Subject: Re: [PATCH v2 02/13] memcg: Kernel memory accounting infrastructure.
Posted by Glauber Costa on Sun, 11 Mar 2012 08:12:04 GMT
View Forum Message <> Reply to Message

On 03/10/2012 12:39 AM, Suleiman Souhlal wrote:
> Enabled with CONFIG_CGROUP_MEM_RES_CTLR_KMEM.
>
> Adds the following files:
> - memory.kmem.independent_kmem_limit
> - memory.kmem.usage_in_bytes
> - memory.kmem.limit_in_bytes
>
> Signed-off-by: Suleiman Souhlal<suleiman@google.com>
> ---
> mm/memcontrol.c | 136
++-
> 1 files changed, 135 insertions(+), 1 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 37ad2cb..e6fd558 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -220,6 +220,10 @@ enum memcg_flags {
> */
> MEMCG_MEMSW_IS_MINIMUM, /* Set when res.limit == memsw.limit */
> MEMCG_OOM_KILL_DISABLE, /* OOM-Killer disable */
> + MEMCG_INDEPENDENT_KMEM_LIMIT, /*
> + * kernel memory is not counted in
> + * memory.usage_in_bytes
> + */
> };
>
> /*
> @@ -244,6 +248,10 @@ struct mem_cgroup {
> */
> struct res_counter memsw;
> /*
> + * 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.
> */
> @@ -355,6 +363,7 @@ enum charge_type {
> #define _MEM (0)
> #define _MEMSWAP (1)
> #define _OOM_TYPE (2)

```
> +#define _KMEM   (3)
>   #define MEMFILE_PRIVATE(x, val) (((x)<<  16) | (val))
>   #define MEMFILE_TYPE(val) (((val)>>  16)&  0xffff)
>   #define MEMFILE_ATTR(val) ((val)&  0xffff)
> @@ -371,6 +380,8 @@ enum charge_type {
>
>   static void mem_cgroup_get(struct mem_cgroup *memcg);
>   static void mem_cgroup_put(struct mem_cgroup *memcg);
> +static void memcg_kmem_init(struct mem_cgroup *memcg,
> +    struct mem_cgroup *parent);
>
>   static inline bool
>   mem_cgroup_test_flag(const struct mem_cgroup *memcg, enum memcg_flags flag)
> @@ -1435,6 +1446,10 @@ 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));
>   }
>
>   /*
> @@ -3868,6 +3883,9 @@ static u64 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();
>     break;
> @@ -3900,8 +3918,15 @@ 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) {
> +   if (!mem_cgroup_test_flag(memcg,
> +       MEMCG_INDEPENDENT_KMEM_LIMIT))
> +    return -EINVAL;
> +   ret = res_counter_set_limit(&memcg->kmem, val);
> +  } else
> +   return -EINVAL;
```

>   break;
>   case RES_SOFT_LIMIT:
>   ret = res_counter_memparse_write_strategy(buffer,&val);
> @@ -4606,8 +4631,56 @@ static int mem_control_numa_stat_open(struct inode *unused,
struct file *file)
>   #endif /* CONFIG_NUMA */
>
>   #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +static u64
> +mem_cgroup_independent_kmem_limit_read(struct cgroup *cgrp, struct cftype *cft)
> +{
> + return mem_cgroup_test_flag(mem_cgroup_from_cont(cgrp),
> +   MEMCG_INDEPENDENT_KMEM_LIMIT);
> +}
> +
> +static int mem_cgroup_independent_kmem_limit_write(struct cgroup *cgrp,
> +   struct cftype *cft, u64 val)
> +{
> + struct mem_cgroup *memcg;
> +
> + memcg = mem_cgroup_from_cont(cgrp);
> + if (val)
> + mem_cgroup_set_flag(memcg, MEMCG_INDEPENDENT_KMEM_LIMIT);
> + else {
> + mem_cgroup_clear_flag(memcg, MEMCG_INDEPENDENT_KMEM_LIMIT);
> + res_counter_set_limit(&memcg->kmem, RESOURCE_MAX);
> + }
> +
> + return 0;
> +}

We need this test to be a bit more strict.
This is what I have in my current version:

```
struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
struct mem_cgroup *parent = parent_mem_cgroup(memcg);

val = !!val;

if (!parent || !parent->use_hierarchy || mem_cgroup_is_root(parent)) {
    if (list_empty(&cgroup->children))
        memcg->kmem_independent_accounting = val;
    else
        return -EBUSY;
} else
    return -EINVAL;

return 0;
```

Over yours, it basically:
  * Makes sure this has no effect on root cgroup
  * disallow changing that when we already have children.

Also, I have a TODO item on that: We need to make sure that no memory
was already accounted. It was a bit tricky for me, because I was not
charging kmem when !independent. But since you are always charging kmem,
and you are so far facing no opposition on that particular point,
maybe you can also test if RES_USAGE == 0 for memcg->kmem.

> +static struct cftype kmem_cgroup_files[] = {
> + {
> + .name = "kmem.independent_kmem_limit",
> + .write_u64 = mem_cgroup_independent_kmem_limit_write,
> + .read_u64 = mem_cgroup_independent_kmem_limit_read,
> + },
> + {
> + .name = "kmem.limit_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
> + .write_string = mem_cgroup_write,
> + .read_u64 = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.usage_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
> + .read_u64 = mem_cgroup_read,
> + },
> +};
> +
> static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
> {
> + int ret;
> +
> + ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
> +   ARRAY_SIZE(kmem_cgroup_files));
> + if (ret)
> + return ret;
>   /*
>   * Part of this would be better living in a separate allocation
>   * function, leaving us with just the cgroup tree population work.
> @@ -4621,6 +4694,10 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)
> static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
>   struct cgroup *cont)
> {
> + struct mem_cgroup *memcg;
> +

```
> + memcg = mem_cgroup_from_cont(cont);
> + BUG_ON(res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0);
>    mem_cgroup_sockets_destroy(cont, ss);
>   }
>   #else
> @@ -4980,6 +5057,8 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>    }
>    memcg->last_scanned_node = MAX_NUMNODES;
>    INIT_LIST_HEAD(&memcg->oom_notify);
> + memcg_kmem_init(memcg, parent&&  mem_cgroup_test_flag(parent,
> +     MEMCG_USE_HIERARCHY) ? parent : NULL);
>
>    if (parent)
>     memcg->swappiness = mem_cgroup_swappiness(parent);
> @@ -5561,3 +5640,58 @@ static int __init enable_swap_account(char *s)
>   __setup("swapaccount=", enable_swap_account);
>
>   #endif
> +
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +int
> +memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, long long delta)
> +{
> + struct res_counter *fail_res;
> + struct mem_cgroup *_memcg;
> + int may_oom, ret;
> +
> + may_oom = (gfp&  __GFP_WAIT)&&  (gfp&  __GFP_FS)&&
> +     !(gfp&  __GFP_NORETRY);
> +
> + ret = 0;
> +
> + _memcg = memcg;
> + if (memcg&&  !mem_cgroup_test_flag(memcg,
> +     MEMCG_INDEPENDENT_KMEM_LIMIT)) {
> +  ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
> +  &_memcg, may_oom);
> +  if (ret == -ENOMEM)
> +   return ret;
> + }
> +
> + if (memcg&&  _memcg == memcg)
> +  ret = res_counter_charge(&memcg->kmem, delta,&fail_res);
> +
I don't really follow this if (memcg

If you are planning to call this unconditionally from the slab code, I
think we should at least exit early if !memcg.
```

Like this:

int may_oom, ret

if (mem_cgroup_disabled() || !memcg)
    return 0;

(And you may need the mem_cgroup_disabled() in your code anyway)

> + return ret;
> +}
> +
> +void
> +memcg_uncharge_kmem(struct mem_cgroup *memcg, long long delta)
> +{
> + if (memcg)
> + res_counter_uncharge(&memcg->kmem, delta);
> +
> + if (memcg&& !mem_cgroup_test_flag(memcg, MEMCG_INDEPENDENT_KMEM_LIMIT))
> + res_counter_uncharge(&memcg->res, delta);
> +}
mem_cgroup_disabled() here too ?

(Actually, I just grep'd and noticed you wrap some code around it in
patch 7. It'd make more sense not to call this function when memcg ==
NULL then ?

mem_cgroup_disabled() goes on that function, before the rcu lock.

> +
> +static void
> +memcg_kmem_init(struct mem_cgroup *memcg, struct mem_cgroup *parent)
> +{
> + struct res_counter *parent_res;
> +
> + parent_res = NULL;
> + if (parent&& parent != root_mem_cgroup)
> + parent_res =&parent->kmem;
> + res_counter_init(&memcg->kmem, parent_res);
> +}
> +#else /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> +static void
> +memcg_kmem_init(struct mem_cgroup *memcg, struct mem_cgroup *parent)
> +{
> +}
> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */