
Subject: [PATCH] Show slab memory usage on OOM and SysRq-M

Posted by [xemul](#) on Tue, 17 Apr 2007 12:50:55 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.

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

Signed-off-by: Kirill Korotaev <dev@openvz.org>

```
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/mm.h b/include/linux/mm.h
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 *kmallocc_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..9a5829a 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -749,6 +749,7 @@ static inline void init_lock_keys(void)
 * 2. Protect sanity of cpu_online_map against cpu hotplug events
 */
static DEFINE_MUTEX(cache_chain_mutex);
+static DEFINE_SPINLOCK(cache_chain_lock);
static struct list_head cache_chain;

/*
@@ -2377,7 +2378,9 @@ 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",
@@ -2566,10 +2569,14 @@ 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;
}
@@ -4543,6 +4550,73 @@ 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 slabs;
+ struct kmem_list3 *l3;
+ struct list_head *lh;
+ int node;
+
+ slabs = 0;
+
+ for_each_online_node (node) {
+ l3 = cachep->nodelists[node];
+ if (l3 == NULL)
+ continue;
+
+ spin_lock(&l3->list_lock);
+ list_for_each (lh, &l3->slabs_full)
+ slabs++;
+ list_for_each (lh, &l3->slabs_partial)
+ slabs++;
+ list_for_each (lh, &l3->slabs_free)
+ slabs++;
+ spin_unlock(&l3->list_lock);
+ }
+
+ return slabs * ((PAGE_SIZE << cachep->gfporder) +
+ (OFF_SLAB(cachep) ? cachep->slabp_cache->buffer_size : 0));
+}
+
+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: size %10lu objsize %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
```
