Subject: [PATCH 07/10] memcg: Stop res_counter underflows.
Posted by Suleiman Souhlal on Mon, 27 Feb 2012 22:58:50 GMT
View Forum Message <> Reply to Message

From: Hugh Dickins <hughd@google.com>

If __mem_cgroup_try_charge() goes the "bypass" route in charging slab
(typically when the task has been OOM-killed), that later results in
res_counter_uncharge_locked() underflows - a stream of warnings from
kernel/res_counter.c:96!

Solve this by accounting kmem_bypass when we shift that charge to root,
and whenever a memcg has any kmem_bypass outstanding, deduct from that
when unaccounting kmem, before deducting from kmem_bytes: so that its
kmem_bytes soon returns to being a fair account.

The amount of memory bypassed is shown in memory.stat as
kernel_bypassed_memory.

Signed-off-by: Hugh Dickins <hughd@google.com>
Signed-off-by: Suleiman Souhlal <suleiman@google.com>
---
 mm/memcontrol.c | 43 ++++++++++++++++++++++++++++++++++++++++---
 1 files changed, 40 insertions(+), 3 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index d1c0cd7..6a475ed 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -302,6 +302,9 @@ struct mem_cgroup {
  /* Slab accounting */
  struct kmem_cache *slabs[MAX_KMEM_CACHE_TYPES];
 #endif
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ atomic64_t kmem_bypassed;
+#endif
  int independent_kmem_limit;
 };

@@ -4037,6 +4040,7 @@ enum {
  MCS_INACTIVE_FILE,
  MCS_ACTIVE_FILE,
  MCS_UNEVICTABLE,
+ MCS_KMEM_BYPASSED,
  NR_MCS_STAT,
 };

@@ -4060,7 +4064,8 @@ struct {

```
  {"active_anon", "total_active_anon"},
  {"inactive_file", "total_inactive_file"},
  {"active_file", "total_active_file"},
- {"unevictable", "total_unevictable"}
+ {"unevictable", "total_unevictable"},
+ {"kernel_bypassed_memory", "total_kernel_bypassed_memory"}
 };


@@ -4100,6 +4105,10 @@ mem_cgroup_get_local_stat(struct mem_cgroup *memcg, struct
mcs_total_stat *s)
  s->stat[MCS_ACTIVE_FILE] += val * PAGE_SIZE;
  val = mem_cgroup_nr_lru_pages(memcg, BIT(LRU_UNEVICTABLE));
  s->stat[MCS_UNEVICTABLE] += val * PAGE_SIZE;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ s->stat[MCS_KMEM_BYPASSED] += atomic64_read(&memcg->kmem_bypassed);
+#endif
 }

 static void
@@ -5616,14 +5625,24 @@ memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, long
long delta)
  ret = 0;

  if (memcg && !memcg->independent_kmem_limit) {
+  /*
+   * __mem_cgroup_try_charge may decide to bypass the charge and
+   * set _memcg to NULL, in which case we need to account to the
+   * root.
+   */
   _memcg = memcg;
   if (__mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
       &_memcg, may_oom) != 0)
    return -ENOMEM;
+
+  if (!_memcg && memcg != root_mem_cgroup) {
+   atomic64_add(delta, &memcg->kmem_bypassed);
+   memcg = NULL;
+  }
  }

- if (_memcg)
-  ret = res_counter_charge(&_memcg->kmem_bytes, delta, &fail_res);
+ if (memcg)
+  ret = res_counter_charge(&memcg->kmem_bytes, delta, &fail_res);

  return ret;
```

```
 }
@@ -5631,6 +5650,22 @@ memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, long
long delta)
 void
 memcg_uncharge_kmem(struct mem_cgroup *memcg, long long delta)
 {
+ long long bypassed;
+
+ if (memcg) {
+  bypassed = atomic64_read(&memcg->kmem_bypassed);
+  if (bypassed > 0) {
+   if (bypassed > delta)
+    bypassed = delta;
+   do {
+    memcg_uncharge_kmem(NULL, bypassed);
+    delta -= bypassed;
+    bypassed = atomic64_sub_return(bypassed,
+        &memcg->kmem_bypassed);
+   } while (bypassed < 0); /* Might have raced */
+  }
+ }
+
  if (memcg)
   res_counter_uncharge(&memcg->kmem_bytes, delta);

@@ -5956,6 +5991,7 @@ memcg_kmem_init(struct mem_cgroup *memcg, struct mem_cgroup
*parent)

  memcg_slab_init(memcg);

+ atomic64_set(&memcg->kmem_bypassed, 0);
  memcg->independent_kmem_limit = 0;
 }

@@ -5967,6 +6003,7 @@ memcg_kmem_move(struct mem_cgroup *memcg)

  memcg_slab_move(memcg);

+ atomic64_set(&memcg->kmem_bypassed, 0);
  spin_lock_irqsave(&memcg->kmem_bytes.lock, flags);
  kmem_bytes = memcg->kmem_bytes.usage;
  res_counter_uncharge_locked(&memcg->kmem_bytes, kmem_bytes);
--
1.7.7.3
```