Subject: [PATCH 5/5] Account for the slub objects
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:24:20 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).

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

```
diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
index 48df4b4..1a41901 100644
--- a/include/linux/mm_types.h
+++ b/include/linux/mm_types.h
@@ -83,9 +83,14 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+ 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/slub_def.h b/include/linux/slub_def.h
index 40801e7..fd9a3d4 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -69,6 +69,8 @@ struct kmem_cache {
 #endif
```

```
 };

+int slab_index(void *p, struct kmem_cache *s, void *addr);
+
 /*
  * Kmalloc subsystem.
  */
diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..a2c45bb
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -13,6 +13,9 @@
  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  * GNU General Public License for more details.
+ *
+ * Changelog:
+ *   2007 Pavel Emelyanov : Add slub accounting
  */

 #include <linux/mm.h>
@@ -121,3 +124,128 @@
  .subsys_id = kmem_subsys_id,
  .early_init = 1,
 };
+
+/*
+ * slub accounting
+ */
+
+int slub_newpage_notify(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;
+}
+
+void slub_freepage_notify(struct kmem_cache *s, struct page *pg)
+{
+ struct kmem_container **ptr;
+
+ ptr = pg->containers;
```

```
+ if (ptr == NULL)
+  return;
+
+ kfree(ptr);
+ pg->containers = NULL;
+}
+
+int slub_alloc_notify(struct kmem_cache *s, void *obj, gfp_t gfp)
+{
+ 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 (slub_newpage_notify(s, pg, gfp) < 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;
+}
+
+void slub_free_notify(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);
+}
+
+#ifdef CONFIG_SLUB_NOTIFY
+static int kmem_notify(struct notifier_block *nb, unsigned long cmd, void *arg)
+{
+ int ret;
+ struct slub_notify_struct *ns;
+
+ ns = (struct slub_notify_struct *)arg;
+
+ switch (cmd) {
+ case SLUB_ALLOC:
+  ret = slub_alloc_notify(ns->cachep, ns->objp, ns->gfp);
+  break;
+ case SLUB_FREE:
+  ret = 0;
+  slub_free_notify(ns->cachep, ns->objp);
+  break;
+ case SLUB_NEWPAGE:
+  ret = slub_newpage_notify(ns->cachep, ns->objp, ns->gfp);
+  break;
+ case SLUB_FREEPAGE:
+  ret = 0;
+  slub_freepage_notify(ns->cachep, ns->objp);
+  break;
+ default:
+  return NOTIFY_DONE;
+ }
+
+ return (ret < 0) ? notifier_from_errno(ret) : NOTIFY_OK;
+}
+
+static struct notifier_block kmem_block = {
+ .notifier_call = kmem_notify,
+};
+
```

```
+static int kmem_subsys_register(void)
+{
+ return slub_register_notifier(&kmem_block);
+}
+
+__initcall(kmem_subsys_register);
+#endif
diff --git a/mm/slub.c b/mm/slub.c
index ac4f157..b5af598 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -327,7 +327,7 @@ static inline void set_freepointer(struc
  for (__p = (__free); __p; __p = get_freepointer((__s), __p))

 /* Determine object index from a given position */
-static inline int slab_index(void *p, struct kmem_cache *s, void *addr)
+inline int slab_index(void *p, struct kmem_cache *s, void *addr)
 {
  return (p - addr) / s->size;
 }
@@ -2360,6 +2483,14 @@ EXPORT_SYMBOL(kmem_cache_destroy);
 struct kmem_cache kmalloc_caches[PAGE_SHIFT] __cacheline_aligned;
 EXPORT_SYMBOL(kmalloc_caches);

+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);
+}
+
 #ifdef CONFIG_ZONE_DMA
 static struct kmem_cache *kmalloc_caches_dma[PAGE_SHIFT];
 #endif
@@ -2811,6 +2943,16 @@ static int slab_unmergeable(struct kmem_
  if (s->refcount < 0)
   return 1;

+#ifdef CONFIG_CONTAINER_KMEM
+ /*
+  * many caches that can be accountable are usually merged with
+  * kmalloc caches, which are disabled for accounting for a while
+  */
+
+ if (is_kmalloc_cache(s))
+  return 1;
+#endif
```

```
+
  return 0;
 }

@@ -3775,6 +3917,13 @@ static ssize_t defrag_ratio_store(struct
   const char *buf, size_t length)
 {
  if (buf[0] == '1') {
+#ifdef CONFIG_CONTAINER_KMEM
+  if (is_kmalloc_cache(s))
+   /*
+    * cannot just make these caches accountable
+    */
+   return -EINVAL;
+#endif
   s->flags |= SLAB_NOTIFY;
   return length;
  }
```