
Subject: [PATCH 07/11] mm: Allocate kernel pages to the right memcg
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:24 GMT
[View Forum Message](#) <> [Reply to Message](#)

When a process tries to allocate a page with the `__GFP_KMEMCG` flag, the page allocator will call the corresponding memcg functions to validate the allocation. Tasks in the root memcg can always proceed.

To avoid adding markers to the page - and a kmem flag that would necessarily follow, as much as doing `page_cgroup` lookups for no reason, whoever is marking its allocations with `__GFP_KMEMCG` flag is responsible for telling the page allocator that this is such an allocation at `free_pages()` time. This is done by the invocation of `__free_accounted_pages()` and `free_accounted_pages()`.

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/gfp.h | 3 +++
mm/page_alloc.c    | 27 +++++
2 files changed, 30 insertions(+)
```

```
diff --git a/include/linux/gfp.h b/include/linux/gfp.h
index 8f4079f..4e27cd8 100644
--- a/include/linux/gfp.h
+++ b/include/linux/gfp.h
@@ -368,6 +368,9 @@ extern void free_pages(unsigned long addr, unsigned int order);
extern void free_hot_cold_page(struct page *page, int cold);
extern void free_hot_cold_page_list(struct list_head *list, int cold);

+extern void __free_accounted_pages(struct page *page, unsigned int order);
+extern void free_accounted_pages(unsigned long addr, unsigned int order);
+
#define __free_page(page) __free_pages((page), 0)
#define free_page(addr) free_pages((addr), 0)
```

```
diff --git a/mm/page_alloc.c b/mm/page_alloc.c
index 4403009..cb8867e 100644
--- a/mm/page_alloc.c
+++ b/mm/page_alloc.c
@@ -2479,6 +2479,7 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
    struct page *page = NULL;
    int migratetype = allocflags_to_migratetype(gfp_mask);
```

```

    unsigned int cpuset_mems_cookie;
+ void *handle = NULL;

    gfp_mask &= gfp_allowed_mask;

@@ -2490,6 +2491,13 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
    return NULL;

    /*
+ * Will only have any effect when __GFP_KMEMCG is set.
+ * This is verified in the (always inline) callee
+ */
+ if (!mem_cgroup_new_kmem_page(gfp_mask, &handle, order))
+ return NULL;
+
+ /*
    * Check the zones suitable for the gfp_mask contain at least one
    * valid zone. It's possible to have an empty zonelist as a result
    * of GFP_THISNODE and a memoryless node
@@ -2528,6 +2536,8 @@ out:
    if (unlikely(!put_mems_allowed(cpuset_mems_cookie) && !page))
        goto retry_cpuset;

+ mem_cgroup_commit_kmem_page(page, handle, order);
+
    return page;
}
EXPORT_SYMBOL(__alloc_pages_nodemask);
@@ -2580,6 +2590,23 @@ void free_pages(unsigned long addr, unsigned int order)

EXPORT_SYMBOL(free_pages);

+void __free_accounted_pages(struct page *page, unsigned int order)
+{
+ mem_cgroup_free_kmem_page(page, order);
+ __free_pages(page, order);
+}
+EXPORT_SYMBOL(__free_accounted_pages);
+
+void free_accounted_pages(unsigned long addr, unsigned int order)
+{
+ if (addr != 0) {
+ VM_BUG_ON(!virt_addr_valid((void *)addr));
+ mem_cgroup_free_kmem_page(virt_to_page((void *)addr), order);
+ __free_pages(virt_to_page((void *)addr), order);
+ }
+}
+EXPORT_SYMBOL(free_accounted_pages);

```

```
+
static void *make_alloc_exact(unsigned long addr, unsigned order, size_t size)
{
    if (addr) {
--
```

1.7.10.2
