Subject: [PATCH v2 19/29] skip memcg kmem allocations in specified code regions
Posted by Glauber Costa on Fri, 11 May 2012 17:44:21 GMT
View Forum Message <> Reply to Message

This patch creates a mechanism that skip memcg allocations during
certain pieces of our core code. 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.

We need this to prevent races in early cache creation, when we
allocate data using caches that are not necessarily created already.

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 | 25 ++++++++++++++++++++++++
 2 files changed, 26 insertions(+), 0 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 81a173c..0501114 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;
+ atomic_t 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 5a7416b..c4ecf9c 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -479,6 +479,21 @@ 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)
+{
+ if (!current->mm)
+ return;

```
+
+ atomic_inc(&current->memcg_kmem_skip_account);
+}
+
+static void memcg_resume_kmem_account(void)
+{
+ if (!current->mm)
+  return;
+
+ atomic_dec(&current->memcg_kmem_skip_account);
+}
 char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct kmem_cache *cachep)
 {
  char *name;
@@ -540,7 +555,9 @@ 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_resume_kmem_account();

  if (new_cachep == NULL) {
   new_cachep = cachep;
@@ -631,7 +648,9 @@ 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_resume_kmem_account();
  if (cw == NULL) {
   css_put(&memcg->css);
   return;
@@ -666,6 +685,9 @@ struct kmem_cache *__mem_cgroup_get_kmem_cache(struct
kmem_cache *cachep,
  int idx;
  struct task_struct *p;

+ if (!current->mm || atomic_read(&current->memcg_kmem_skip_account))
+  return cachep;
+
  gfp |=  cachep->allocflags;

  if (cachep->memcg_params.memcg)
@@ -700,6 +722,9 @@ bool __mem_cgroup_new_kmem_page(struct page *page, gfp_t gfp)
  if (!current->mm || in_interrupt())
```

```
   return true;

+ if (!current->mm || atomic_read(&current->memcg_kmem_skip_account))
+  return true;
+
  rcu_read_lock();
  p = rcu_dereference(current->mm->owner);
  memcg = mem_cgroup_from_task(p);
--
1.7.7.6
```