Subject: [PATCH] Show slab memory usage on OOM and SysRq-M (v2)
Posted by xemul on Tue, 17 Apr 2007 15:29:19 GMT
View Forum Message <> Reply to Message

The out_of_memory() function and SysRq-M handler call
show_mem() to show the current memory usage state.

This is also helpful to see which slabs are the largest
in the system.

Thanks Pekka for good idea of how to make it better.

The nr_pages is stored on kmem_list3 because:

1. as Eric pointed out, we do not want to defeat
   NUMA optimizations;
2. we do not need for additional LOCK-ed operation on
   altering this field - l3->list_lock is already taken
   where needed.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>
Cc: Pekka Enberg <penberg@cs.helsinki.fi>
Cc: Eric Dumazet <dada1@cosmosbay.com>

```
diff --git a/drivers/char/sysrq.c b/drivers/char/sysrq.c
index 39cc318..7c27647 100644
--- a/drivers/char/sysrq.c
+++ b/drivers/char/sysrq.c
@@ -234,6 +234,7 @@ static struct sysrq_key_op sysrq_showsta
 static void sysrq_handle_showmem(int key, struct tty_struct *tty)
 {
  show_mem();
+ show_slabs();
 }
 static struct sysrq_key_op sysrq_showmem_op = {
  .handler = sysrq_handle_showmem,
diff --git a/include/linux/slab.h b/include/linux/slab.h
index 67425c2..1e2919d 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -170,6 +170,12 @@ static inline void *kzalloc(size_t size,
 }
 #endif

+#ifdef CONFIG_SLAB
```

```
+extern void show_slabs(void);
+#else
+#define show_slabs(void) do { } while (0)
+#endif
+
 #ifndef CONFIG_NUMA
 static inline void *kmalloc_node(size_t size, gfp_t flags, int node)
 {
diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index 4bdc7c0..aefdd06 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
@@ -409,6 +409,7 @@ void out_of_memory(struct zonelist *zone
    current->comm, gfp_mask, order, current->oomkilladj);
   dump_stack();
   show_mem();
+  show_slabs();
  }

  cpuset_lock();
diff --git a/mm/slab.c b/mm/slab.c
index 21b3c61..5e6a0dc 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -299,6 +299,7 @@ struct kmem_list3 {
  struct array_cache **alien; /* on other nodes */
  unsigned long next_reap; /* updated without locking */
  int free_touched;  /* updated without locking */
+ unsigned long nr_pages;  /* pages allocated for this l3 */
 };

 /*
@@ -357,6 +358,7 @@ static void kmem_list3_init(struct kmem_
  spin_lock_init(&parent->list_lock);
  parent->free_objects = 0;
  parent->free_touched = 0;
+ parent->nr_pages = 0;
 }

 #define MAKE_LIST(cachep, listp, slab, nodeid)    \
@@ -747,8 +749,18 @@ static inline void init_lock_keys(void)
 /*
  * 1. Guard access to the cache-chain.
  * 2. Protect sanity of cpu_online_map against cpu hotplug events
+ *
+ * cache_chain_lock is needed to protect the cache chain list only.
+ * This is needed by show_slabs() only to be sure kmem_caches won't disappear
+ * from under it. The additional spinlock is used because:
```

```
+ *  1. mutex_trylock() may fail if we race with cache_reap;
+ *  2. show_slabs() may be called with this mutex already locked
+ *     (do_tune_cpucache -> kmalloc -> out_of_memory) and thus lock will
+ *     deadlock and trylock will fail for sure.
+ * In both cases we are risking of loosing the info that might be usefull
  */
 static DEFINE_MUTEX(cache_chain_mutex);
+static DEFINE_SPINLOCK(cache_chain_lock);
 static struct list_head cache_chain;

 /*
@@ -2377,7 +2389,9 @@ kmem_cache_create (const char *name, siz
  }

  /* cache setup completed, link it into the list */
+ spin_lock_irq(&cache_chain_lock);
  list_add(&cachep->next, &cache_chain);
+ spin_unlock_irq(&cache_chain_lock);
 oops:
  if (!cachep && (flags & SLAB_PANIC))
   panic("kmem_cache_create(): failed to create slab `%s'\n",
@@ -2492,6 +2506,7 @@ static int drain_freelist(struct kmem_ca
    * to the cache.
    */
   l3->free_objects -= cache->num;
+  l3->nr_pages--;
   spin_unlock_irq(&l3->list_lock);
   slab_destroy(cache, slabp);
   nr_freed++;
@@ -2566,10 +2581,14 @@ void kmem_cache_destroy(struct kmem_cach
  /*
   * the chain is never empty, cache_cache is never destroyed
   */
+ spin_lock_irq(&cache_chain_lock);
  list_del(&cachep->next);
+ spin_unlock_irq(&cache_chain_lock);
  if (__cache_shrink(cachep)) {
   slab_error(cachep, "Can't free all objects");
+  spin_lock_irq(&cache_chain_lock);
   list_add(&cachep->next, &cache_chain);
+  spin_unlock_irq(&cache_chain_lock);
   mutex_unlock(&cache_chain_mutex);
   return;
  }
@@ -2825,6 +2844,7 @@ static int cache_grow(struct kmem_cache
  list_add_tail(&slabp->list, &(l3->slabs_free));
  STATS_INC_GROWN(cachep);
  l3->free_objects += cachep->num;
```

```
+ l3->nr_pages++;
  spin_unlock(&l3->list_lock);
  return 1;
 opps1:
@@ -3520,6 +3540,7 @@ static void free_block(struct kmem_cache
       * a different cache, refer to comments before
       * alloc_slabmgmt.
       */
+    l3->nr_pages--;
      slab_destroy(cachep, slabp);
     } else {
      list_add(&slabp->list, &l3->slabs_free);
@@ -4543,6 +4564,61 @@ const struct seq_operations slabstats_op
 #endif
 #endif

+#define SHOW_TOP_SLABS 10
+
+static unsigned long get_cache_size(struct kmem_cache *cachep)
+{
+ unsigned long pages;
+ int node;
+ struct kmem_list3 *l3;
+
+ pages = 0;
+ for_each_online_node (node) {
+  l3 = cachep->nodelists[node];
+  if (l3 != NULL)
+   pages += l3->nr_pages;
+ }
+
+ return pages << cachep->gfporder;
+}
+
+void show_slabs(void)
+{
+ int i, j;
+ unsigned long size;
+ struct kmem_cache *ptr;
+ unsigned long sizes[SHOW_TOP_SLABS];
+ struct kmem_cache *top[SHOW_TOP_SLABS];
+ unsigned long flags;
+
+ printk("Top %d caches:\n", SHOW_TOP_SLABS);
+ memset(top, 0, sizeof(top));
+ memset(sizes, 0, sizeof(sizes));
+
+ spin_lock_irqsave(&cache_chain_lock, flags);
```

```
+ list_for_each_entry (ptr, &cache_chain, next) {
+  size = get_cache_size(ptr);
+
+  /* find and replace the smallest size seen so far */
+  j = 0;
+  for (i = 1; i < SHOW_TOP_SLABS; i++)
+   if (sizes[i] < sizes[j])
+    j = i;
+
+  if (size > sizes[j]) {
+   sizes[j] = size;
+   top[j] = ptr;
+  }
+ }
+
+ for (i = 0; i < SHOW_TOP_SLABS; i++)
+  if (top[i])
+   printk("%-21s: pages %10lu buffer_size %10u\n",
+     top[i]->name, sizes[i],
+     top[i]->buffer_size);
+ spin_unlock_irqrestore(&cache_chain_lock, flags);
+}
+
 /**
  * ksize - get the actual amount of memory allocated for a given object
  * @objp: Pointer to the object
```