Subject: Re: [PATCH 04/11] kmem slab accounting basic infrastructure
Posted by David Rientjes on Tue, 26 Jun 2012 04:22:38 GMT
View Forum Message <> Reply to Message

On Mon, 25 Jun 2012, Glauber Costa wrote:

> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 9352d40..6f34b77 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;
> @@ -391,6 +396,7 @@ enum res_type {
> _MEM,
> _MEMSWAP,
> _OOM_TYPE,
> + _KMEM,
> };
>
> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
> @@ -1438,6 +1444,10 @@ 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));
> }
>
> /*

> @@ -3879,6 +3889,11 @@ 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

This shouldn't need an #ifdef, ->kmem is available on all CONFIG_CGROUP_MEM_RES_CTLR kernels. Same with several of the other instances in this patch.

Can't these instances be addressed by not adding kmem_cgroup_files without CONFIG_CGROUP_MEM_RES_CTLR_KMEM?

> default:
> BUG();
> }
> @@ -3916,8 +3931,26 @@ 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);
> @@ -3982,12 +4015,20 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
>   case RES_MAX_USAGE:
>   if (type == _MEM)
>   res_counter_reset_max(&memcg->res);
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + else if (type == _KMEM)
> +   res_counter_reset_max(&memcg->kmem);
> +#endif
>   else
>   res_counter_reset_max(&memcg->memsw);
>   break;
>   case RES_FAILCNT:
>   if (type == _MEM)
>   res_counter_reset_failcnt(&memcg->res);
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + else if (type == _KMEM)
> +   res_counter_reset_failcnt(&memcg->kmem);
> +#endif
>   else
>   res_counter_reset_failcnt(&memcg->memsw);
>   break;
> @@ -4549,6 +4590,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
> }
>
> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +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);
> @@ -4892,6 +4960,12 @@ mem_cgroup_create(struct cgroup *cont)
>    int cpu;
>    enable_swap_cgroup();
>    parent = NULL;
> +
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_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;
> @@ -4910,6 +4984,7 @@ 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);
>    /*
>     * We increment refcnt of the parent to ensure that we can
>     * safely access it on res_counter_charge/uncharge.
> @@ -4920,6 +4995,7 @@ mem_cgroup_create(struct cgroup *cont)
>   } else {
>    res_counter_init(&memcg->res, NULL);
>    res_counter_init(&memcg->memsw, NULL);
> +  res_counter_init(&memcg->kmem, NULL);
>   }
>   memcg->last_scanned_node = MAX_NUMNODES;
>   INIT_LIST_HEAD(&memcg->oom_notify);
```