Subject: [PATCH 06/10] memcg: Track all the memcg children of a kmem_cache.
Posted by Suleiman Souhlal on Mon, 27 Feb 2012 22:58:49 GMT
View Forum Message <> Reply to Message

This enables us to remove all the children of a kmem_cache being
destroyed, if for example the kernel module it's being used in
gets unloaded. Otherwise, the children will still point to the
destroyed parent.

We also use this to propagate /proc/slabinfo settings to all
the children of a cache, when, for example, changing its
batchsize.

Signed-off-by: Suleiman Souhlal <suleiman@google.com>
---
 include/linux/slab_def.h | 1 +
 mm/slab.c | 45 ++---
 2 files changed, 42 insertions(+), 4 deletions(-)

diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
index 449a0de..185e4a2 100644
--- a/include/linux/slab_def.h
+++ b/include/linux/slab_def.h
@@ -100,6 +100,7 @@ struct kmem_cache {

  atomic_t refcnt;
  struct list_head destroyed_list; /* Used when deleting cpuset cache */
+ struct list_head sibling_list;
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

 /* 6) per-cpu/per-node data, touched during every alloc/free */
diff --git a/mm/slab.c b/mm/slab.c
index 6c6bc49..bf38af6 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -2508,6 +2508,7 @@ __kmem_cache_create(const char *name, size_t size, size_t align,
  cachep->orig_cache = NULL;
  atomic_set(&cachep->refcnt, 1);
  INIT_LIST_HEAD(&cachep->destroyed_list);
+ INIT_LIST_HEAD(&cachep->sibling_list);

  if (!memcg) {
   int id;
@@ -2580,6 +2581,7 @@ kmem_cache_create_memcg(struct kmem_cache *cachep, char
*name)
  new->flags |= SLAB_MEMCG;
  new->orig_cache = cachep;

```
+ list_add(&new->sibling_list, &cachep->sibling_list);
  if ((cachep->limit != new->limit) ||
      (cachep->batchcount != new->batchcount) ||
      (cachep->shared != new->shared))
@@ -2767,6 +2769,26 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
 {
  BUG_ON(!cachep || in_interrupt());

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /* Destroy all the children caches if we aren't a memcg cache */
+ if (cachep->id != -1) {
+  struct kmem_cache *c, *tmp;
+
+  mutex_lock(&cache_chain_mutex);
+  list_for_each_entry_safe(c, tmp, &cachep->sibling_list,
+      sibling_list) {
+   if (c == cachep)
+    continue;
+   mutex_unlock(&cache_chain_mutex);
+   BUG_ON(c->id != -1);
+   mem_cgroup_remove_child_kmem_cache(c, cachep->id);
+   kmem_cache_destroy(c);
+   mutex_lock(&cache_chain_mutex);
+  }
+  mutex_unlock(&cache_chain_mutex);
+ }
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
  /* Find the cache in the chain of caches. */
  get_online_cpus();
  mutex_lock(&cache_chain_mutex);
@@ -2774,6 +2796,9 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
   * the chain is never empty, cache_cache is never destroyed
   */
  list_del(&cachep->next);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ list_del(&cachep->sibling_list);
+#endif
  if (__cache_shrink(cachep)) {
   slab_error(cachep, "Can't free all objects");
   list_add(&cachep->next, &cache_chain);
@@ -4628,11 +4653,23 @@ static ssize_t slabinfo_write(struct file *file, const char __user
*buffer,
    if (limit < 1 || batchcount < 1 ||
      batchcount > limit || shared < 0) {
     res = 0;
-   } else {
-    res = do_tune_cpucache(cachep, limit,
```

```
-       batchcount, shared, GFP_KERNEL |
-       __GFP_NOACCOUNT);
+    break;
    }
+
+   res = do_tune_cpucache(cachep, limit, batchcount,
+       shared, GFP_KERNEL | __GFP_NOACCOUNT);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+   {
+    struct kmem_cache *c;
+
+    list_for_each_entry(c, &cachep->sibling_list,
+        sibling_list)
+     do_tune_cpucache(c, limit, batchcount,
+         shared, GFP_KERNEL |
+         __GFP_NOACCOUNT);
+   }
+#endif
    break;
   }
  }
--
1.7.7.3
```