
Subject: [PATCH v5 13/18] memcg/sl[au]b Track all the memcg children of a kmem_cache.

Posted by [Glauber Costa](#) on Fri, 19 Oct 2012 14:20:37 GMT

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

This enables us to remove all the children of a kmem_cache being destroyed, if for example the kernel module it's being used in gets unloaded. Otherwise, the children will still point to the destroyed parent.

Signed-off-by: Suleiman Souhlal <suleiman@google.com>

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: Tejun Heo <tj@kernel.org>

include/linux/memcontrol.h | 5 +++++

mm/memcontrol.c | 32 ++++++++-----

mm/slab_common.c | 3 +++

3 files changed, 38 insertions(+), 2 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

index 9152d49..5c1d234 100644

--- a/include/linux/memcontrol.h

+++ b/include/linux/memcontrol.h

@@ -427,6 +427,7 @@ struct kmem_cache *

__memcg_kmem_get_cache(struct kmem_cache *cachep, gfp_t gfp);

void mem_cgroup_destroy_cache(struct kmem_cache *cachep);

+void kmem_cache_destroy_memcg_children(struct kmem_cache *s);

/**

* memcg_kmem_newpage_charge: verify if a new kmem allocation is allowed.

@@ -576,6 +577,10 @@ memcg_kmem_get_cache(struct kmem_cache *cachep, gfp_t gfp)

{
 return cachep;

}

+

+static inline void kmem_cache_destroy_memcg_children(struct kmem_cache *s)

+{

+}

#endif /* CONFIG_MEMCG_KMEM */

#endif /* _LINUX_MEMCONTROL_H */

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 0359b3a..f5089b3 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

```
@@ -2715,6 +2715,8 @@ static void __mem_cgroup_commit_charge(struct mem_cgroup
*memcg,
    memcg_check_events(memcg, page);
}
```

```
+static DEFINE_MUTEX(set_limit_mutex);
```

```
+
```

```
#ifdef CONFIG_MEMCG_KMEM
```

```
static inline bool memcg_can_account_kmem(struct mem_cgroup *memcg)
```

```
{
```

```
@@ -3095,6 +3097,34 @@ out:
```

```
    return new_cachep;
```

```
}
```

```
+void kmem_cache_destroy_memcg_children(struct kmem_cache *s)
```

```
+{
```

```
+ struct kmem_cache *c;
```

```
+ int i;
```

```
+
```

```
+ if (!s->memcg_params)
```

```
+ return;
```

```
+ if (!s->memcg_params->is_root_cache)
```

```
+ return;
```

```
+
```

```
+ /*
```

```
+ * If the cache is being destroyed, we trust that there is no one else
```

```
+ * requesting objects from it. Even if there are, the sanity checks in
```

```
+ * kmem_cache_destroy should caught this ill-case.
```

```
+ *
```

```
+ * Still, we don't want anyone else freeing memcg_caches under our
```

```
+ * noses, which can happen if a new memcg comes to life. As usual,
```

```
+ * we'll take the set_limit_mutex to protect ourselves against this.
```

```
+ */
```

```
+ mutex_lock(&set_limit_mutex);
```

```
+ for (i = 0; i < memcg_limited_groups_array_size; i++) {
```

```
+ c = s->memcg_params->memcg_caches[i];
```

```
+ if (c)
```

```
+ kmem_cache_destroy(c);
```

```
+ }
```

```
+ mutex_unlock(&set_limit_mutex);
```

```
+}
```

```
+
```

```
struct create_work {
```

```
    struct mem_cgroup *memcg;
```

```
    struct kmem_cache *cachep;
```

```

@@ -4192,8 +4222,6 @@ void mem_cgroup_print_bad_page(struct page *page)
}
#endif

-static DEFINE_MUTEX(set_limit_mutex);
-
static int mem_cgroup_resize_limit(struct mem_cgroup *memcg,
    unsigned long long val)
{
diff --git a/mm/slab_common.c b/mm/slab_common.c
index fcf59d7..c02faf5 100644
--- a/mm/slab_common.c
+++ b/mm/slab_common.c
@@ -214,6 +214,9 @@ EXPORT_SYMBOL(kmem_cache_create);

void kmem_cache_destroy(struct kmem_cache *s)
{
+ /* Destroy all the children caches if we aren't a memcg cache */
+ kmem_cache_destroy_memcg_children(s);
+
    get_online_cpus();
    mutex_lock(&slab_mutex);
    s->refcount--;
}
--
1.7.11.7

```
