

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 14:33:10 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/26/2012 06:03 PM, Michal Hocko wrote:

> On Tue 18-09-12 18:04:01, Glauber Costa wrote:

>> This patch adds the basic infrastructure for the accounting of the slab  
>> caches. To control that, the following files are created:

>>

>> \* memory.kmem.usage\_in\_bytes

>> \* memory.kmem.limit\_in\_bytes

>> \* memory.kmem.failcnt

>> \* memory.kmem.max\_usage\_in\_bytes

>>

>> They have the same meaning of their user memory counterparts. They  
>> reflect the state of the "kmem" res\_counter.

>

>> The code is not enabled until a limit is set.

>

> "Per cgroup slab memory accounting is not enabled until a limit is set  
> for the group. Once the limit is set the accounting cannot be disabled  
> such a group."

>

> Better?

>

>> This can be tested by the flag "kmem\_accounted".

>

> Sounds as if it could be done from userspace (because you were talking  
> about an user interface) which it cannot and we do not see it in this  
> patch because it is not used anywhere. So please be more specific.

>

>> This means that after the patch is applied, no behavioral changes  
>> exists for whoever is still using memcg to control their memory usage.

>>

>> We always account to both user and kernel resource\_counters.

>

> This is in contradiction with your claim that there is no behavioral  
> change for memcg users. Please clarify when we use u and when u+k  
> accounting.

> "

> There is no behavioral change if the kmem accounting is turned off for  
> memcg users but when there is a kmem.limit\_in\_bytes is set then the  
> memory.usage\_in\_bytes will include both user and kmem memory.

> "

>

>> This

>> effectively means that an independent kernel limit is in place when the  
>> limit is set to a lower value than the user memory. A equal or higher

```

>> value means that the user limit will always hit first, meaning that kmem
>> is effectively unlimited.
>>
>> People who want to track kernel memory but not limit it, can set this
>> limit to a very high number (like RESOURCE_MAX - 1page - that no one
>> will ever hit, or equal to the user memory)
>>
>> Signed-off-by: Glauber Costa <glommer@parallels.com>
>> CC: Michal Hocko <mhocko@suse.cz>
>> CC: Johannes Weiner <hannes@cmpxchg.org>
>> Aacked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>> ---
>> mm/memcontrol.c | 64
+++++
>> 1 file changed, 63 insertions(+), 1 deletion(-)
>>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index d6ad138..f3fd354 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -265,6 +265,10 @@ struct mem_cgroup {
>> };
>>
>> /*
>> + * the counter to account for kernel memory usage.
>> + */
>> + struct res_counter kmem;
>> + /*
>> + * Per cgroup active and inactive list, similar to the
>> + * per zone LRU lists.
>> + */
>> @@ -279,6 +283,7 @@ struct mem_cgroup {
>> + * Should the accounting and control be hierarchical, per subtree?
>> + */
>> + bool use_hierarchy;
>> + bool kmem_accounted;
>>
>> + bool oom_lock;
>> + atomic_t under_oom;
>> @@ -389,6 +394,7 @@ enum res_type {
>> + _MEM,
>> + _MEMSWAP,
>> + _OOM_TYPE,
>> + _KMEM,
>> };
>>
>> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
>> @@ -1439,6 +1445,10 @@ done:

```

```

>> res_counter_read_u64(&memcg->memsw, RES_USAGE) >> 10,
>> res_counter_read_u64(&memcg->memsw, RES_LIMIT) >> 10,
>> res_counter_read_u64(&memcg->memsw, RES_FAILCNT));
>> + printk(KERN_INFO "kmem: usage %lluB, limit %lluB, failcnt %llu\n",
>> + res_counter_read_u64(&memcg->kmem, RES_USAGE) >> 10,
>> + res_counter_read_u64(&memcg->kmem, RES_LIMIT) >> 10,
>> + res_counter_read_u64(&memcg->kmem, RES_FAILCNT));
>> }
>>
>> /*
>> @@ -3946,6 +3956,9 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
>> else
>> val = res_counter_read_u64(&memcg->memsw, name);
>> break;
>> + case _KMEM:
>> + val = res_counter_read_u64(&memcg->kmem, name);
>> + break;
>> default:
>> BUG();
>> }
>> @@ -3984,8 +3997,18 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>> break;
>> if (type == _MEM)
>> ret = mem_cgroup_resize_limit(memcg, val);
>> - else
>> + else if (type == _MEMSWAP)
>> ret = mem_cgroup_resize_memsw_limit(memcg, val);
>> + else if (type == _KMEM) {
>> + ret = res_counter_set_limit(&memcg->kmem, val);
>> + if (ret)
>> + break;
>> +
>> + /* For simplicity, we won't allow this to be disabled */
>> + if (!memcg->kmem_accounted && val != RESOURCE_MAX)
>> + memcg->kmem_accounted = true;
>> + } else
>> + return -EINVAL;
>> break;
>> case RES_SOFT_LIMIT:
>> ret = res_counter_memparse_write_strategy(buffer, &val);
>> @@ -4051,12 +4074,16 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
>> case RES_MAX_USAGE:
>> if (type == _MEM)
>> res_counter_reset_max(&memcg->res);
>> + else if (type == _KMEM)
>> + res_counter_reset_max(&memcg->kmem);

```

```

>> else
>> res_counter_reset_max(&memcg->memsw);
>> break;
>> case RES_FAILCNT:
>> if (type == _MEM)
>> res_counter_reset_failcnt(&memcg->res);
>> + else if (type == _KMEM)
>> + res_counter_reset_failcnt(&memcg->kmem);
>> else
>> res_counter_reset_failcnt(&memcg->memsw);
>> break;
>> @@ -4618,6 +4645,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
>> }
>>
>> #ifdef CONFIG_MEMCG_KMEM
>
> Some things are guarded CONFIG_MEMCG_KMEM but some are not (e.g. struct
> mem_cgroup.kmem). I do understand you want to keep ifdefs on the leash
> but we should clean this up one day.
>
>> +static struct cftype kmem_cgroup_files[] = {
>> + {
>> + .name = "kmem.limit_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
>> + .write_string = mem_cgroup_write,
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "kmem.usage_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "kmem.failcnt",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
>> + .trigger = mem_cgroup_reset,
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "kmem.max_usage_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
>> + .trigger = mem_cgroup_reset,
>> + .read = mem_cgroup_read,
>> + },
>> + {},
>> +};
>> +
>> static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)

```

```

>> {
>> return mem_cgroup_sockets_init(memcg, ss);
>> @@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
>> int cpu;
>> enable_swap_cgroup();
>> parent = NULL;
>> +
>> + #ifdef CONFIG_MEMCG_KMEM
>> + WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
>> + kmem_cgroup_files));
>> + #endif
>> +
>> if (mem_cgroup_soft_limit_tree_init())
>> goto free_out;
>> root_mem_cgroup = memcg;
>> @@ -4979,6 +5039,7 @@ mem_cgroup_create(struct cgroup *cont)
>> if (parent && parent->use_hierarchy) {
>> res_counter_init(&memcg->res, &parent->res);
>> res_counter_init(&memcg->memsw, &parent->memsw);
>> + res_counter_init(&memcg->kmem, &parent->kmem);
>
> Haven't we already discussed that a new memcg should inherit kmem_accounted
> from its parent for use_hierarchy?
> Say we have
> root
> |
> A (kmem_accounted = 1, use_hierarchy = 1)
> \
> B (kmem_accounted = 0)
> \
> C (kmem_accounted = 1)
>
> B find's itself in an awkward situation because it doesn't want to
> account u+k but it ends up doing so because C.
>

```

Ok, I haven't updated it here. But that should be taken care of in the lifecycle patch.