Subject: [RFC][PATCH 2/3] The accounting hooks and core
Posted by Pavel Emelianov on Thu, 13 Sep 2007 09:14:40 GMT
View Forum Message <> Reply to Message

The struct page gets an extra pointer (just like it has with
the RSS controller) and this pointer points to the array of
the kmem_container pointers - one for each object stored on
that page itself.

Thus the i'th object on the page is accounted to the container
pointed by the i'th pointer on that array and when the object
is freed we unaccount its size to this particular container,
not the container current task belongs to.

This is done so, because the context objects are freed is most
often not the same as the one this objects was allocated in
(due to RCU and reference counters).

Kmem cache marked as SLAB_CHARGE will perform the accounting.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---

```
 include/linux/mm_types.h |   12 +++
 include/linux/slab.h     |    1
 mm/slub.c                |  165 +++++++++++++++++++++++++++++++++++++++++++++++-
 3 files changed, 176 insertions(+), 2 deletions(-)


diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
index 48df4b4..67e8ea4 100644
--- a/include/linux/mm_types.h
+++ b/include/linux/mm_types.h
@@ -83,9 +83,19 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+
+ /*
+  * one page cannot be mapped to the userspace and be
+  * allocated for slub at the same time
+  */
+ union {
 #ifdef CONFIG_CONTAINER_MEM_CONT
- unsigned long page_container;
+  unsigned long page_container;
+#endif
```

```
+#ifdef CONFIG_CONTAINER_KMEM
+  struct kmem_container **containers;
 #endif
+ };
 #ifdef CONFIG_PAGE_OWNER
  int order;
  unsigned int gfp_mask;
diff --git a/include/linux/slab.h b/include/linux/slab.h
index 3a5bad3..cd7d50d 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -28,6 +28,7 @@
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* Defer freeing slabs to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
 #define SLAB_TRACE  0x00200000UL /* Trace allocations and frees */
+#define SLAB_CHARGE  0x00400000UL /* Charge allocations */

 /* The following flags affect the page allocator grouping pages by mobility */
 #define SLAB_RECLAIM_ACCOUNT 0x00020000UL  /* Objects are reclaimable */
diff --git a/mm/slub.c b/mm/slub.c
index 16da4d4..113df81 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1018,6 +1018,73 @@ static inline void add_full(struct kmem_
 static inline void kmem_cache_open_debug_check(struct kmem_cache *s) {}
 #define slub_debug 0
 #endif
+
+#ifdef CONFIG_CONTAINER_KMEM
+/*
+ * Fast path stubs
+ */
+
+static int __kmem_charge(struct kmem_cache *s, void *obj, gfp_t flags);
+static inline
+int kmem_charge(struct kmem_cache *s, void *obj, gfp_t flags)
+{
+ return (s->flags & SLAB_CHARGE) ? __kmem_charge(s, obj, flags) : 0;
+}
+
+static void __kmem_uncharge(struct kmem_cache *s, void *obj);
+static inline
+void kmem_uncharge(struct kmem_cache *s, void *obj)
+{
+ if (s->flags & SLAB_CHARGE)
+  __kmem_uncharge(s, obj);
+}
+
```

```
+static int __kmem_prepare(struct kmem_cache *s, struct page *pg, gfp_t flags);
+static inline
+int kmem_prepare(struct kmem_cache *s, struct page *pg, gfp_t flags)
+{
+ return (s->flags & SLAB_CHARGE) ? __kmem_prepare(s, pg, flags) : 0;
+}
+
+static void __kmem_release(struct kmem_cache *s, struct page *pg);
+static inline
+void kmem_release(struct kmem_cache *s, struct page *pg)
+{
+ if (s->flags & SLAB_CHARGE)
+  __kmem_release(s, pg);
+}
+
+static inline int is_kmalloc_cache(struct kmem_cache *s)
+{
+ int km_idx;
+
+ km_idx = s - kmalloc_caches;
+ return km_idx >= 0 && km_idx < ARRAY_SIZE(kmalloc_caches);
+}
+#else
+static inline
+int kmem_charge(struct kmem_cache *s, void *obj, gfp_t flags)
+{
+ return 0;
+}
+
+static inline
+void kmem_uncharge(struct kmem_cache *s, void *obj)
+{
+}
+
+static inline
+int kmem_prepare(struct kmem_cache *s, struct page *pg, gfp_t flags)
+{
+ return 0;
+}
+
+static inline
+void kmem_release(struct kmem_cache *s, struct page *pg)
+{
+}
+#endif
+
 /*
  * Slab allocation and freeing
```

```
  */
@@ -1041,7 +1108,10 @@ static struct page *allocate_slab(struct
   page = alloc_pages_node(node, flags, s->order);

  if (!page)
-  return NULL;
+  goto err_page;
+
+ if (kmem_prepare(s, page, flags) < 0)
+  goto err_prep;

  mod_zone_page_state(page_zone(page),
   (s->flags & SLAB_RECLAIM_ACCOUNT) ?
@@ -1049,6 +1119,11 @@ static struct page *allocate_slab(struct
   pages);

  return page;
+
+err_prep:
+ __free_pages(page, s->order);
+err_page:
+ return NULL;
 }

 static void setup_object(struct kmem_cache *s, struct page *page,
@@ -1141,6 +1216,8 @@ static void rcu_free_slab(struct rcu_hea

 static void free_slab(struct kmem_cache *s, struct page *page)
 {
+ kmem_release(s, page);
+
  if (unlikely(s->flags & SLAB_DESTROY_BY_RCU)) {
   /*
    * RCU free overloads the RCU head over the LRU
@@ -1560,6 +1637,11 @@ static void __always_inline *slab_alloc(
  }
  local_irq_restore(flags);

+ if (object && kmem_charge(s, object, gfpflags) < 0) {
+  kmem_cache_free(s, object);
+  return NULL;
+ }
+
  if (unlikely((gfpflags & __GFP_ZERO) && object))
   memset(object, 0, c->objsize);

@@ -1656,6 +1738,8 @@ static void __always_inline slab_free(st
  unsigned long flags;
```

```
  struct kmem_cache_cpu *c;

+ kmem_uncharge(s, x);
+
  local_irq_save(flags);
  debug_check_no_locks_freed(object, s->objsize);
  c = get_cpu_slab(s, smp_processor_id());
@@ -4041,6 +4125,85 @@ struct kmem_container *task_kmem_contain
  return css_to_kmem(task_subsys_state(tsk, kmem_subsys_id));
 }

+static int __kmem_charge(struct kmem_cache *s, void *obj, gfp_t flags)
+{
+ struct page *pg;
+ struct kmem_container *cnt;
+ struct kmem_container **obj_container;
+
+ pg = virt_to_head_page(obj);
+ obj_container = pg->containers;
+ if (unlikely(obj_container == NULL)) {
+  /*
+   * turned on after some objects were allocated
+   */
+  if (__kmem_prepare(s, pg, flags) < 0)
+   goto err;
+
+  obj_container = pg->containers;
+ }
+
+ rcu_read_lock();
+ cnt = task_kmem_container(current);
+ if (res_counter_charge(&cnt->res, s->size))
+  goto err_locked;
+
+ css_get(&cnt->css);
+ rcu_read_unlock();
+ obj_container[slab_index(obj, s, page_address(pg))] = cnt;
+ return 0;
+
+err_locked:
+ rcu_read_unlock();
+err:
+ return -ENOMEM;
+}
+
+static void __kmem_uncharge(struct kmem_cache *s, void *obj)
+{
+ struct page *pg;
```

```
+ struct kmem_container *cnt;
+ struct kmem_container **obj_container;
+
+ pg = virt_to_head_page(obj);
+ obj_container = pg->containers;
+ if (obj_container == NULL)
+  return;
+
+ obj_container += slab_index(obj, s, page_address(pg));
+ cnt = *obj_container;
+ if (cnt == NULL)
+  return;
+
+ res_counter_uncharge(&cnt->res, s->size);
+ *obj_container = NULL;
+ css_put(&cnt->css);
+}
+
+static int __kmem_prepare(struct kmem_cache *s, struct page *pg, gfp_t flags)
+{
+ struct kmem_container **ptr;
+
+ ptr = kzalloc(s->objects * sizeof(struct kmem_container *), flags);
+ if (ptr == NULL)
+  return -ENOMEM;
+
+ pg->containers = ptr;
+ return 0;
+}
+
+static void __kmem_release(struct kmem_cache *s, struct page *pg)
+{
+ struct kmem_container **ptr;
+
+ ptr = pg->containers;
+ if (ptr == NULL)
+  return;
+
+ kfree(ptr);
+ pg->containers = NULL;
+}
+
 /*
  * containers interface
  */
```