
Subject: [PATCH 06/11] memcg: kmem controller infrastructure
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:23 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch introduces infrastructure for tracking kernel memory pages to a given memcg. This will happen whenever the caller includes the flag `__GFP_KMEMCG` flag, and the task belong to a memcg other than the root.

In `memcontrol.h` those functions are wrapped in inline accessors. The idea is to later on, patch those with jump labels, so we don't incur any overhead when no mem cgroups are being used.

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>

include/linux/memcontrol.h | 44 +++++
mm/memcontrol.c | 172 +++++
2 files changed, 216 insertions(+)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 83e7ba9..22479eb 100644

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

@@ -21,6 +21,7 @@
#define _LINUX_MEMCONTROL_H
#include <linux/cgroup.h>
#include <linux/vm_event_item.h>
+#include <linux/hardirq.h>

struct mem_cgroup;
struct page_cgroup;
@@ -409,6 +410,12 @@ struct sock;
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
void sock_update_memcg(struct sock *sk);
void sock_release_memcg(struct sock *sk);
+
+#define mem_cgroup_kmem_on 1
+bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
+void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
+void __mem_cgroup_free_kmem_page(struct page *page, int order);
+#define is_kmem_tracked_alloc(gfp & __GFP_KMEMCG)
#else
static inline void sock_update_memcg(struct sock *sk)


```

+++ b/mm/memcontrol.c
@@ -10,6 +10,10 @@
 * Copyright (C) 2009 Nokia Corporation
 * Author: Kirill A. Shutemov
 *
+ * Kernel Memory Controller
+ * Copyright (C) 2012 Parallels Inc. and Google Inc.
+ * Authors: Glauber Costa and Suleiman Souhlal
+ *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
@@ -422,6 +426,9 @@ static void mem_cgroup_put(struct mem_cgroup *memcg);
#include <net/ip.h>

static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
+static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta);
+static void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta);
+
void sock_update_memcg(struct sock *sk)
{
    if (mem_cgroup_sockets_enabled) {
@@ -476,6 +483,105 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
    }
    EXPORT_SYMBOL(tcp_proto_cgroup);
    #endif /* CONFIG_INET */
+
+static inline bool mem_cgroup_kmem_enabled(struct mem_cgroup *memcg)
+{
+    return !mem_cgroup_disabled() && memcg &&
+        !mem_cgroup_is_root(memcg) && memcg->kmem_accounted;
+}
+
+bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *_handle, int order)
+{
+    struct mem_cgroup *memcg;
+    struct mem_cgroup **handle = (struct mem_cgroup **) _handle;
+    bool ret = true;
+    size_t size;
+    struct task_struct *p;
+
+    *handle = NULL;
+    rcu_read_lock();
+    p = rcu_dereference(current->mm->owner);
+    memcg = mem_cgroup_from_task(p);
+    if (!mem_cgroup_kmem_enabled(memcg))
+        goto out;
+

```

```

+ mem_cgroup_get(memcg);
+
+ size = (1 << order) << PAGE_SHIFT;
+ ret = memcg_charge_kmem(memcg, gfp, size) == 0;
+ if (!ret) {
+   mem_cgroup_put(memcg);
+   goto out;
+ }
+
+ *handle = memcg;
+out:
+ rcu_read_unlock();
+ return ret;
+}
+EXPORT_SYMBOL(__mem_cgroup_new_kmem_page);
+
+void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order)
+{
+   struct page_cgroup *pc;
+   struct mem_cgroup *memcg = handle;
+   size_t size;
+
+   if (!memcg)
+       return;
+
+   WARN_ON(mem_cgroup_is_root(memcg));
+   /* The page allocation must have failed. Revert */
+   if (!page) {
+       size = (1 << order) << PAGE_SHIFT;
+       memcg_uncharge_kmem(memcg, size);
+       mem_cgroup_put(memcg);
+       return;
+   }
+
+   pc = lookup_page_cgroup(page);
+   lock_page_cgroup(pc);
+   pc->mem_cgroup = memcg;
+   SetPageCgroupUsed(pc);
+   unlock_page_cgroup(pc);
+}
+void __mem_cgroup_free_kmem_page(struct page *page, int order)
+{
+   struct mem_cgroup *memcg;
+   size_t size;
+   struct page_cgroup *pc;
+
+   if (mem_cgroup_disabled())
+       return;

```



```

+ ret = 0;
+
+ if (!memcg)
+ return ret;
+
+ _memcg = memcg;
+ ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
+   &_memcg, may_oom);
+
+ if (ret == -EINTR) {
+   nofail = true;
+   /*
+    * __mem_cgroup_try_charge() chose to bypass to root due
+    * to OOM kill or fatal signal.
+    * Since our only options are to either fail the
+    * allocation or charge it to this cgroup, do it as
+    * a temporary condition. But we can't fail. From a kmem/slab
+    * perspective, the cache has already been selected, by
+    * mem_cgroup_get_kmem_cache(), so it is too late to change our
+    * minds
+    */
+   res_counter_charge_nofail(&memcg->res, delta, &fail_res);
+   if (do_swap_account)
+     res_counter_charge_nofail(&memcg->memsw, delta,
+       &fail_res);
+   ret = 0;
+ } else if (ret == -ENOMEM)
+ return ret;
+
+ if (nofail)
+   res_counter_charge_nofail(&memcg->kmem, delta, &fail_res);
+ else
+   ret = res_counter_charge(&memcg->kmem, delta, &fail_res);
+
+ if (ret) {
+   res_counter_uncharge(&memcg->res, delta);
+   if (do_swap_account)
+     res_counter_uncharge(&memcg->memsw, delta);
+ }
+
+ return ret;
+}
+
+void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta)
+{
+ if (!memcg)
+ return;
+
+

```

```
+ res_counter_uncharge(&memcg->kmem, delta);
+ res_counter_uncharge(&memcg->res, delta);
+ if (do_swap_account)
+ res_counter_uncharge(&memcg->memsw, delta);
+}
+ #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
--
1.7.10.2
```
