Subject: Re: [PATCH v2 06/11] memcg: kmem controller infrastructure
Posted by Greg Thelen on Sat, 11 Aug 2012 05:11:22 GMT
View Forum Message <> Reply to Message

On Thu, Aug 09 2012, Glauber Costa wrote:

> This patch introduces infrastructure for tracking kernel memory pages to
> a given memcg. This will happen whenever the caller includes the flag
> __GFP_KMEMCG flag, and the task belong to a memcg other than the root.
>
> In memcontrol.h those functions are wrapped in inline accessors. The
> idea is to later on, patch those with static branches, so we don't incur
> any overhead when no mem cgroups with limited kmem are being used.
>
> [v2: improved comments and standardized function names]
>
> 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: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> ---
> include/linux/memcontrol.h | 79 ++++++++++++++++++++++
> mm/memcontrol.c | 185 ++
> 2 files changed, 264 insertions(+)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 8d9489f..75b247e 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -21,6 +21,7 @@
> #define _LINUX_MEMCONTROL_H
> #include <linux/cgroup.h>
> #include <linux/vm_event_item.h>
> +#include <linux/hardirq.h>
>
> struct mem_cgroup;
> struct page_cgroup;
> @@ -399,6 +400,11 @@ struct sock;
> #ifdef CONFIG_MEMCG_KMEM
> void sock_update_memcg(struct sock *sk);
> void sock_release_memcg(struct sock *sk);
> +
> +#define memcg_kmem_on 1
> +bool __memcg_kmem_new_page(gfp_t gfp, void *handle, int order);
> +void __memcg_kmem_commit_page(struct page *page, void *handle, int order);
> +void __memcg_kmem_free_page(struct page *page, int order);

> #else
> static inline void sock_update_memcg(struct sock *sk)
> {
> @@ -406,6 +412,79 @@ static inline void sock_update_memcg(struct sock *sk)
> static inline void sock_release_memcg(struct sock *sk)
> {
> }
> +
> +#define memcg_kmem_on 0
> +static inline bool
> +__memcg_kmem_new_page(gfp_t gfp, void *handle, int order)
> +{
> + return false;
> +}
> +
> +static inline void __memcg_kmem_free_page(struct page *page, int order)
> +{
> +}
> +
> +static inline void
> +__memcg_kmem_commit_page(struct page *page, struct mem_cgroup *handle, int order)
> +{
> +}
> #endif /* CONFIG_MEMCG_KMEM */
> +
> +/**
> + * memcg_kmem_new_page: verify if a new kmem allocation is allowed.
> + * @gfp: the gfp allocation flags.
> + * @handle: a pointer to the memcg this was charged against.
> + * @order: allocation order.
> + *
> + * returns true if the memcg where the current task belongs can hold this
> + * allocation.
> + *
> + * We return true automatically if this allocation is not to be accounted to
> + * any memcg.
> + */
> +static __always_inline bool
> +memcg_kmem_new_page(gfp_t gfp, void *handle, int order)
> +{
> + if (!memcg_kmem_on)
> + return true;
> + if (!(gfp & __GFP_KMEMCG) || (gfp & __GFP_NOFAIL))
> + return true;
> + if (in_interrupt() || (!current->mm) || (current->flags & PF_KTHREAD))
> + return true;
> + return __memcg_kmem_new_page(gfp, handle, order);
> +}

> +
> +/**
> + * memcg_kmem_free_page: uncharge pages from memcg
> + * @page: pointer to struct page being freed
> + * @order: allocation order.
> + *
> + * there is no need to specify memcg here, since it is embedded in page_cgroup
> + */
> +static __always_inline void
> +memcg_kmem_free_page(struct page *page, int order)
> +{
> + if (memcg_kmem_on)
> + __memcg_kmem_free_page(page, order);
> +}
> +
> +/**
> + * memcg_kmem_commit_page: embeds correct memcg in a page
> + * @handle: a pointer to the memcg this was charged against.
> + * @page: pointer to struct page recently allocated
> + * @handle: the memcg structure we charged against
> + * @order: allocation order.
> + *
> + * Needs to be called after memcg_kmem_new_page, regardless of success or
> + * failure of the allocation. if @page is NULL, this function will revert the
> + * charges. Otherwise, it will commit the memcg given by @handle to the
> + * corresponding page_cgroup.
> + */
> +static __always_inline void
> +memcg_kmem_commit_page(struct page *page, struct mem_cgroup *handle, int order)
> +{
> + if (memcg_kmem_on)
> + __memcg_kmem_commit_page(page, handle, order);
> +}
> #endif /* _LINUX_MEMCONTROL_H */
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 54e93de..e9824c1 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -10,6 +10,10 @@
>   * Copyright (C) 2009 Nokia Corporation
>   * Author: Kirill A. Shutemov
>   *
> + * Kernel Memory Controller
> + * Copyright (C) 2012 Parallels Inc. and Google Inc.
> + * Authors: Glauber Costa and Suleiman Souhlal
> + *
>   * This program is free software; you can redistribute it and/or modify

>   * it under the terms of the GNU General Public License as published by
>   * the Free Software Foundation; either version 2 of the License, or
> @@ -434,6 +438,9 @@ struct mem_cgroup *mem_cgroup_from_css(struct
cgroup_subsys_state *s)
> #include <net/ip.h>
>
> static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
> +static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta);
> +static void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta);
> +
> void sock_update_memcg(struct sock *sk)
> {
>   if (mem_cgroup_sockets_enabled) {
> @@ -488,6 +495,118 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
> }
> EXPORT_SYMBOL(tcp_proto_cgroup);
> #endif /* CONFIG_INET */
> +
> +static inline bool memcg_kmem_enabled(struct mem_cgroup *memcg)
> +{
> + return !mem_cgroup_disabled() && !mem_cgroup_is_root(memcg) &&
> + memcg->kmem_accounted;
> +}
> +
> +/*
> + * We need to verify if the allocation against current->mm->owner's memcg is
> + * possible for the given order. But the page is not allocated yet, so we'll
> + * need a further commit step to do the final arrangements.
> + *
> + * It is possible for the task to switch cgroups in this mean time, so at
> + * commit time, we can't rely on task conversion any longer. We'll then use
> + * the handle argument to return to the caller which cgroup we should commit
> + * against
> + *
> + * Returning true means the allocation is possible.
> + */
> +bool __memcg_kmem_new_page(gfp_t gfp, void *_handle, int order)
> +{
> + struct mem_cgroup *memcg;
> + struct mem_cgroup **handle = (struct mem_cgroup **)_handle;
> + bool ret = true;
> + size_t size;
> + struct task_struct *p;
> +
> + *handle = NULL;
> + rcu_read_lock();
> + p = rcu_dereference(current->mm->owner);
> + memcg = mem_cgroup_from_task(p);

> + if (!memcg_kmem_enabled(memcg))
> + goto out;
> +
> + mem_cgroup_get(memcg);
> +
> + size = PAGE_SIZE << order;
> + ret = memcg_charge_kmem(memcg, gfp, size) == 0;
> + if (!ret) {
> + mem_cgroup_put(memcg);
> + goto out;
> + }
> +
> + *handle = memcg;
> +out:
> + rcu_read_unlock();
> + return ret;
> +}
> +EXPORT_SYMBOL(__memcg_kmem_new_page);

While running f853d89 from git://github.com/glommer/linux.git , I hit a lockdep issue. To create this I allocated and held reference to some kmem in the context of a kmem limited memcg. Then I moved the allocating process out of memcg and then deleted the memcg. Due to the kmem reference the struct mem_cgroup is still active but invisible in cgroupfs namespace. No problems yet. Then I killed the user process which freed the kmem from the now unlinked memcg. Dropping the kmem caused the memcg ref to hit zero. Then the memcg is deleted but that acquires a non-irqsafe spinlock in softirq which annoys lockdep. I think the lock in question is the mctz below:

```
mem_cgroup_remove_exceeded(struct mem_cgroup *memcg,
    struct mem_cgroup_per_zone *mz,
    struct mem_cgroup_tree_per_zone *mctz)
{
 spin_lock(&mctz->lock);
 __mem_cgroup_remove_exceeded(memcg, mz, mctz);
 spin_unlock(&mctz->lock);
}
```

Perhaps your patches expose this problem by being the first time we call __mem_cgroup_free() from softirq (this is just an educated guess). I'm not sure how this would interact with Ying's soft limit rework: https://lwn.net/Articles/501338/

Here's the dmesg splat.

```
[  335.550398] =================================
[  335.554739] [ INFO: inconsistent lock state ]
```

```
[  335.559091] 3.5.0-dbg-DEV #3 Tainted: G        W
[  335.563946] --------------------------------
[  335.568290] inconsistent {SOFTIRQ-ON-W} -> {IN-SOFTIRQ-W} usage.
[  335.574286] swapper/10/0 [HC0[0]:SC1[1]:HE1:SE0] takes:
[  335.579508]  (&(&rtpz->lock)->rlock){+.?...}, at: [<ffffffff8118216d>]
__mem_cgroup_free+0x8d/0x1b0
[  335.588525] {SOFTIRQ-ON-W} state was registered at:
[  335.593389]   [<ffffffff810cb073>] __lock_acquire+0x623/0x1a50
[  335.599200]   [<ffffffff810cca55>] lock_acquire+0x95/0x150
[  335.604670]   [<ffffffff81582531>] _raw_spin_lock+0x41/0x50
[  335.610232]   [<ffffffff8118216d>] __mem_cgroup_free+0x8d/0x1b0
[  335.616135]   [<ffffffff811822d5>] mem_cgroup_put+0x45/0x50
[  335.621696]   [<ffffffff81182302>] mem_cgroup_destroy+0x22/0x30
[  335.627592]   [<ffffffff810e093f>] cgroup_diput+0xbf/0x160
[  335.633062]   [<ffffffff811a07ef>] d_delete+0x12f/0x1a0
[  335.638276]   [<ffffffff8119671e>] vfs_rmdir+0x11e/0x140
[  335.643565]   [<ffffffff81199173>] do_rmdir+0x113/0x130
[  335.648773]   [<ffffffff8119a5e6>] sys_rmdir+0x16/0x20
[  335.653900]   [<ffffffff8158c74f>] cstar_dispatch+0x7/0x1f
[  335.659370] irq event stamp: 399732
[  335.662846] hardirqs last  enabled at (399732): [<ffffffff810e8e08>]
res_counter_uncharge_until+0x68/0xa0
[  335.672383] hardirqs last disabled at (399731): [<ffffffff810e8dc8>]
res_counter_uncharge_until+0x28/0xa0
[  335.681916] softirqs last  enabled at (399710): [<ffffffff81085dd3>]
_local_bh_enable+0x13/0x20
[  335.690590] softirqs last disabled at (399711): [<ffffffff8158c48c>] call_softirq+0x1c/0x30
[  335.698914]
[  335.698914] other info that might help us debug this:
[  335.705415]  Possible unsafe locking scenario:
[  335.705415]
[  335.711317]        CPU0
[  335.713757]        ----
[  335.716198]   lock(&(&rtpz->lock)->rlock);
[  335.720282]   <Interrupt>
[  335.722896]     lock(&(&rtpz->lock)->rlock);
[  335.727153]
[  335.727153]  *** DEADLOCK ***
[  335.727153]
[  335.733055] no locks held by swapper/10/0.
[  335.737141]
[  335.737141] stack backtrace:
[  335.741483] Pid: 0, comm: swapper/10 Tainted: G        W    3.5.0-dbg-DEV #3
[  335.748510] Call Trace:
[  335.750952]  <IRQ>  [<ffffffff81579a27>] print_usage_bug+0x1fc/0x20d
[  335.757286]  [<ffffffff81058a9f>] ? save_stack_trace+0x2f/0x50
[  335.763098]  [<ffffffff810ca9ed>] mark_lock+0x29d/0x300
[  335.768309]  [<ffffffff810c9e10>] ? print_irq_inversion_bug.part.36+0x1f0/0x1f0
```

[335.775599] [<ffffffff810caffc>] __lock_acquire+0x5ac/0x1a50
[335.781323] [<ffffffff810cad34>] ? __lock_acquire+0x2e4/0x1a50
[335.787224] [<ffffffff8118216d>] ? __mem_cgroup_free+0x8d/0x1b0
[335.793212] [<ffffffff810cca55>] lock_acquire+0x95/0x150
[335.798594] [<ffffffff8118216d>] ? __mem_cgroup_free+0x8d/0x1b0
[335.804581] [<ffffffff810e8ddd>] ? res_counter_uncharge_until+0x3d/0xa0
[335.811263] [<ffffffff81582531>] _raw_spin_lock+0x41/0x50
[335.816731] [<ffffffff8118216d>] ? __mem_cgroup_free+0x8d/0x1b0
[335.822724] [<ffffffff8118216d>] __mem_cgroup_free+0x8d/0x1b0
[335.828538] [<ffffffff811822d5>] mem_cgroup_put+0x45/0x50
[335.834002] [<ffffffff811828a6>] __memcg_kmem_free_page+0xa6/0x110
[335.840256] [<ffffffff81138109>] free_accounted_pages+0x99/0xa0
[335.846243] [<ffffffff8107b09f>] free_task+0x3f/0x70
[335.851278] [<ffffffff8107b18c>] __put_task_struct+0xbc/0x130
[335.857094] [<ffffffff81081524>] delayed_put_task_struct+0x54/0xd0
[335.863338] [<ffffffff810fd354>] __rcu_process_callbacks+0x1e4/0x490
[335.869757] [<ffffffff810fd62f>] rcu_process_callbacks+0x2f/0x80
[335.875835] [<ffffffff810862f5>] __do_softirq+0xc5/0x270
[335.881218] [<ffffffff810c49b4>] ? clockevents_program_event+0x74/0x100
[335.887895] [<ffffffff810c5d94>] ? tick_program_event+0x24/0x30
[335.893882] [<ffffffff8158c48c>] call_softirq+0x1c/0x30
[335.899179] [<ffffffff8104cefd>] do_softirq+0x8d/0xc0
[335.904301] [<ffffffff810867de>] irq_exit+0xae/0xe0
[335.909251] [<ffffffff8158cc3e>] smp_apic_timer_interrupt+0x6e/0x99
[335.915591] [<ffffffff8158ba9c>] apic_timer_interrupt+0x6c/0x80
[335.921583] <EOI> [<ffffffff810530e7>] ? default_idle+0x67/0x270
[335.927741] [<ffffffff810530e5>] ? default_idle+0x65/0x270