
Subject: Re: [PATCH v3 07/13] mm: Allocate kernel pages to the right memcg
Posted by [Mel Gorman](#) on Thu, 27 Sep 2012 13:50:53 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Tue, Sep 18, 2012 at 06:04:04PM +0400, Glauber Costa wrote:

```
> When a process tries to allocate a page with the __GFP_KMEMCG flag, the
> page allocator will call the corresponding memcg functions to validate
> the allocation. Tasks in the root memcg can always proceed.
>
> To avoid adding markers to the page - and a kmem flag that would
> necessarily follow, as much as doing page_cgroup lookups for no reason,
> whoever is marking its allocations with __GFP_KMEMCG flag is responsible
> for telling the page allocator that this is such an allocation at
> free_pages() time. This is done by the invocation of
> __free_accounted_pages() and free_accounted_pages().
>
> [ v2: inverted test order to avoid a memcg_get leak,
>   free_accounted_pages simplification ]
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Christoph Lameter <cl@linux.com>
> CC: Pekka Enberg <penberg@cs.helsinki.fi>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Suleiman Souhlal <suleiman@google.com>
> CC: Mel Gorman <mgorman@suse.de>
> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> ---
> include/linux/gfp.h | 3 +++
> mm/page_alloc.c     | 35 +++++++++++++++++++++++++++++++++++++
> 2 files changed, 38 insertions(+)
>
> diff --git a/include/linux/gfp.h b/include/linux/gfp.h
> index d8eae4d..029570f 100644
> --- a/include/linux/gfp.h
> +++ b/include/linux/gfp.h
> @@ -370,6 +370,9 @@ extern void free_pages(unsigned long addr, unsigned int order);
> extern void free_hot_cold_page(struct page *page, int cold);
> extern void free_hot_cold_page_list(struct list_head *list, int cold);
>
> +extern void __free_accounted_pages(struct page *page, unsigned int order);
> +extern void free_accounted_pages(unsigned long addr, unsigned int order);
> +
> #define __free_page(page) __free_pages((page), 0)
> #define free_page(addr) free_pages((addr), 0)
>
> diff --git a/mm/page_alloc.c b/mm/page_alloc.c
> index b0c5a52..897d8e2 100644
```

```

> --- a/mm/page_alloc.c
> +++ b/mm/page_alloc.c
> @@ -2573,6 +2573,7 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
> struct page *page = NULL;
> int migratetype = allocflags_to_migratetype(gfp_mask);
> unsigned int cpuset_mems_cookie;
> + struct mem_cgroup *memcg = NULL;
>
> gfp_mask &= gfp_allowed_mask;
>
> @@ -2591,6 +2592,13 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
> if (unlikely(!zonelist->_zonerefs->zone))
> return NULL;
>
> + /*
> + * Will only have any effect when __GFP_KMEMCG is set. This is
> + * verified in the (always inline) callee
> + */
> + if (!memcg_kmem_newpage_charge(gfp_mask, &memcg, order))
> + return NULL;
> +

```

1. returns quickly if memcg disabled
2. returns quickly if !__GFP_KMEMCG
3. returns quickly for kernel threads and interrupts

I'm expecting that these paths are completely dead when kmem accounting is off so

Acked-by: Mel Gorman <mgorman@suse.de>

That said, it's not directly related to this patch but I would suggest that you also check for TIF_MEMDIE in memcg_kmem_newpage_charge. It would be very silly if a process failed to exit because it couldn't allocate a page it needed. I expect that such a case is impossible today but it might change in the future. If you're doing another revision, an extra check would not hurt.

It's difficult to predict if it should be making all the checks that gfp_to_alloc_flags() does but you might need to in the future so keep it in mind.

```

> retry_cpuset:
> cpuset_mems_cookie = get_mems_allowed();
>

```

```

> @@ -2624,6 +2632,8 @@ out:
> if (unlikely(!put_mems_allowed(cpuset_mems_cookie) && !page))
> goto retry_cpuset;
>
> + memcg_kmem_commit_charge(page, memcg, order);
> +
> return page;

```

Ok.

```

> }
> EXPORT_SYMBOL(__alloc_pages_nodemask);
> @@ -2676,6 +2686,31 @@ void free_pages(unsigned long addr, unsigned int order)
>
> EXPORT_SYMBOL(free_pages);
>
> +/*
> + * __free_accounted_pages and free_accounted_pages will free pages allocated
> + * with __GFP_KMEMCG.
> + *
> + * Those pages are accounted to a particular memcg, embedded in the
> + * corresponding page_cgroup. To avoid adding a hit in the allocator to search
> + * for that information only to find out that it is NULL for users who have no
> + * interest in that whatsoever, we provide these functions.
> + *
> + * The caller knows better which flags it relies on.
> + */
> +void __free_accounted_pages(struct page *page, unsigned int order)
> +{
> + memcg_kmem_uncharge_page(page, order);
> + __free_pages(page, order);
> +}
> +
> +void free_accounted_pages(unsigned long addr, unsigned int order)
> +{
> + if (addr != 0) {
> + VM_BUG_ON(!virt_addr_valid((void *)addr));

```

This is probably overkill. If it's invalid, the next line is likely to blow up anyway. It's no biggie.

```

> + __free_accounted_pages(virt_to_page((void *)addr), order);
> + }
> +}
> +
> static void *make_alloc_exact(unsigned long addr, unsigned order, size_t size)
> {
> if (addr) {

```

> --
> 1.7.11.4
>

--
Mel Gorman
SUSE Labs
