Subject: Re: [PATCH] Show slab memory usage on OOM and SysRq-M
Posted by xemul on Tue, 17 Apr 2007 13:46:25 GMT
View Forum Message <> Reply to Message

Pekka Enberg wrote:
> Hi,
>
> On 4/17/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> 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.
>
> Makes sense.

Thanks! :)

> On 4/17/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> 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);
>
> So, now we have two locks protecting cache_chain? Please explain why
> you can't use the mutex.

Because OOM can actually happen with this mutex locked. For example
kmem_cache_create() locks it and calls kmalloc(), or write to
/proc/slabinfo also locks it and calls do_tune_cpu_caches(). This is
very rare case and the deadlock is VERY unlikely to happen, but it
will be very disappointing if it happens.

Moreover, I put the call to show_slabs() into sysrq handler, so it may
be called from atomic context.

Making mutex_trylock() is possible, but we risk of loosing this info
in case OOM happens while the mutex is locked for cache shrinking (see
cache_reap() for example)...

So we have a choice - either we have an additional lock on a slow and
rare paths and show this info for sure, or we do not have a lock, but
have a risk of loosing this info.

```
>> +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));
>> +}
>
> Considering you're doing this at out_of_memory() time, wouldn't it
> make more sense to add a ->nr_pages to struct kmem_cache and do the
> tracking in kmem_getpages/kmem_freepages?
```

Sounds good.

```
> I would also drop the OFF_SLAB bits because it really doesn't matter
> that much for your purposes. Besides, you're already per-node and
> per-CPU caches here which attribute to much more memory on NUMA setups
> for example.
```

This gives us a more precise information :) The precision is less than 1%
so if nobody likes/needs it, this may be dropped.

Pavel.