the wish to see a context-based method to skip
> accounting for caches, I came up with the following proposal for
> your appreciation.
>
> It basically works in the same way as preempt_disable()/preempt_enable():
> By marking a region under which all allocations will be accounted
> to the root memcg.
>
> I basically see two main advantages of it:
>
> * No need to clutter the code with *_noaccount functions; they could

> become specially widespread if we needed to skip accounting for
> kmalloc variants like track, zalloc, etc.
> * Works with other caches, not only kmalloc; specially interesting
> since during cache creation we touch things like cache_cache,
> that could very well we wrapped inside a noaccount region.
>
> However:
>
> * It touches task_struct
> * It is harder to keep drivers away from using it. With
> kmalloc_no_account we could simply not export it. Here, one can
> always set this in the task_struct...
>
> Let me know what you think of it.

I like this idea a lot.

>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Christoph Lameter <cl@linux.com>
> CC: Pekka Enberg <penberg@cs.helsinki.fi>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Suleiman Souhlal <suleiman@google.com>
> ---
> include/linux/sched.h | 1 +
> mm/memcontrol.c | 34 ++++++++++++++++++++++++++++++++++
> 2 files changed, 35 insertions(+), 0 deletions(-)
>
> diff --git a/include/linux/sched.h b/include/linux/sched.h
> index 81a173c..516a9fe 100644
> --- a/include/linux/sched.h
> +++ b/include/linux/sched.h
> @@ -1613,6 +1613,7 @@ struct task_struct {
> unsigned long nr_pages; /* uncharged usage */
> unsigned long memsw_nr_pages; /* uncharged mem+swap usage */
> } memcg_batch;
> + int memcg_kmem_skip_account;
> #endif
> #ifdef CONFIG_HAVE_HW_BREAKPOINT
> atomic_t ptrace_bp_refcnt;
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 8c7c404..833f4cd 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -479,6 +479,33 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
> EXPORT_SYMBOL(tcp_proto_cgroup);

> #endif /* CONFIG_INET */
>
> +static void memcg_stop_kmem_account(void)
> +{
> +	struct task_struct *p;
> +
> +	if (!current->mm)
> +		return;
> +
> +	p = rcu_dereference(current->mm->owner);
> +	if (p) {
> +		task_lock(p);
> +		p->memcg_kmem_skip_account = true;
> +	}

This doesn't seem right. The flag has to be set on current, not on another task, or weird things will happen (like the flag getting lost).

Also, we might want to make it a count instead of a boolean, so that it's possible to nest it.

-- Suleiman

Subject: Re: [RFC] alternative mechanism to skip memcg kmem allocations
Posted by Glauber Costa on Tue, 08 May 2012 20:48:08 GMT
View Forum Message <> Reply to Message

On 05/08/2012 05:47 PM, Suleiman Souhlal wrote:
> On Mon, May 7, 2012 at 8:37 PM, Glauber Costa<glommer@parallels.com> wrote:
>> Since Kame expressed the wish to see a context-based method to skip
>> accounting for caches, I came up with the following proposal for
>> your appreciation.
>>
>> It basically works in the same way as preempt_disable()/preempt_enable():
>> By marking a region under which all allocations will be accounted
>> to the root memcg.
>>
>> I basically see two main advantages of it:
>>
>>   * No need to clutter the code with *_noaccount functions; they could
>>   become specially widespread if we needed to skip accounting for
>>   kmalloc variants like track, zalloc, etc.
>>   * Works with other caches, not only kmalloc; specially interesting
>>   since during cache creation we touch things like cache_cache,
>>   that could very well we wrapped inside a noaccount region.
>>

>> However:
>>
>> * It touches task_struct
>> * It is harder to keep drivers away from using it. With
>> kmalloc_no_account we could simply not export it. Here, one can
>> always set this in the task_struct...
>>
>> Let me know what you think of it.
>
> I like this idea a lot.
>
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Christoph Lameter<cl@linux.com>
>> CC: Pekka Enberg<penberg@cs.helsinki.fi>
>> CC: Michal Hocko<mhocko@suse.cz>
>> CC: Kamezawa Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
>> CC: Johannes Weiner<hannes@cmpxchg.org>
>> CC: Suleiman Souhlal<suleiman@google.com>
>> ---
>> include/linux/sched.h | 1 +
>> mm/memcontrol.c | 34 ++++++++++++++++++++++++++++++++++
>> 2 files changed, 35 insertions(+), 0 deletions(-)
>>
>> diff --git a/include/linux/sched.h b/include/linux/sched.h
>> index 81a173c..516a9fe 100644
>> --- a/include/linux/sched.h
>> +++ b/include/linux/sched.h
>> @@ -1613,6 +1613,7 @@ struct task_struct {
>> unsigned long nr_pages; /* uncharged usage */
>> unsigned long memsw_nr_pages; /* uncharged mem+swap usage */
>> } memcg_batch;
>> + int memcg_kmem_skip_account;
>> #endif
>> #ifdef CONFIG_HAVE_HW_BREAKPOINT
>> atomic_t ptrace_bp_refcnt;
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 8c7c404..833f4cd 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -479,6 +479,33 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>> EXPORT_SYMBOL(tcp_proto_cgroup);
>> #endif /* CONFIG_INET */
>>
>> +static void memcg_stop_kmem_account(void)
>> +{
>> + struct task_struct *p;
>> +

```
>> +       if (!current->mm)
>> +               return;
>> +
>> +       p = rcu_dereference(current->mm->owner);
>> +       if (p) {
>> +               task_lock(p);
>> +               p->memcg_kmem_skip_account = true;
>> +       }
>
> This doesn't seem right. The flag has to be set on current, not on
> another task, or weird things will happen (like the flag getting
> lost).
```

Won't get lost if changed to a counter, as you suggested.

As for another task, in follow up patches I will make cache selection based on charges based on mm->owner, instead of current. That's why I did it based on mm->owner.

But thinking again, here, it is somewhat different, who are we charging too doesn't matter that much: what really matters is in which piece of code we're in, so current makes more sense...

will update it.

```
>
> Also, we might want to make it a count instead of a boolean, so that
> it's possible to nest it.
```
but do we want to nest it?