Subject: [PATCH 04/10] memcg: skip memcg kmem allocations in specified code regions
Posted by Glauber Costa on Wed, 25 Jul 2012 14:38:15 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 | 27 +++++++++++++++++++++++++++
 2 files changed, 28 insertions(+)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index c350e60..c09cd1b 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1562,6 +1562,7 @@ struct task_struct {
   unsigned long nr_pages; /* uncharged usage */
   unsigned long memsw_nr_pages; /* uncharged mem+swap usage */
  } memcg_batch;
+ unsigned 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 8d012c7..854f6cc 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -541,6 +541,22 @@ static inline bool memcg_kmem_enabled(struct mem_cgroup *memcg)
   memcg->kmem_accounted;
 }

+static void memcg_stop_kmem_account(void)
+{
+ if (!current->mm)

```
+  return;
+
+ current->memcg_kmem_skip_account++;
+}
+
+static void memcg_resume_kmem_account(void)
+{
+ if (!current->mm)
+  return;
+
+ current->memcg_kmem_skip_account--;
+}
+
 static char *memcg_cache_name(struct mem_cgroup *memcg, struct kmem_cache *cachep)
 {
  char *name;
@@ -614,6 +630,10 @@ bool __memcg_kmem_new_page(gfp_t gfp, void *_handle, int order)
  struct task_struct *p;

  *handle = NULL;
+
+ if (current->memcg_kmem_skip_account)
+  return true;
+
  rcu_read_lock();
  p = rcu_dereference(current->mm->owner);
  memcg = mem_cgroup_from_task(p);
@@ -734,7 +754,9 @@ 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;
@@ -824,7 +846,9 @@ 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;
```

```
@@ -859,6 +883,9 @@ struct kmem_cache *__memcg_kmem_get_cache(struct kmem_cache
*cachep,
  int idx;
  struct task_struct *p;

+ if (!current->mm || current->memcg_kmem_skip_account)
+  return cachep;
+
  if (cachep->memcg_params.memcg)
   return cachep;

--
1.7.10.4
```