Subject: [PATCH 2/7] Basic kernel memory functionality for the Memory Controller
Posted by Glauber Costa on Tue, 21 Feb 2012 11:34:34 GMT
View Forum Message <> Reply to Message

This patch lays down the foundation for the kernel memory component
of the Memory Controller.

As of today, I am only laying down the following files:

 * memory.independent_kmem_limit
 * memory.kmem.limit_in_bytes
 * memory.kmem.soft_limit_in_bytes
 * memory.kmem.usage_in_bytes

I am omitting the Documentation files in this version, at least
in the first cycle. But they should not differ much from what
I posted previously. The patch itself is not much different
than the previous versions I posted.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kirill A. Shutemov <kirill@shutemov.name>
CC: Greg Thelen <gthelen@google.com>
CC: Johannes Weiner <jweiner@redhat.com>
CC: Michal Hocko <mhocko@suse.cz>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Turner <pjt@google.com>
CC: Frederic Weisbecker <fweisbec@gmail.com>
---
 mm/memcontrol.c | 98 ++-
 1 files changed, 97 insertions(+), 1 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index b15a693..26fda11 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -235,6 +235,10 @@ struct mem_cgroup {
   */
  struct res_counter memsw;
  /*
+ * the counter to account for kmem usage.
+ */
+ struct res_counter kmem;
+ /*
   * Per cgroup active and inactive list, similar to the
   * per zone LRU lists.
   */
@@ -280,6 +284,11 @@ struct mem_cgroup {
   */

```
  unsigned long  move_charge_at_immigrate;
  /*
+  * Should kernel memory limits be stabilished independently
+  * from user memory ?
+  */
+ int  kmem_independent_accounting;
+ /*
   * percpu counter.
   */
  struct mem_cgroup_stat_cpu *stat;
@@ -356,6 +365,7 @@ enum mem_type {
  _MEM = 0,
  _MEMSWAP,
  _OOM_TYPE,
+ _KMEM,
 };

 #define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
@@ -3844,6 +3854,11 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
   else
    val = res_counter_read_u64(&memcg->memsw, name);
   break;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ case _KMEM:
+  val = res_counter_read_u64(&memcg->kmem, name);
+  break;
+#endif
  default:
   BUG();
   break;
@@ -3876,7 +3891,13 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
    break;
   if (type == _MEM)
    ret = mem_cgroup_resize_limit(memcg, val);
-  else
+  else if (type == _KMEM) {
+   if (!memcg->kmem_independent_accounting) {
+    ret = -EINVAL;
+    break;
+   }
+   ret = res_counter_set_limit(&memcg->kmem, val);
+  } else
    ret = mem_cgroup_resize_memsw_limit(memcg, val);
   break;
  case RES_SOFT_LIMIT:
@@ -3890,6 +3911,16 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
    */
   if (type == _MEM)
```

```
    ret = res_counter_set_soft_limit(&memcg->res, val);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+  else if (type == _KMEM) {
+   if (!memcg->kmem_independent_accounting) {
+    ret = -EINVAL;
+    break;
+   }
+   ret = res_counter_set_soft_limit(&memcg->kmem, val);
+   break;
+  }
+#endif
   else
    ret = -EINVAL;
   break;
@@ -4573,8 +4604,69 @@ static int mem_control_numa_stat_open(struct inode *unused, struct
file *file)
 #endif /* CONFIG_NUMA */

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static u64 kmem_limit_independent_read(struct cgroup *cgroup, struct cftype *cft)
+{
+ return mem_cgroup_from_cont(cgroup)->kmem_independent_accounting;
+}
+
+static int kmem_limit_independent_write(struct cgroup *cgroup, struct cftype *cft,
+     u64 val)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+
+ val = !!val;
+ /*
+  * This follows the same hierarchy restrictions than
+  * mem_cgroup_hierarchy_write().
+  *
+  * TODO: We also shouldn't allow cgroups
+  * with tasks in it to change this value. Otherwise it is impossible
+  * to track the kernel memory that is already in memcg->res.
+  */
+ if (!parent || !parent->use_hierarchy || mem_cgroup_is_root(parent)) {
+  if (list_empty(&cgroup->children))
+   memcg->kmem_independent_accounting = val;
+  else
+   return -EBUSY;
+ } else
+  return -EINVAL;
+
+ return 0;
```

```
+}
+static struct cftype kmem_cgroup_files[] = {
+ {
+  .name = "independent_kmem_limit",
+  .read_u64 = kmem_limit_independent_read,
+  .write_u64 = kmem_limit_independent_write,
+ },
+ {
+  .name = "kmem.usage_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
+  .read_u64 = mem_cgroup_read,
+ },
+ {
+  .name = "kmem.limit_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
+  .read_u64 = mem_cgroup_read,
+  .write_string = mem_cgroup_write,
+ },
+ {
+  .name = "kmem.soft_limit_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_SOFT_LIMIT),
+  .write_string = mem_cgroup_write,
+  .read_u64 = mem_cgroup_read,
+ },
+};
+
+
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
+ int ret;
+ ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
+         ARRAY_SIZE(kmem_cgroup_files));
+ if (ret)
+  return ret;
  /*
   * Part of this would be better living in a separate allocation
   * function, leaving us with just the cgroup tree population work.
@@ -4926,6 +5018,9 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  if (parent && parent->use_hierarchy) {
   res_counter_init(&memcg->res, &parent->res);
   res_counter_init(&memcg->memsw, &parent->memsw);
+  res_counter_init(&memcg->kmem, &parent->kmem);
+  memcg->kmem_independent_accounting =
+     parent->kmem_independent_accounting;
   /*
    * We increment refcnt of the parent to ensure that we can
    * safely access it on res_counter_charge/uncharge.
@@ -4936,6 +5031,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
  } else {
   res_counter_init(&memcg->res, NULL);
   res_counter_init(&memcg->memsw, NULL);
+  res_counter_init(&memcg->kmem, NULL);
  }
  memcg->last_scanned_node = MAX_NUMNODES;
  INIT_LIST_HEAD(&memcg->oom_notify);
--
1.7.7.6
```