Subject: Re: [PATCH 01/10] memcg: Kernel memory accounting infrastructure.
Posted by Glauber Costa on Tue, 28 Feb 2012 13:11:16 GMT
View Forum Message <> Reply to Message

On 02/27/2012 07:58 PM, 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 | 121
++-
> 1 files changed, 120 insertions(+), 1 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 228d646..11e31d6 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -235,6 +235,10 @@ struct mem_cgroup {
> */
> struct res_counter memsw;
> /*
> + * the counter to account for kernel memory usage.
> + */
> + struct res_counter kmem_bytes;
> + /*
Not terribly important, but I find this name inconsistent. I like
just kmem better.

> * Per cgroup active and inactive list, similar to the
> * per zone LRU lists.
> */
> @@ -293,6 +297,7 @@ struct mem_cgroup {
> #ifdef CONFIG_INET
> struct tcp_memcontrol tcp_mem;
> #endif
> + int independent_kmem_limit;
> };
bool ?

But that said, we are now approaching some 4 or 5 selectables in the
memcg structure. How about we turn them into flags?

> /* Stuffs for move charges at task migration. */

```
> @@ -354,6 +359,7 @@ 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)
> @@ -370,6 +376,8 @@ 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);
>
>   /* Writing them here to avoid exposing memcg's inner layout */
>   #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> @@ -1402,6 +1410,10 @@ 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_bytes, RES_USAGE)>>  10,
> +  res_counter_read_u64(&memcg->kmem_bytes, RES_LIMIT)>>  10,
> +  res_counter_read_u64(&memcg->kmem_bytes, RES_FAILCNT));
>   }
>
>   /*
> @@ -3840,6 +3852,9 @@ 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_bytes, name);
> +  break;
>    default:
>     BUG();
>     break;
> @@ -3872,8 +3887,14 @@ 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 (!memcg->independent_kmem_limit)
> +    return -EINVAL;
```

```
> +   ret = res_counter_set_limit(&memcg->kmem_bytes, val);
> +  } else
> +   return -EINVAL;
>     break;
>    case RES_SOFT_LIMIT:
>     ret = res_counter_memparse_write_strategy(buffer,&val);
> @@ -4572,8 +4593,47 @@ 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_from_cont(cgrp)->independent_kmem_limit;
> +}
> +
> +static int mem_cgroup_independent_kmem_limit_write(struct cgroup *cgrp,
> +    struct cftype *cft, u64 val)
> +{
> + mem_cgroup_from_cont(cgrp)->independent_kmem_limit = !!val;
> +
> + return 0;
> +}
> +
> +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.
> @@ -4587,6 +4647,10 @@ 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_bytes, RES_USAGE) != 0);