Subject: Re: [PATCH 09/23] kmem slab accounting basic infrastructure
Posted by Suleiman Souhlal on Mon, 30 Apr 2012 19:33:53 GMT
View Forum Message <> Reply to Message

On Fri, Apr 20, 2012 at 2:57 PM, Glauber Costa <glommer@parallels.com> 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. This can be tested by the flag
> "kmem_accounted". 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 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: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> ---
> mm/memcontrol.c | 80
++-
> 1 files changed, 79 insertions(+), 1 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 2810228..36f1e6b 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -252,6 +252,10 @@ 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.
>         */
> @@ -266,6 +270,7 @@ 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;
> @@ -378,6 +383,7 @@ enum res_type {
>        _MEM,
>        _MEMSWAP,
>        _OOM_TYPE,
> +       _KMEM,
>  };
>
>  #define MEMFILE_PRIVATE(x, val)        (((x) << 16) | (val))
> @@ -1470,6 +1476,10 @@ 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 %llukB, limit %llukB, 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));
>  }
>
>  /*
> @@ -3914,6 +3924,11 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
>                else
>                        val = res_counter_read_u64(&memcg->memsw, name);
>                break;
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +       case _KMEM:
> +               val = res_counter_read_u64(&memcg->kmem, name);
> +               break;
> +#endif
>        default:
>                BUG();
>        }
> @@ -3951,8 +3966,26 @@ 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);
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +             else if (type == _KMEM) {
> +                     ret = res_counter_set_limit(&memcg->kmem, val);
> +                     if (ret)
> +                             break;
> +                     /*
> +                      * Once enabled, can't be disabled. We could in theory
> +                      * disable it if we haven't yet created any caches, or
> +                      * if we can shrink them all to death.
> +                      *
> +                      * But it is not worth the trouble
> +                      */
> +                     if (!memcg->kmem_accounted && val != RESOURCE_MAX)
> +                             memcg->kmem_accounted = true;
> +             }
> +#endif
> +             else
> +                     return -EINVAL;
>               break;
>       case RES_SOFT_LIMIT:
>               ret = res_counter_memparse_write_strategy(buffer, &val);
```

Why is RESOURCE_MAX special?

-- Suleiman