Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Balbir Singh on Mon, 01 Oct 2007 14:07:27 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> The struct page gets an extra pointer (just like it has with
> the RSS controller) and this pointer points to the array of
> the kmem_container pointers - one for each object stored on
> that page itself.
>
> Thus the i'th object on the page is accounted to the container
> pointed by the i'th pointer on that array and when the object
> is freed we unaccount its size to this particular container,
> not the container current task belongs to.
>
> This is done so, because the context objects are freed is most
> often not the same as the one this objects was allocated in
> (due to RCU and reference counters).
>
> This controller disables the kmalloc caches accounting, since
> we cannot just track all the kmalloc calls, but we need to
> explicitly specify which kmalloc do we want to account for
> with (e.g.) additional GFP flag.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
> index f83bd17..8e1ef21 100644
> --- a/include/linux/mm_types.h
> +++ b/include/linux/mm_types.h
> @@ -83,9 +83,14 @@ struct page {
> void *virtual; /* Kernel virtual address (NULL if
> not kmapped, ie. highmem) */
> #endif /* WANT_PAGE_VIRTUAL */
> + union {
> #ifdef CONFIG_CGROUP_MEM_CONT
> - unsigned long page_cgroup;
> + unsigned long page_cgroup;
> +#endif
> +#ifdef CONFIG_CGROUP_KMEM
> + struct kmem_container **cgroups;
> #endif
> + };
> #ifdef CONFIG_PAGE_OWNER
> int order;
> unsigned int gfp_mask;

> diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
> index 40801e7..8cfd9ff 100644
> --- a/include/linux/slub_def.h
> +++ b/include/linux/slub_def.h
> @@ -69,6 +69,9 @@ struct kmem_cache {
> #endif
> };
>
> +int slab_index(void *p, struct kmem_cache *s, void *addr);
> +int is_kmalloc_cache(struct kmem_cache *s);
> +
> /*
>   * Kmalloc subsystem.
>   */
> diff --git a/mm/Makefile b/mm/Makefile
> index 81232a1..f7ec4b7 100644
> --- a/mm/Makefile
> +++ b/mm/Makefile
> @@ -31,4 +31,5 @@ obj-$(CONFIG_MIGRATION) += migrate.o
> obj-$(CONFIG_SMP) += allocpercpu.o
> obj-$(CONFIG_QUICKLIST) += quicklist.o
> obj-$(CONFIG_CGROUP_MEM_CONT) += memcontrol.o
> +obj-$(CONFIG_CGROUP_KMEM) += kmemcontrol.o
>
> diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
> new file mode 100644
> index 0000000..5698c0f
> --- /dev/null
> +++ b/mm/kmemcontrol.c
> @@ -1,6 +1,9 @@
>   * but WITHOUT ANY WARRANTY; without even the implied warranty of
>   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
>   * GNU General Public License for more details.
> + *
> + * Changelog:
> + *   2007 Pavel Emelyanov : Add slub accounting
>   */
>
> #include <linux/mm.h>
> @@ -120,3 +129,144 @@
>   .subsys_id = kmem_subsys_id,
>   .early_init = 1,
> };
> +
> +/*
> + * slub accounting
> + */
> +

```
> +int slub_newpage_notify(struct kmem_cache *s, struct page *pg, gfp_t flags)
> +{
> + struct kmem_container **ptr;
> +
> + ptr = kzalloc(s->objects * sizeof(struct kmem_container *), flags);
> + if (ptr == NULL)
> +  return -ENOMEM;
> +
> + pg->cgroups = ptr;
> + return 0;
> +}
> +
> +void slub_freepage_notify(struct kmem_cache *s, struct page *pg)
> +{
> + struct kmem_container **ptr;
> +
> + ptr = pg->cgroups;
> + if (ptr == NULL)
> +  return;
> +
> + kfree(ptr);
> + pg->cgroups = NULL;
> +}
> +
> +int slub_alloc_notify(struct kmem_cache *s, void *obj, gfp_t gfp)
> +{
> + struct page *pg;
> + struct kmem_container *cnt;
> + struct kmem_container **obj_container;
> +
> + pg = virt_to_head_page(obj);
> + obj_container = pg->cgroups;
> + if (unlikely(obj_container == NULL)) {
> +  /*
> +   * turned on after some objects were allocated
> +   */
> +  if (slub_newpage_notify(s, pg, GFP_ATOMIC) < 0)
> +   goto err;
> +
> +  obj_container = pg->cgroups;
> + }
> +
> + rcu_read_lock();
> + cnt = task_kmem_container(current);
> + if (res_counter_charge(&cnt->res, s->size))
```

Is s->size measure in pages or bytes? I suspect it is bytes.

```
> +  goto err_locked;
> +
> + css_get(&cnt->css);
> + rcu_read_unlock();
> + obj_container[slab_index(obj, s, page_address(pg))] = cnt;
> + return 0;
> +
> +err_locked:
> + rcu_read_unlock();
> +err:
> + return -ENOMEM;
> +}
> +
> +void slub_free_notify(struct kmem_cache *s, void *obj)
> +{
> + struct page *pg;
> + struct kmem_container *cnt;
> + struct kmem_container **obj_container;
> +
> + pg = virt_to_head_page(obj);
> + obj_container = pg->cgroups;
> + if (obj_container == NULL)
> +  return;
> +
> + obj_container += slab_index(obj, s, page_address(pg));
> + cnt = *obj_container;
> + if (cnt == NULL)
> +  return;
> +
> + res_counter_uncharge(&cnt->res, s->size);
> + *obj_container = NULL;
> + css_put(&cnt->css);
> +}
> +
```

Quick check, slub_free_notify() and slab_alloc_notify() are called
from serialized contexts, right?

```
> +int slub_on_notify(struct kmem_cache *cachep)
> +{
> + return (is_kmalloc_cache(cachep) ? -EINVAL : 0);
> +}
> +
```

I know you've mentioned several times in the comments that
kmalloc slab's cannot be accounted for, could you please
add a big comment here as well.

Also, looks like if I turn on notification for kmalloc
slab's, the listener fails, which will cause all
allocations to fail?

```
> +int slub_off_notify(struct kmem_cache *cachep)
> +{
> + return 0;
> +}
> +
> +#ifdef CONFIG_SLUB_NOTIFY
> +static int kmem_notify(struct notifier_block *nb, unsigned long cmd, void *arg)
> +{
> + int ret;
> + struct slub_notify_struct *ns;
> +
> + ns = (struct slub_notify_struct *)arg;
> +
> + switch (cmd) {
> + case SLUB_ALLOC:
> +  ret = slub_alloc_notify(ns->cachep, ns->objp, ns->gfp);
> +  break;
> + case SLUB_FREE:
> +  ret = 0;
> +  slub_free_notify(ns->cachep, ns->objp);
> +  break;
> + case SLUB_NEWPAGE:
> +  ret = slub_newpage_notify(ns->cachep, ns->objp, ns->gfp);
> +  break;
> + case SLUB_FREEPAGE:
> +  ret = 0;
> +  slub_freepage_notify(ns->cachep, ns->objp);
> +  break;
> + case SLUB_ON:
> +  ret = slub_on_notify(ns->cachep);
> +  break;
> + case SLUB_OFF:
> +  ret = slub_off_notify(ns->cachep);
> +  break;
> + default:
> +  return NOTIFY_DONE;
> + }
> +
> + return (ret < 0) ? notifier_from_errno(ret) : NOTIFY_OK;
> +}
> +
> +static struct notifier_block kmem_block = {
> + .notifier_call = kmem_notify,
> +};
```

```
> +
> +static int kmem_subsys_register(void)
> +{
> + return slub_register_notifier(&kmem_block);
> +}
> +
> +__initcall(kmem_subsys_register);
> +#endif
> diff --git a/mm/slub.c b/mm/slub.c
> index 31d04a3..e066a0e 100644
> --- a/mm/slub.c
> +++ b/mm/slub.c
> @@ -327,7 +327,7 @@ static inline void set_freepointer(struc
>   for (__p = (__free); __p; __p = get_freepointer((__s), __p))
>
>  /* Determine object index from a given position */
> -static inline int slab_index(void *p, struct kmem_cache *s, void *addr)
> +inline int slab_index(void *p, struct kmem_cache *s, void *addr)
>  {
>   return (p - addr) / s->size;
>  }
> @@ -2360,6 +2504,14 @@ EXPORT_SYMBOL(kmem_cache_destroy);
>  struct kmem_cache kmalloc_caches[PAGE_SHIFT] __cacheline_aligned;
>  EXPORT_SYMBOL(kmalloc_caches);
>
> +int is_kmalloc_cache(struct kmem_cache *s)
> +{
> + int km_idx;
> +
> + km_idx = s - kmalloc_caches;
> + return km_idx >= 0 && km_idx < ARRAY_SIZE(kmalloc_caches);
> +}
> +
```

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL