

Hi,

What I am proposing with this series is a stripped down version of the kmem controller for memcg that would allow us to merge significant parts of the infrastructure, while leaving out, for now, the polemic bits about the slab while it is being reworked by Cristoph.

Me reasoning for that is that after the last change to introduce a gfp flag to mark kernel allocations, it became clear to me that tracking other resources like the stack would then follow extremely naturally. I figured that at some point we'd have to solve the issue pointed by David, and avoid testing the Slab flag in the page allocator, since it would soon be made more generic. I do that by having the callers to explicit mark it.

So to demonstrate how it would work, I am introducing a stack tracker here, that is already a functionality per-se: it successfully stops fork bombs to happen. (Sorry for doing all your work, Frederic =p ). Note that after all memcg infrastructure is deployed, it becomes very easy to track anything. The last patch of this series is extremely simple.

The infrastructure is exactly the same we had in memcg, but stripped down of the slab parts. And because what we have after those patches is a feature per-se, I think it could be considered for merging.

Let me know what you think.

Glauber Costa (9):

- memcg: change defines to an enum
- kmem slab accounting basic infrastructure
- Add a \_\_GFP\_KMEMCG flag
- memcg: kmem controller infrastructure
- mm: Allocate kernel pages to the right memcg
- memcg: disable kmem code when not in use.
- memcg: propagate kmem limiting information to children
- memcg: allow a memcg with kmem charges to be destructed.
- protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Suleiman Souhlal (2):

- memcg: Make it possible to use the stock for more than one page.
- memcg: Reclaim when more than one page needed.

```
include/linux/gfp.h      | 11 +-  
include/linux/memcontrol.h | 46 +++++
```

```
include/linux/thread_info.h | 6 +
kernel/fork.c                | 4 +-
mm/memcontrol.c              | 395 ++++++-----
mm/page_alloc.c              | 27 +++
6 files changed, 464 insertions(+), 25 deletions(-)
```

--  
1.7.10.2

---

---

Subject: [PATCH 01/11] memcg: Make it possible to use the stock for more than one page.

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:18 GMT

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

---

From: Suleiman Souhlal <[ssouhlal@FreeBSD.org](mailto:ssouhlal@FreeBSD.org)>

Signed-off-by: Suleiman Souhlal <[suleiman@google.com](mailto:suleiman@google.com)>

Signed-off-by: Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)>

Acked-by: Kamezawa Hiroyuki <[kamezawa.hiroyu@jp.fujitsu.com](mailto:kamezawa.hiroyu@jp.fujitsu.com)>

---

```
mm/memcontrol.c | 21 ++++++-----
1 file changed, 12 insertions(+), 9 deletions(-)
```

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index f72b5e5..9304db2 100644
```

```
--- a/mm/memcontrol.c
```

```
+++ b/mm/memcontrol.c
```

```
@ @ -1967,19 +1967,22 @ @ static DEFINE_PER_CPU(struct memcg_stock_pcp,
memcg_stock);
static DEFINE_MUTEX(percpu_charge_mutex);
```

```
/*
```

```
- * Try to consume stocked charge on this cpu. If success, one page is consumed
- * from local stock and true is returned. If the stock is 0 or charges from a
- * cgroup which is not current target, returns false. This stock will be
- * refilled.
```

```
+ * Try to consume stocked charge on this cpu. If success, nr_pages pages are
+ * consumed from local stock and true is returned. If the stock is 0 or
+ * charges from a cgroup which is not current target, returns false.
+ * This stock will be refilled.
```

```
*/
```

```
-static bool consume_stock(struct mem_cgroup *memcg)
```

```
+static bool consume_stock(struct mem_cgroup *memcg, int nr_pages)
```

```
{
    struct memcg_stock_pcp *stock;
    bool ret = true;
```

```

+ if (nr_pages > CHARGE_BATCH)
+ return false;
+
  stock = &get_cpu_var(memcg_stock);
- if (memcg == stock->cached && stock->nr_pages)
- stock->nr_pages--;
+ if (memcg == stock->cached && stock->nr_pages >= nr_pages)
+ stock->nr_pages -= nr_pages;
  else /* need to call res_counter_charge */
    ret = false;
  put_cpu_var(memcg_stock);
@@ -2278,7 +2281,7 @@ again:
  VM_BUG_ON(css_is_removed(&memcg->css));
  if (mem_cgroup_is_root(memcg))
    goto done;
- if (nr_pages == 1 && consume_stock(memcg))
+ if (consume_stock(memcg, nr_pages))
  goto done;
  css_get(&memcg->css);
} else {
@@ -2303,7 +2306,7 @@ again:
  rcu_read_unlock();
  goto done;
}
- if (nr_pages == 1 && consume_stock(memcg)) {
+ if (consume_stock(memcg, nr_pages)) {
  /*
   * It seems dangerous to access memcg without css_get().
   * But considering how consume_stok works, it's not
  --
1.7.10.2

```

---

Subject: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
 Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:19 GMT

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

---

From: Suleiman Souhlal <ssouhlal@FreeBSD.org>

mem\_cgroup\_do\_charge() was written before slab accounting, and expects three cases: being called for 1 page, being called for a stock of 32 pages, or being called for a hugepage. If we call for 2 or 3 pages (and several slabs used in process creation are such, at least with the debug options I had), it assumed it's being called for stock and just retried without reclaiming.

Fix that by passing down a minsize argument in addition to the csize.

And what to do about that (csize == PAGE\_SIZE && ret) retry? If it's

needed at all (and presumably is since it's there, perhaps to handle races), then it should be extended to more than PAGE\_SIZE, yet how far? And should there be a retry count limit, of what? For now retry up to COSTLY\_ORDER (as page\_alloc.c does), stay safe with a cond\_resched(), and make sure not to do it if \_\_GFP\_NORETRY.

[v4: fixed nr pages calculation pointed out by Christoph Lameter ]

Signed-off-by: Suleiman Souhlal <suleiman@google.com>

Signed-off-by: Glauber Costa <glommer@parallels.com>

Reviewed-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---

mm/memcontrol.c | 23 ++++++-----

1 file changed, 16 insertions(+), 7 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 9304db2..8e601e8 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -2158,8 +2158,16 @@ enum {

CHARGE\_OOM\_DIE, /\* the current is killed because of OOM \*/

};

+/\*

+ \* We need a number that is small enough to be likely to have been

+ \* reclaimed even under pressure, but not too big to trigger unnecessary

+ \* retries

+ \*/

+#define NR\_PAGES\_TO\_RETRY 2

+

static int mem\_cgroup\_do\_charge(struct mem\_cgroup \*memcg, gfp\_t gfp\_mask,

- unsigned int nr\_pages, bool oom\_check)

+ unsigned int nr\_pages, unsigned int min\_pages,

+ bool oom\_check)

{

unsigned long csize = nr\_pages \* PAGE\_SIZE;

struct mem\_cgroup \*mem\_over\_limit;

@@ -2182,18 +2190,18 @@ static int mem\_cgroup\_do\_charge(struct mem\_cgroup \*memcg, gfp\_t gfp\_mask,

} else

mem\_over\_limit = mem\_cgroup\_from\_res\_counter(fail\_res, res);

/\*

- \* nr\_pages can be either a huge page (HPAGE\_PMD\_NR), a batch

- \* of regular pages (CHARGE\_BATCH), or a single regular page (1).

- \*

\* Never reclaim on behalf of optional batching, retry with a

\* single page instead.

\*/

```

- if (nr_pages == CHARGE_BATCH)
+ if (nr_pages > min_pages)
    return CHARGE_RETRY;

    if (!(gfp_mask & __GFP_WAIT))
        return CHARGE_WOULDBLOCK;

+ if (gfp_mask & __GFP_NORETRY)
+ return CHARGE_NOMEM;
+
    ret = mem_cgroup_reclaim(mem_over_limit, gfp_mask, flags);
    if (mem_cgroup_margin(mem_over_limit) >= nr_pages)
        return CHARGE_RETRY;
@@ -2206,7 +2214,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t
gfp_mask,
    * unlikely to succeed so close to the limit, and we fall back
    * to regular pages anyway in case of failure.
    */
- if (nr_pages == 1 && ret)
+ if (nr_pages <= NR_PAGES_TO_RETRY && ret)
    return CHARGE_RETRY;

/*
@@ -2341,7 +2349,8 @@ again:
    nr_oom_retries = MEM_CGROUP_RECLAIM_RETRIES;
}

- ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, oom_check);
+ ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, nr_pages,
+ oom_check);
    switch (ret) {
    case CHARGE_OK:
        break;
--
1.7.10.2

```

---

Subject: [PATCH 03/11] memcg: change defines to an enum  
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:20 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This is just a cleanup patch for clarity of expression.  
In earlier submissions, people asked it to be in a separate  
patch, so here it is.

Signed-off-by: Glauber Costa <glommer@parallels.com>  
CC: Michal Hocko <mhocko@suse.cz>  
CC: Johannes Weiner <hannes@cmpxchg.org>

Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---

mm/memcontrol.c | 9 ++++++---

1 file changed, 6 insertions(+), 3 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 8e601e8..9352d40 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -387,9 +387,12 @@ enum charge\_type {  
};

/\* for encoding cft->private value on file \*/

+#define \_MEM (0)

+#define \_MEMSWAP (1)

+#define \_OOM\_TYPE (2)

+enum res\_type {

+ \_MEM,

+ \_MEMSWAP,

+ \_OOM\_TYPE,

+};

+

#define MEMFILE\_PRIVATE(x, val) ((x) << 16 | (val))

#define MEMFILE\_TYPE(val) ((val) >> 16 & 0xffff)

#define MEMFILE\_ATTR(val) ((val) & 0xffff)

--

1.7.10.2

---

Subject: [PATCH 04/11] kmem slab accounting basic infrastructure

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:21 GMT

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

---

This patch adds the basic infrastructure for the accounting of the slab caches. To control that, the following files are created:

- \* memory.kmem.usage\_in\_bytes
- \* memory.kmem.limit\_in\_bytes
- \* memory.kmem.failcnt
- \* memory.kmem.max\_usage\_in\_bytes

They have the same meaning of their user memory counterparts. They reflect the state of the "kmem" res\_counter.

The code is not enabled until a limit is set. This can be tested by the flag "kmem\_accounted". This means that after the patch is applied, no behavioral changes exists for whoever is still using memcg to control their memory usage.

We always account to both user and kernel resource\_counters. This effectively means that an independent kernel limit is in place when the limit is set to a lower value than the user memory. A equal or higher value means that the user limit will always hit first, meaning that kmem is effectively unlimited.

People who want to track kernel memory but not limit it, can set this limit to a very high number (like RESOURCE\_MAX - 1page - that no one will ever hit, or equal to the user memory)

Signed-off-by: Glauber Costa <glommer@parallels.com>

CC: Michal Hocko <mhocko@suse.cz>

CC: Johannes Weiner <hannes@cmpxchg.org>

Reviewed-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---  
mm/memcontrol.c | 78 ++  
1 file changed, 77 insertions(+), 1 deletion(-)

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 %lluB, limit %lluB, 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
    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);
--
1.7.10.2

```

---

Subject: [PATCH 05/11] Add a \_\_GFP\_KMEMCG flag  
 Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:22 GMT

This flag is used to indicate to the callees that this allocation will be serviced to the kernel. It is not supposed to be passed by the callers of `kmem_cache_alloc`, but rather by the cache core itself.

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>  
CC: Suleiman Souhlal <suleiman@google.com>

---

include/linux/gfp.h | 8 ++++++--  
1 file changed, 7 insertions(+), 1 deletion(-)

```
diff --git a/include/linux/gfp.h b/include/linux/gfp.h
index 1e49be4..8f4079f 100644
--- a/include/linux/gfp.h
+++ b/include/linux/gfp.h
@@ -37,6 +37,9 @@ struct vm_area_struct;
#define __GFP_NO_KSWAPD 0x400000u
#define __GFP_OTHER_NODE 0x800000u
#define __GFP_WRITE 0x1000000u
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#define __GFP_KMEMCG 0x2000000u
+#endif

/*
 * GFP bitmasks..
@@ -88,13 +91,16 @@ struct vm_area_struct;
#define __GFP_OTHER_NODE ((__force gfp_t) __GFP_OTHER_NODE) /* On behalf of other
node */
#define __GFP_WRITE ((__force gfp_t) __GFP_WRITE) /* Allocator intends to dirty page */

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#define __GFP_KMEMCG ((__force gfp_t) __GFP_KMEMCG) /* Allocation comes from a
memcg-accounted resource */
+#endif

/*
 * This may seem redundant, but it's a way of annotating false positives vs.
 * allocations that simply cannot be supported (e.g. page tables).
 */
#define __GFP_NOTRACK_FALSE_POSITIVE (__GFP_NOTRACK)

-#define __GFP_BITS_SHIFT 25 /* Room for N __GFP_FOO bits */
+#define __GFP_BITS_SHIFT 26 /* Room for N __GFP_FOO bits */
#define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))
```

```
/* This equals 0, but use constants in case they ever change */
```

```
--
```

```
1.7.10.2
```

---

Subject: [PATCH 06/11] memcg: kmem controller infrastructure

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:23 GMT

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

---

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 jump labels, so we don't incur any overhead when no mem cgroups are being used.

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 | 44 ++++++++
mm/memcontrol.c             | 172 +++++++++++++++++++++++++++++++++++++
2 files changed, 216 insertions(+)
```

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
```

```
index 83e7ba9..22479eb 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;
```

```
@ @ -409,6 +410,12 @ @ struct sock;
```

```
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
```

```
void sock_update_memcg(struct sock *sk);
```

```
void sock_release_memcg(struct sock *sk);
```

```
+
```

```
+#define mem_cgroup_kmem_on 1
```

```
+bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
```

```

+void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
+void __mem_cgroup_free_kmem_page(struct page *page, int order);
+#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)
+else
+static inline void sock_update_memcg(struct sock *sk)
+{
+@@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
+static inline void sock_release_memcg(struct sock *sk)
+{
+}
+
+#define mem_cgroup_kmem_on 0
+#define __mem_cgroup_new_kmem_page(a, b, c) false
+#define __mem_cgroup_free_kmem_page(a,b )
+#define __mem_cgroup_commit_kmem_page(a, b, c)
+#define is_kmem_tracked_alloc (false)
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+static __always_inline
+bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
+{
+if (!mem_cgroup_kmem_on)
+return true;
+if (!is_kmem_tracked_alloc)
+return true;
+if (!current->mm)
+return true;
+if (in_interrupt())
+return true;
+if (gfp & __GFP_NOFAIL)
+return true;
+return __mem_cgroup_new_kmem_page(gfp, handle, order);
+}
+
+static __always_inline
+void mem_cgroup_free_kmem_page(struct page *page, int order)
+{
+if (mem_cgroup_kmem_on)
+__mem_cgroup_free_kmem_page(page, order);
+}
+
+static __always_inline
+void mem_cgroup_commit_kmem_page(struct page *page, struct mem_cgroup *handle,
+int order)
+{
+if (mem_cgroup_kmem_on)
+__mem_cgroup_commit_kmem_page(page, handle, order);
+}

```

```
#endif /* _LINUX_MEMCONTROL_H */
```

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
```

```
index 6f34b77..27b2b6f 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
```

```
@@ -422,6 +426,9 @@ static void mem_cgroup_put(struct mem_cgroup *memcg);
```

```
#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) {
```

```
@@ -476,6 +483,105 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
```

```
}
```

```
EXPORT_SYMBOL(tcp_proto_cgroup);
```

```
#endif /* CONFIG_INET */
```

```
+
```

```
+static inline bool mem_cgroup_kmem_enabled(struct mem_cgroup *memcg)
```

```
+{
```

```
+ return !mem_cgroup_disabled() && memcg &&
```

```
+     !mem_cgroup_is_root(memcg) && memcg->kmem_accounted;
```

```
+}
```

```
+
```

```
+bool __mem_cgroup_new_kmem_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 (!mem_cgroup_kmem_enabled(memcg))
+ goto out;
+
+ mem_cgroup_get(memcg);
+
+ size = (1 << order) << PAGE_SHIFT;
+ 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(__mem_cgroup_new_kmem_page);
+
+void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order)
+{
+ struct page_cgroup *pc;
+ struct mem_cgroup *memcg = handle;
+ size_t size;
+
+ if (!memcg)
+ return;
+
+ WARN_ON(mem_cgroup_is_root(memcg));
+ /* The page allocation must have failed. Revert */
+ if (!page) {
+ size = (1 << order) << PAGE_SHIFT;
+ memcg_uncharge_kmem(memcg, size);
+ mem_cgroup_put(memcg);
+ return;
+ }
+
+ pc = lookup_page_cgroup(page);
+ lock_page_cgroup(pc);
+ pc->mem_cgroup = memcg;
+ SetPageCgroupUsed(pc);
+ unlock_page_cgroup(pc);
+}
+void __mem_cgroup_free_kmem_page(struct page *page, int order)
+{
+ struct mem_cgroup *memcg;

```

```

+ size_t size;
+ struct page_cgroup *pc;
+
+ if (mem_cgroup_disabled())
+ return;
+
+ pc = lookup_page_cgroup(page);
+ lock_page_cgroup(pc);
+ memcg = pc->mem_cgroup;
+ pc->mem_cgroup = NULL;
+ if (!PageCgroupUsed(pc)) {
+ unlock_page_cgroup(pc);
+ return;
+ }
+ ClearPageCgroupUsed(pc);
+ unlock_page_cgroup(pc);
+
+ /*
+  * The classical disabled check won't work
+  * for uncharge, since it is possible that the user enabled
+  * kmem tracking, allocated, and then disabled.
+  *
+  * We trust if there is a memcg associated with the page,
+  * it is a valid allocation
+  */
+ if (!memcg)
+ return;
+
+ WARN_ON(mem_cgroup_is_root(memcg));
+ size = (1 << order) << PAGE_SHIFT;
+ memcg_uncharge_kmem(memcg, size);
+ mem_cgroup_put(memcg);
+}
+EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

#ifdef CONFIG_INET && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
@@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
__setup("swapaccount=", enable_swap_account);

#endif
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta)
+{
+ struct res_counter *fail_res;
+ struct mem_cgroup *_memcg;
+ int may_oom, ret;

```



```

+ bool nofail = false;
+
+ may_oom = (gfp & __GFP_WAIT) && (gfp & __GFP_FS) &&
+   !(gfp & __GFP_NORETRY);
+
+ ret = 0;
+
+ if (!memcg)
+   return ret;
+
+ _memcg = memcg;
+ ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
+   &_amp;_memcg, may_oom);
+
+ if (ret == -EINTR) {
+   nofail = true;
+   /*
+    * __mem_cgroup_try_charge() chose to bypass to root due
+    * to OOM kill or fatal signal.
+    * Since our only options are to either fail the
+    * allocation or charge it to this cgroup, do it as
+    * a temporary condition. But we can't fail. From a kmem/slab
+    * perspective, the cache has already been selected, by
+    * mem_cgroup_get_kmem_cache(), so it is too late to change our
+    * minds
+    */
+   res_counter_charge_nofail(&memcg->res, delta, &fail_res);
+   if (do_swap_account)
+     res_counter_charge_nofail(&memcg->memsw, delta,
+       &fail_res);
+   ret = 0;
+ } else if (ret == -ENOMEM)
+   return ret;
+
+ if (nofail)
+   res_counter_charge_nofail(&memcg->kmem, delta, &fail_res);
+ else
+   ret = res_counter_charge(&memcg->kmem, delta, &fail_res);
+
+ if (ret) {
+   res_counter_uncharge(&memcg->res, delta);
+   if (do_swap_account)
+     res_counter_uncharge(&memcg->memsw, delta);
+ }
+
+ return ret;
+}
+

```

```
+void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta)
+{
+ if (!memcg)
+ return;
+
+ res_counter_uncharge(&memcg->kmem, delta);
+ res_counter_uncharge(&memcg->res, delta);
+ if (do_swap_account)
+ res_counter_uncharge(&memcg->memsw, delta);
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
--
1.7.10.2
```

---

Subject: [PATCH 07/11] mm: Allocate kernel pages to the right memcg  
 Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

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()`.

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>  
 CC: Suleiman Souhlal <suleiman@google.com>

```
---
include/linux/gfp.h | 3 +++
mm/page_alloc.c | 27 ++++++
2 files changed, 30 insertions(+)
```

```
diff --git a/include/linux/gfp.h b/include/linux/gfp.h
index 8f4079f..4e27cd8 100644
--- a/include/linux/gfp.h
+++ b/include/linux/gfp.h
@@ -368,6 +368,9 @@ 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 4403009..cb8867e 100644
--- a/mm/page_alloc.c
+++ b/mm/page_alloc.c
@@ -2479,6 +2479,7 @@ __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;
+ void *handle = NULL;

    gfp_mask &= gfp_allowed_mask;

@@ -2490,6 +2491,13 @@ __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
    return NULL;

    /*
+ * Will only have any effect when __GFP_KMEMCG is set.
+ * This is verified in the (always inline) callee
+ */
+ if (!mem_cgroup_new_kmem_page(gfp_mask, &handle, order))
+ return NULL;
+
+ /*
    * Check the zones suitable for the gfp_mask contain at least one
    * valid zone. It's possible to have an empty zonelist as a result
    * of GFP_THISNODE and a memoryless node
@@ -2528,6 +2536,8 @@ out:
    if (unlikely(!put_mems_allowed(cpuset_mems_cookie) && !page))
        goto retry_cpuset;

+ mem_cgroup_commit_kmem_page(page, handle, order);
+
    return page;
}
EXPORT_SYMBOL(__alloc_pages_nodemask);
@@ -2580,6 +2590,23 @@ void free_pages(unsigned long addr, unsigned int order)

EXPORT_SYMBOL(free_pages);

+void __free_accounted_pages(struct page *page, unsigned int order)

```

```
+{
+ mem_cgroup_free_kmem_page(page, order);
+ __free_pages(page, order);
+}
+EXPORT_SYMBOL(__free_accounted_pages);
+
+void free_accounted_pages(unsigned long addr, unsigned int order)
+{
+ if (addr != 0) {
+ VM_BUG_ON(!virt_addr_valid((void *)addr));
+ mem_cgroup_free_kmem_page(virt_to_page((void *)addr), order);
+ __free_pages(virt_to_page((void *)addr), order);
+ }
+}
+EXPORT_SYMBOL(free_accounted_pages);
+
+static void *make_alloc_exact(unsigned long addr, unsigned order, size_t size)
+{
+ if (addr) {

```

Subject: [PATCH 08/11] memcg: disable kmem code when not in use.  
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:25 GMT  
[View Forum Message](#) <> [Reply to Message](#)

We can use jump labels to patch the code in or out when not used.

Because the assignment: `memcg->kmem_accounted = true` is done after the jump labels increment, we guarantee that the root memcg will always be selected until all call sites are patched (see `mem_cgroup_kmem_enabled`). This guarantees that no mischarges are applied.

Jump label decrement happens when the last reference count from the memcg dies. This will only happen when the caches are all dead.

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>  
CC: Suleiman Souhlal <suleiman@google.com>

```
include/linux/memcontrol.h | 4 +++-
mm/memcontrol.c           | 28 ++++++-----
2 files changed, 29 insertions(+), 3 deletions(-)
```

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
```

```
index 22479eb..4d69ff8 100644
```

```
--- a/include/linux/memcontrol.h
```

```
+++ b/include/linux/memcontrol.h
```

```
@ @ -22,6 +22,7 @ @
```

```
#include <linux/cgroup.h>
```

```
#include <linux/vm_event_item.h>
```

```
#include <linux/hardirq.h>
```

```
+#include <linux/jump_label.h>
```

```
struct mem_cgroup;
```

```
struct page_cgroup;
```

```
@ @ -411,7 +412,8 @ @ struct sock;
```

```
void sock_update_memcg(struct sock *sk);
```

```
void sock_release_memcg(struct sock *sk);
```

```
+#define mem_cgroup_kmem_on 1
```

```
+extern struct static_key mem_cgroup_kmem_enabled_key;
```

```
+#define mem_cgroup_kmem_on static_key_false(&mem_cgroup_kmem_enabled_key)
```

```
bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
```

```
void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
```

```
void __mem_cgroup_free_kmem_page(struct page *page, int order);
```

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
```

```
index 27b2b6f..fe5388e 100644
```

```
--- a/mm/memcontrol.c
```

```
+++ b/mm/memcontrol.c
```

```
@ @ -425,6 +425,10 @ @ static void mem_cgroup_put(struct mem_cgroup *memcg);
```

```
#include <net/sock.h>
```

```
#include <net/ip.h>
```

```
+struct static_key mem_cgroup_kmem_enabled_key;
```

```
+/+ so modules can inline the checks +/
```

```
+EXPORT_SYMBOL(mem_cgroup_kmem_enabled_key);
```

```
+
```

```
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);
```

```
@ @ -582,6 +586,16 @ @ void __mem_cgroup_free_kmem_page(struct page *page, int order)
    mem_cgroup_put(memcg);
```

```
}
```

```
EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
```

```
+
```

```
+static void disarm_kmem_keys(struct mem_cgroup *memcg)
```

```
+{
```



Subject: [PATCH 09/11] memcg: propagate kmem limiting information to children  
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:26 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

The current memcg slab cache management fails to present satisfactory hierarchical behavior in the following scenario:

-> /cgroups/memory/A/B/C

- \* kmem limit set at A
- \* A and B empty taskwise
- \* bash in C does find /

Because kmem\_accounted is a boolean that was not set for C, no accounting would be done. This is, however, not what we expect.

The basic idea, is that when a cgroup is limited, we walk the tree upwards (something Kame and I already thought about doing for other purposes), and make sure that we store the information about the parent being limited in kmem\_accounted (that is turned into a bitmap: two booleans would not be space efficient). The code for that is taken from sched/core.c. My reasons for not putting it into a common place is to dodge the type issues that would arise from a common implementation between memcg and the scheduler - but I think that it should ultimately happen, so if you want me to do it now, let me know.

We do the reverse operation when a formerly limited cgroup becomes unlimited.

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>  
CC: Suleiman Souhlal <suleiman@google.com>

---

```
mm/memcontrol.c | 86 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-----  
1 file changed, 71 insertions(+), 15 deletions(-)
```

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c  
index fe5388e..a6a440b 100644  
--- a/mm/memcontrol.c  
+++ b/mm/memcontrol.c  
@@ -287,7 +287,11 @@ struct mem_cgroup {  
    * Should the accounting and control be hierarchical, per subtree?  
    */  
    bool use_hierarchy;  
-    bool kmem_accounted;  
+    /*
```

```

+ * bit0: accounted by this cgroup
+ * bit1: accounted by a parent.
+ */
+ volatile unsigned long kmem_accounted;

    bool oom_lock;
    atomic_t under_oom;
@@ -340,6 +344,9 @@ struct mem_cgroup {
#endif
};

+#define KMEM_ACCOUNTED_THIS 0
+#define KMEM_ACCOUNTED_PARENT 1
+
+/* Stuffs for move charges at task migration. */
+
+ * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
@@ -589,7 +596,7 @@ EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);

static void disarm_kmem_keys(struct mem_cgroup *memcg)
{
- if (memcg->kmem_accounted)
+ if (test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted))
    static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
}
#else
@@ -4027,6 +4034,66 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
    len = scnprintf(str, sizeof(str), "%llu\n", (unsigned long long)val);
    return simple_read_from_buffer(buf, nbytes, ppos, str, len);
}
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static void mem_cgroup_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct mem_cgroup *iter;
+
+ mutex_lock(&set_limit_mutex);
+ if (!test_and_set_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted) &&
+ val != RESOURCE_MAX) {
+
+ /*
+ * 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
+ */

```



```

+ static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
+
+ if (!memcg->use_hierarchy)
+   goto out;
+
+ for_each_mem_cgroup_tree(iter, memcg) {
+   if (iter == memcg)
+     continue;
+   set_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
+ }
+
+ } else if (test_and_clear_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted)
+   && val == RESOURCE_MAX) {
+
+   if (!memcg->use_hierarchy)
+     goto out;
+
+   for_each_mem_cgroup_tree(iter, memcg) {
+     struct mem_cgroup *parent;
+     if (iter == memcg)
+       continue;
+     /*
+      * We should only have our parent bit cleared if none of
+      * our parents are accounted. The transversal order of
+      * our iter function forces us to always look at the
+      * parents.
+      */
+     parent = parent_mem_cgroup(iter);
+     while (parent && (parent != memcg)) {
+       if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
+         goto noclear;
+
+       parent = parent_mem_cgroup(parent);
+     }
+     clear_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
+   noclear:
+     continue;
+   }
+ }
+
+out:
+ mutex_unlock(&set_limit_mutex);
+}
+
+#endif
+
+/*
+ * The user of this function is...
+ * RES_LIMIT.
+ */
@@ -4064,19 +4131,8 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
    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
-  */
- mutex_lock(&set_limit_mutex);
- if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
-     static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
-     memcg->kmem_accounted = true;
- }
- mutex_unlock(&set_limit_mutex);
+ mem_cgroup_update_kmem_limit(memcg, val);
+ break;
    }
#endif
    else
--
1.7.10.2

```

---

Subject: [PATCH 10/11] memcg: allow a memcg with kmem charges to be destructed.

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:27 GMT

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

---

Because the ultimate goal of the kmem tracking in memcg is to track slab pages as well, we can't guarantee that we'll always be able to point a page to a particular process, and migrate the charges along with it - since in the common case, a page will contain data belonging to multiple processes.

Because of that, when we destroy a memcg, we only make sure the destruction will succeed by discounting the kmem charges from the user charges when we try to empty the cgroup.

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>  
 CC: Suleiman Souhlal <suleiman@google.com>

---

mm/memcontrol.c | 10 ++++++++-

1 file changed, 9 insertions(+), 1 deletion(-)

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index a6a440b..bb9b6fe 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -598,6 +598,11 @@ static void disarm_kmem_keys(struct mem_cgroup *memcg)
{
    if (test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted))
        static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
+ /*
+  * This check can't live in kmem destruction function,
+  * since the charges will outlive the cgroup
+  */
+ BUG_ON(res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0);
}
#else
static void disarm_kmem_keys(struct mem_cgroup *memcg)
@@ -3838,6 +3843,7 @@ static int mem_cgroup_force_empty(struct mem_cgroup *memcg, bool
free_all)
    int node, zid, shrink;
    int nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
    struct cgroup *cgrp = memcg->css.cgroup;
+ u64 usage;

    css_get(&memcg->css);

@@ -3877,8 +3883,10 @@ move_account:
    if (ret == -ENOMEM)
        goto try_to_free;
    cond_resched();
+ usage = res_counter_read_u64(&memcg->res, RES_USAGE) -
+ res_counter_read_u64(&memcg->kmem, RES_USAGE);
    /* "ret" should also be checked to ensure all lists are empty. */
- } while (res_counter_read_u64(&memcg->res, RES_USAGE) > 0 || ret);
+ } while (usage > 0 || ret);
out:
    css_put(&memcg->css);
    return ret;
--
1.7.10.2
```

---

Subject: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 14:15:28 GMT

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

---

Because those architectures will draw their stacks directly from the page allocator, rather than the slab cache, we can directly pass `__GFP_KMEMCG` flag, and issue the corresponding `free_pages`.

This code path is taken when the architecture doesn't define `CONFIG_ARCH_THREAD_INFO_ALLOCATOR` (only ia64 seems to), and has `THREAD_SIZE >= PAGE_SIZE`. Luckily, most - if not all - of the remaining architectures fall in this category.

This will guarantee that every stack page is accounted to the memcg the process currently lives on, and will have the allocations to fail if they go over limit.

For the time being, I am defining a new variant of `THREADINFO_GFP`, not to mess with the other path. Once the slab is also tracked by memcg, we can get rid of that flag.

Tested to successfully protect against `:(){ :|& };:`

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>  
CC: Suleiman Souhlal <suleiman@google.com>  
CC: Frederic Weisbecker <fweisbec@redhat.com>

---

```
include/linux/thread_info.h | 6 ++++++
kernel/fork.c                | 4 +---
2 files changed, 8 insertions(+), 2 deletions(-)
```

diff --git a/include/linux/thread\_info.h b/include/linux/thread\_info.h

index ccc1899..914ec07 100644

--- a/include/linux/thread\_info.h

+++ b/include/linux/thread\_info.h

```
@@ -61,6 +61,12 @@ extern long do_no_restart_syscall(struct restart_block *parm);
# define THREADINFO_GFP (GFP_KERNEL | __GFP_NOTRACK)
#endif
```

```
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
```

```
+# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP | __GFP_KMEMCG)
```

```
+#else
```

```
+# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP)
```

```
+#endif
```

```
+
```

```
/*
```

```
 * flag set/clear/test wrappers
```

```

* - pass TIF_xxxx constants to these functions
diff --git a/kernel/fork.c b/kernel/fork.c
index ab5211b..06b414f 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -142,7 +142,7 @@ void __weak arch_release_thread_info(struct thread_info *ti) { }
static struct thread_info *alloc_thread_info_node(struct task_struct *tsk,
int node)
{
- struct page *page = alloc_pages_node(node, THREADINFO_GFP,
+ struct page *page = alloc_pages_node(node, THREADINFO_GFP_ACCOUNTED,
THREAD_SIZE_ORDER);

return page ? page_address(page) : NULL;
@@ -151,7 +151,7 @@ static struct thread_info *alloc_thread_info_node(struct task_struct *tsk,
static inline void free_thread_info(struct thread_info *ti)
{
arch_release_thread_info(ti);
- free_pages((unsigned long)ti, THREAD_SIZE_ORDER);
+ free_accounted_pages((unsigned long)ti, THREAD_SIZE_ORDER);
}
# else
static struct kmem_cache *thread_info_cache;
--
1.7.10.2

```

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Frederic Weisbecker](#) on Mon, 25 Jun 2012 16:55:35 GMT

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

---

On 06/25/2012 04:15 PM, Glauber Costa wrote:

```

> Because those architectures will draw their stacks directly from
> the page allocator, rather than the slab cache, we can directly
> pass __GFP_KMEMCG flag, and issue the corresponding free_pages.
>
> This code path is taken when the architecture doesn't define
> CONFIG_ARCH_THREAD_INFO_ALLOCATOR (only ia64 seems to), and has
> THREAD_SIZE >= PAGE_SIZE. Luckily, most - if not all - of the
> remaining architectures fall in this category.
>
> This will guarantee that every stack page is accounted to the memcg
> the process currently lives on, and will have the allocations to fail
> if they go over limit.
>
> For the time being, I am defining a new variant of THREADINFO_GFP, not

```

> to mess with the other path. Once the slab is also tracked by memcg,  
> we can get rid of that flag.  
>  
> Tested to successfully protect against :(){ :|:& }::  
>  
> 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>  
> CC: Suleiman Souhlal <suleiman@google.com>

Acked-by: Frederic Weisbecker <fweisbec@redhat.com>

Thanks!

---

Subject: Re: [PATCH 01/11] memcg: Make it possible to use the stock for more than one page.

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 17:44:37 GMT

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

---

Hey, Glauber.

Just a couple nits.

On Mon, Jun 25, 2012 at 06:15:18PM +0400, Glauber Costa wrote:

> From: Suleiman Souhlal <ssouhlal@FreeBSD.org>

It would be nice to explain why this is being done. Just a simple statement like - "prepare for XXX" or "will be needed by XXX".

```
> /*
> - * Try to consume stocked charge on this cpu. If success, one page is consumed
> - * from local stock and true is returned. If the stock is 0 or charges from a
> - * cgroup which is not current target, returns false. This stock will be
> - * refilled.
> + * Try to consume stocked charge on this cpu. If success, nr_pages pages are
> + * consumed from local stock and true is returned. If the stock is 0 or
> + * charges from a cgroup which is not current target, returns false.
> + * This stock will be refilled.
```

I hope this were converted to proper /\*\* function comment with arguments and return value description.

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:06:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Again, nits.

On Mon, Jun 25, 2012 at 06:15:23PM +0400, Glauber Costa wrote:

```
> +#define mem_cgroup_kmem_on 1
> +bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
> +void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
> +void __mem_cgroup_free_kmem_page(struct page *page, int order);
> +#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)
```

Ugh... please do the following instead.

```
static inline bool is_kmem_tracked_alloc(gfp_t gfp)
{
    return gfp & __GFP_KMEMCG;
}
```

```
> #else
> static inline void sock_update_memcg(struct sock *sk)
> {
> @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
> static inline void sock_release_memcg(struct sock *sk)
> {
> }
> +
> +#define mem_cgroup_kmem_on 0
> +#define __mem_cgroup_new_kmem_page(a, b, c) false
> +#define __mem_cgroup_free_kmem_page(a,b )
> +#define __mem_cgroup_commit_kmem_page(a, b, c)
> +#define is_kmem_tracked_alloc (false)
```

I would prefer static inlines here too. It's a bit more code in the header but leads to less surprises (e.g. arg evals w/ side effects or compiler warning about unused vars) and makes it easier to avoid cosmetic errors.

Thanks.

--

tejun

---

---

Subject: Re: [PATCH 07/11] mm: Allocate kernel pages to the right memcg

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:07:47 GMT

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

---

On Mon, Jun 25, 2012 at 06:15:24PM +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()`.

Shouldn't we be documenting that in the code somewhere, preferably in  
the function comments?

--

tejun

---

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to  
children

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:29:07 GMT

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

---

Feeling like a nit pervert but..

On Mon, Jun 25, 2012 at 06:15:26PM +0400, Glauber Costa wrote:

```
> @@ -287,7 +287,11 @@ struct mem_cgroup {  
>  * Should the accounting and control be hierarchical, per subtree?  
>  */  
>  bool use_hierarchy;  
> - bool kmem_accounted;  
> + /*  
> +  * bit0: accounted by this cgroup  
> +  * bit1: accounted by a parent.  
> +  */  
> + volatile unsigned long kmem_accounted;
```

Is the volatile declaration really necessary? Why is it necessary?  
Why no comment explaining it?



```

> + #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + static void mem_cgroup_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
> + {
> + struct mem_cgroup *iter;
> +
> + mutex_lock(&set_limit_mutex);
> + if (!test_and_set_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted) &&
> + val != RESOURCE_MAX) {
> +
> + /*
> +  * 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
> +  */
> + static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
> +
> + if (!memcg->use_hierarchy)
> + goto out;
> +
> + for_each_mem_cgroup_tree(iter, memcg) {
> + if (iter == memcg)
> + continue;
> + set_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
> + }
> +
> + } else if (test_and_clear_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted)
> + && val == RESOURCE_MAX) {
> +
> + if (!memcg->use_hierarchy)
> + goto out;
> +
> + for_each_mem_cgroup_tree(iter, memcg) {
> + struct mem_cgroup *parent;

```

Blank line between decl and body please.

```

> + if (iter == memcg)
> + continue;
> + /*
> +  * We should only have our parent bit cleared if none of
> +  * our parents are accounted. The transversal order of

```

^ type

```

> +  * our iter function forces us to always look at the

```

> + \* parents.

Also, it's okay here but the text filling in comments and patch descriptions tend to be quite inconsistent. If you're on emacs, alt-q is your friend and I'm sure vim can do text filling pretty nicely too.

```
> + */
> + parent = parent_mem_cgroup(iter);
> + while (parent && (parent != memcg)) {
> +     if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
> +         goto noclear;
> +
> +     parent = parent_mem_cgroup(parent);
> + }
```

Better written in for (;)? Also, if we're breaking on parent == memcg, can we ever hit NULL parent in the above loop?

```
> + clear_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
> +noclear:
> + continue;
> + }
> + }
> +out:
> + mutex_unlock(&set_limit_mutex);
```

Can we please branch on val != RECURSE\_MAX first? I'm not even sure whether the above conditionals are correct. If the user updates an existing kmem limit, the first test\_and\_set\_bit() returns non-zero, so the code proceeds onto clearing KMEM\_ACCOUNTED\_THIS, which succeeds but val == RESOURCE\_MAX fails so it doesn't do anything. If the user changes it again, it will set ACCOUNTED\_THIS again. So, changing an existing kmem limit toggles KMEM\_ACCOUNTED\_THIS, which just seems wacky to me.

Thanks.

--

tejun

---

Subject: Re: [PATCH 10/11] memcg: allow a memcg with kmem charges to be destructed.

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:34:42 GMT

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

---

On Mon, Jun 25, 2012 at 06:15:27PM +0400, Glauber Costa wrote:

> Because the ultimate goal of the kmem tracking in memcg is to

```

> track slab pages as well, we can't guarantee that we'll always
> be able to point a page to a particular process, and migrate
> the charges along with it - since in the common case, a page
> will contain data belonging to multiple processes.
>
> Because of that, when we destroy a memcg, we only make sure
> the destruction will succeed by discounting the kmem charges
> from the user charges when we try to empty the cgroup.
>
> 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>
> CC: Suleiman Souhlal <suleiman@google.com>
> ---
> mm/memcontrol.c | 10 ++++++++--
> 1 file changed, 9 insertions(+), 1 deletion(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index a6a440b..bb9b6fe 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -598,6 +598,11 @@ static void disarm_kmem_keys(struct mem_cgroup *memcg)
> {
>     if (test_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted))
>         static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
> + /*
> +  * This check can't live in kmem destruction function,
> +  * since the charges will outlive the cgroup
> +  */
> + BUG_ON(res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0);

```

WARN\_ON() please. Misaccounted kernel usually is better than dead kernel.

Thanks.

--  
tejun

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs  
 Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:38:18 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:

> On 06/25/2012 04:15 PM, Glauber Costa wrote:

>

> > Because those architectures will draw their stacks directly from  
> > the page allocator, rather than the slab cache, we can directly  
> > pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.

> >

> > This code path is taken when the architecture doesn't define  
> > CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
> > THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
> > remaining architectures fall in this category.

> >

> > This will guarantee that every stack page is accounted to the memcg  
> > the process currently lives on, and will have the allocations to fail  
> > if they go over limit.

> >

> > For the time being, I am defining a new variant of THREADINFO\_GFP, not  
> > to mess with the other path. Once the slab is also tracked by memcg,  
> > we can get rid of that flag.

> >

> > Tested to successfully protect against :(){ :|:& };;

> >

> > 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>

> > CC: Suleiman Souhlal <suleiman@google.com>

>

>

> Acked-by: Frederic Weisbecker <fweisbec@redhat.com>

Frederic, does this (with proper slab accounting added later) achieve  
what you wanted with the task counter?

Thanks.

--

tejun

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Frederic Weisbecker](#) on Mon, 25 Jun 2012 20:57:51 GMT

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

---

On Mon, Jun 25, 2012 at 11:38:18AM -0700, Tejun Heo wrote:

> On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:  
> > On 06/25/2012 04:15 PM, Glauber Costa wrote:  
> >  
> > > Because those architectures will draw their stacks directly from  
> > > the page allocator, rather than the slab cache, we can directly  
> > > pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.  
> > >  
> > > This code path is taken when the architecture doesn't define  
> > > CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
> > > THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
> > > remaining architectures fall in this category.  
> > >  
> > > This will guarantee that every stack page is accounted to the memcg  
> > > the process currently lives on, and will have the allocations to fail  
> > > if they go over limit.  
> > >  
> > > For the time being, I am defining a new variant of THREADINFO\_GFP, not  
> > > to mess with the other path. Once the slab is also tracked by memcg,  
> > > we can get rid of that flag.  
> > >  
> > > Tested to successfully protect against :(){ :|:& };:  
> > >  
> > > 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>  
> > > CC: Suleiman Souhlal <suleiman@google.com>  
> >  
> >  
> > Acked-by: Frederic Weisbecker <fweisbec@redhat.com>  
>  
> Frederic, does this (with proper slab accounting added later) achieve  
> what you wanted with the task counter?

I think so yeah. Relying on general kernel memory accounting should do the trick for us. And if we need more finegrained limitation on kernel stack accounting we can still add it incrementally. But I believe global limitation can be enough.

Thanks.

---

Subject: Re: [PATCH 10/11] memcg: allow a memcg with kmem charges to be destructed.

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 22:25:46 GMT

On 06/25/2012 10:34 PM, Tejun Heo wrote:

> On Mon, Jun 25, 2012 at 06:15:27PM +0400, Glauber Costa wrote:

>> Because the ultimate goal of the kmem tracking in memcg is to  
>> track slab pages as well, we can't guarantee that we'll always  
>> be able to point a page to a particular process, and migrate  
>> the charges along with it - since in the common case, a page  
>> will contain data belonging to multiple processes.

>>

>> Because of that, when we destroy a memcg, we only make sure  
>> the destruction will succeed by discounting the kmem charges  
>> from the user charges when we try to empty the cgroup.

>>

>> 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>

>> CC: Suleiman Souhlal <suleiman@google.com>

>> ---

>> mm/memcontrol.c | 10 ++++++++-

>> 1 file changed, 9 insertions(+), 1 deletion(-)

>>

>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c

>> index a6a440b..bb9b6fe 100644

>> --- a/mm/memcontrol.c

>> +++ b/mm/memcontrol.c

>> @@ -598,6 +598,11 @@ static void disarm\_kmem\_keys(struct mem\_cgroup \*memcg)

>> {

>> if (test\_bit(KMEM\_ACCOUNTED\_THIS, &memcg->kmem\_accounted))

>> static\_key\_slow\_dec(&mem\_cgroup\_kmem\_enabled\_key);

>> + /\*

>> + \* This check can't live in kmem destruction function,

>> + \* since the charges will outlive the cgroup

>> + \*/

>> + BUG\_ON(res\_counter\_read\_u64(&memcg->kmem, RES\_USAGE) != 0);

>

> WARN\_ON() please. Misaccounted kernel usually is better than dead

> kernel.

>

You're absolutely right, will change.

---

Subject: Re: [PATCH 07/11] mm: Allocate kernel pages to the right memcg  
Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 22:27:36 GMT

On 06/25/2012 10:07 PM, Tejun Heo wrote:

> On Mon, Jun 25, 2012 at 06:15:24PM +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()`.

>

> Shouldn't we be documenting that in the code somewhere, preferably in

> the function comments?

>

I can certainly do that.

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 22:28:00 GMT

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

---

On 06/25/2012 10:06 PM, Tejun Heo wrote:

> Again, nits.

>

> On Mon, Jun 25, 2012 at 06:15:23PM +0400, Glauber Costa wrote:

>> `+#define mem_cgroup_kmem_on 1`

>> `+bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);`

>> `+void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);`

>> `+void __mem_cgroup_free_kmem_page(struct page *page, int order);`

>> `+#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)`

>

> Ugh... please do the following instead.

>

> `static inline bool is_kmem_tracked_alloc(gfp_t gfp)`

> {

> `return gfp & __GFP_KMEMCG;`

> }

>

>> `#else`

>> `static inline void sock_update_memcg(struct sock *sk)`

>> {

>> `@@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)`

>> `static inline void sock_release_memcg(struct sock *sk)`

>> {

```
>> }
>> +
>> + #define mem_cgroup_kmem_on 0
>> + #define __mem_cgroup_new_kmem_page(a, b, c) false
>> + #define __mem_cgroup_free_kmem_page(a, b)
>> + #define __mem_cgroup_commit_kmem_page(a, b, c)
>> + #define is_kmem_tracked_alloc (false)
>
> I would prefer static inlines here too. It's a bit more code in the
> header but leads to less surprises (e.g. arg evals w/ side effects or
> compiler warning about unused vars) and makes it easier to avoid
> cosmetic errors.
>
> Thanks.
>
```

Sure thing.

---

Subject: Re: [PATCH 01/11] memcg: Make it possible to use the stock for more than one page.

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 22:29:06 GMT

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

---

On 06/25/2012 09:44 PM, Tejun Heo wrote:

> Hey, Glauber.

>

> Just a couple nits.

>

> On Mon, Jun 25, 2012 at 06:15:18PM +0400, Glauber Costa wrote:

>> From: Suleiman Souhlal <[ssouhlal@FreeBSD.org](mailto:ssouhlal@FreeBSD.org)>

>

> It would be nice to explain why this is being done. Just a simple  
> statement like - "prepare for XXX" or "will be needed by XXX".

I picked this patch from Suleiman Souhlal, and tried to keep it as close as possible to his version.

But for the sake of documentation, I can do that, yes.

---

Subject: Re: [PATCH 01/11] memcg: Make it possible to use the stock for more than one page.

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 22:33:01 GMT

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

---

Hello,



On Mon, Jun 25, 2012 at 3:29 PM, Glauber Costa <glommer@parallels.com> wrote:

>> It would be nice to explain why this is being done. Just a simple  
>> statement like - "prepare for XXX" or "will be needed by XXX".

>

>

> I picked this patch from Suleiman Souhlal, and tried to keep it as close as  
> possible to his version.

>

> But for the sake of documentation, I can do that, yes.

Yeah, that would be great. Also, the patch does change behavior,  
right? Explaining / justifying that a bit would be nice too.

Thanks!

--

tejun

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to  
children

Posted by [Glauber Costa](#) on Mon, 25 Jun 2012 22:36:27 GMT

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

---

On 06/25/2012 10:29 PM, Tejun Heo wrote:

> Feeling like a nit pervert but..

>

> On Mon, Jun 25, 2012 at 06:15:26PM +0400, Glauber Costa wrote:

>> @@ -287,7 +287,11 @@ struct mem\_cgroup {

>> \* Should the accounting and control be hierarchical, per subtree?

>> \*/

>> bool use\_hierarchy;

>> - bool kmem\_accounted;

>> + /\*

>> + \* bit0: accounted by this cgroup

>> + \* bit1: accounted by a parent.

>> + \*/

>> + volatile unsigned long kmem\_accounted;

>

> Is the volatile declaration really necessary? Why is it necessary?

> Why no comment explaining it?

Seems to be required by set\_bit and friends. gcc will complain if it is  
not volatile (take a look at the bit function headers)

>> +

>> + for\_each\_mem\_cgroup\_tree(iter, memcg) {

```

>> + struct mem_cgroup *parent;
>
> Blank line between decl and body please.
ok.

>
>> + if (iter == memcg)
>> +     continue;
>> + /*
>> +  * We should only have our parent bit cleared if none of
>> +  * our parents are accounted. The transversal order of
>
>                                     ^ type
>
>> +  * our iter function forces us to always look at the
>> +  * parents.
>
> Also, it's okay here but the text filling in comments and patch
> descriptions tend to be quite inconsistent. If you're on emacs, alt-q
> is your friend and I'm sure vim can do text filling pretty nicely too.
>
>> + */
>> + parent = parent_mem_cgroup(iter);
>> + while (parent && (parent != memcg)) {
>> +     if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
>> +         goto noclear;
>> +
>> +     parent = parent_mem_cgroup(parent);
>> + }
>
> Better written in for (;;) ? Also, if we're breaking on parent ==
> memcg, can we ever hit NULL parent in the above loop?

I can simplify to test parent != memcg only, indeed it is not expected
to be NULL (but if it happens to be due to any kind of bug, we protect
against NULL-dereference, that is why I like to write this way)

>> + continue;
>> + }
>> + }
>> +out:
>> + mutex_unlock(&set_limit_mutex);
>
> Can we please branch on val != RECURSE_MAX first? I'm not even sure
> whether the above conditionals are correct. If the user updates an
> existing kmem limit, the first test_and_set_bit() returns non-zero, so
> the code proceeds onto clearing KMEM_ACCOUNTED_THIS, which succeeds
> but val == RESOURCE_MAX fails so it doesn't do anything. If the user

```

> changes it again, it will set ACCOUNTED\_THIS again. So, changing an  
> existing kmem limit toggles KMEM\_ACCOUNTED\_THIS, which just seems  
> wacky to me.  
>

I will take a look at that tomorrow as well.

---

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 22:49:42 GMT

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

---

Hello, Glauber.

On Tue, Jun 26, 2012 at 02:36:27AM +0400, Glauber Costa wrote:

> >Is the volatile declaration really necessary? Why is it necessary?  
> >Why no comment explaining it?  
>  
> Seems to be required by set\_bit and friends. gcc will complain if it  
> is not volatile (take a look at the bit function headers)

Hmmm? Are you sure gcc includes volatile in type check? There are a lot of bitops users in the kernel but most of them don't use volatile decl on the variable.

```
> >>+ */
> >>+ parent = parent_mem_cgroup(iter);
> >>+ while (parent && (parent != memcg)) {
> >>+   if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
> >>+     goto noclear;
> >>+
> >>+   parent = parent_mem_cgroup(parent);
> >>+ }
> >
> >Better written in for (;)? Also, if we're breaking on parent ==
> >memcg, can we ever hit NULL parent in the above loop?
>
> I can simplify to test parent != memcg only, indeed it is not
> expected to be NULL (but if it happens to be due to any kind of bug,
> we protect against NULL-dereference, that is why I like to write
> this way)
```

I personally don't really like that. It doesn't really add meaningful protection (if that happens the tree walking is already severely broken) while causes confusion to future readers of the code (when can parent be NULL?).

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [akpm](#) on Mon, 25 Jun 2012 23:17:20 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, 25 Jun 2012 18:15:23 +0400  
Glauber Costa <glommer@parallels.com> 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 jump labels, so we don't
> incur any overhead when no mem cgroups are being used.
>
>
> ...
>
> @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
> static inline void sock_release_memcg(struct sock *sk)
> {
> }
> +
> +#define mem_cgroup_kmem_on 0
> +#define __mem_cgroup_new_kmem_page(a, b, c) false
> +#define __mem_cgroup_free_kmem_page(a,b )
> +#define __mem_cgroup_commit_kmem_page(a, b, c)
```

I suggest that the naming consistently follow the model  
"mem\_cgroup\_kmem\_foo". So "mem\_cgroup\_kmem\_" becomes the well-known  
identifier for this subsystem.

Then, s/mem\_cgroup/memcg/g/ - show us some mercy here!

```
> +#define is_kmem_tracked_alloc (false)
```

memcg\_kmem\_tracked\_alloc, perhaps. But what does this actually do?

<looks>

eww, ick.

```
> +#define is_kmem_tracked_alloc(gfp & __GFP_KMEMCG)
```

What Tejun said. This:

```
/*  
 * Nice comment goes here  
 */  
static inline bool memcg_kmem_tracked_alloc(gfp_t gfp)  
{  
    return gfp & __GFP_KMEMCG;  
}
```

```
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */  
> +  
> +static __always_inline  
> +bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
```

memcg\_kmem\_new\_page().

```
> +{  
> + if (!mem_cgroup_kmem_on)  
> + return true;  
> + if (!is_kmem_tracked_alloc)  
> + return true;  
> + if (!current->mm)  
> + return true;  
> + if (in_interrupt())  
> + return true;  
> + if (gfp & __GFP_NOFAIL)  
> + return true;  
> + return __mem_cgroup_new_kmem_page(gfp, handle, order);  
> +}
```

Add documentation, please. The semantics of the return value are inscrutable.

```
> +static __always_inline  
> +void mem_cgroup_free_kmem_page(struct page *page, int order)  
> +{  
> + if (mem_cgroup_kmem_on)  
> + __mem_cgroup_free_kmem_page(page, order);  
> +}
```

memcg\_kmem\_free\_page().

```
> +static __always_inline  
> +void mem_cgroup_commit_kmem_page(struct page *page, struct mem_cgroup *handle,
```

```

> + int order)
> +{
> + if (mem_cgroup_kmem_on)
> + __mem_cgroup_commit_kmem_page(page, handle, order);
> +}

```

memcg\_kmem\_commit\_page().

```

> #endif /* _LINUX_MEMCONTROL_H */
>
>
> ...
>
> +static inline bool mem_cgroup_kmem_enabled(struct mem_cgroup *memcg)
> +{
> + return !mem_cgroup_disabled() && memcg &&
> + !mem_cgroup_is_root(memcg) && memcg->kmem_accounted;
> +}

```

Does this really need to handle a memcg==NULL?

```

> +bool __mem_cgroup_new_kmem_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 (!mem_cgroup_kmem_enabled(memcg))
> + goto out;
> +
> + mem_cgroup_get(memcg);
> +
> + size = (1 << order) << PAGE_SHIFT;

```

size = PAGE\_SIZE << order;

is simpler and more typical.

```

> + ret = memcg_charge_kmem(memcg, gfp, size) == 0;

```

Odd. memcg\_charge\_kmem() returns a nice errno, but this conversion just drops that information on the floor. If the

mem\_cgroup\_new\_kmem\_page() return value had been documented, I might have been able to understand the thinking here. But it wasn't, so I couldn't.

```
> + if (!ret) {
> +   mem_cgroup_put(memcg);
> +   goto out;
> + }
> +
> + *handle = memcg;
> +out:
> + rcu_read_unlock();
> + return ret;
> +}
> +EXPORT_SYMBOL(__mem_cgroup_new_kmem_page);
> +
> +void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order)
> +{
> +   struct page_cgroup *pc;
> +   struct mem_cgroup *memcg = handle;
> +   size_t size;
> +
> +   if (!memcg)
> +       return;
> +
> +   WARN_ON(mem_cgroup_is_root(memcg));
> +   /* The page allocation must have failed. Revert */
> +   if (!page) {
> +       size = (1 << order) << PAGE_SHIFT;
> +
> +       PAGE_SIZE << order
> +
> +       memcg_uncharge_kmem(memcg, size);
> +       mem_cgroup_put(memcg);
> +       return;
> +   }
```

This all looks a bit odd. After a failure you run a commit which undoes the speculative charging. I guess it makes sense. It's the sort of thing which can be expounded upon in the documentation which isn't there, sigh.

```
> + pc = lookup_page_cgroup(page);
> + lock_page_cgroup(pc);
> + pc->mem_cgroup = memcg;
> + SetPageCgroupUsed(pc);
> + unlock_page_cgroup(pc);
> +}
```

missed a newline there.

```
> +void __mem_cgroup_free_kmem_page(struct page *page, int order)
> +{
> + struct mem_cgroup *memcg;
> + size_t size;
> + struct page_cgroup *pc;
> +
> + if (mem_cgroup_disabled())
> + return;
> +
> + pc = lookup_page_cgroup(page);
> + lock_page_cgroup(pc);
> + memcg = pc->mem_cgroup;
> + pc->mem_cgroup = NULL;
> + if (!PageCgroupUsed(pc)) {
> + unlock_page_cgroup(pc);
> + return;
> + }
> + ClearPageCgroupUsed(pc);
> + unlock_page_cgroup(pc);
> +
> + /*
> + * The classical disabled check won't work
```

What is "The classical disabled check"? Be specific. Testing mem\_cgroup\_kmem\_on?

```
> + * for uncharge, since it is possible that the user enabled
> + * kmem tracking, allocated, and then disabled.
> + *
> + * We trust if there is a memcg associated with the page,
> + * it is a valid allocation
> + */
> + if (!memcg)
> + return;
> +
> + WARN_ON(mem_cgroup_is_root(memcg));
> + size = (1 << order) << PAGE_SHIFT;
```

PAGE\_SIZE << order

```
> + memcg_uncharge_kmem(memcg, size);
> + mem_cgroup_put(memcg);
> +}
> +EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
```



```

>
> #if defined(CONFIG_INET) && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
> @@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
> __setup("swapaccount=", enable_swap_account);
>
> #endif
> +
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM

```

gargh. CONFIG\_MEMCG\_KMEM, please!

```

> +int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta)
> +{
> + struct res_counter *fail_res;
> + struct mem_cgroup *_memcg;
> + int may_oom, ret;
> + bool nofail = false;
> +
> + may_oom = (gfp & __GFP_WAIT) && (gfp & __GFP_FS) &&
> +   !(gfp & __GFP_NORETRY);

```

may\_oom should have bool type.

```

> + ret = 0;
> +
> + if (!memcg)
> + return ret;
> +
> + _memcg = memcg;
> + ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
> +   &_memcg, may_oom);
> +
> + if (ret == -EINTR) {
> + nofail = true;
> + /*
> +  * __mem_cgroup_try_charge() chose to bypass to root due
> +  * to OOM kill or fatal signal.

```

Is "bypass" correct? Maybe "fall back"?

```

> + * Since our only options are to either fail the
> + * allocation or charge it to this cgroup, do it as
> + * a temporary condition. But we can't fail. From a kmem/slab
> + * perspective, the cache has already been selected, by
> + * mem_cgroup_get_kmem_cache(), so it is too late to change our
> + * minds
> + */
> + res_counter_charge_nofail(&memcg->res, delta, &fail_res);

```

```

> + if (do_swap_account)
> +   res_counter_charge_nofail(&memcg->memsw, delta,
> +     &fail_res);
> +   ret = 0;
> + } else if (ret == -ENOMEM)
> +   return ret;
> +
> + if (nofail)
> +   res_counter_charge_nofail(&memcg->kmem, delta, &fail_res);
> + else
> +   ret = res_counter_charge(&memcg->kmem, delta, &fail_res);
> +
> + if (ret) {
> +   res_counter_uncharge(&memcg->res, delta);
> +   if (do_swap_account)
> +     res_counter_uncharge(&memcg->memsw, delta);
> + }
> +
> + return ret;
> +}
>
> ...
>

```

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [akpm](#) on Mon, 25 Jun 2012 23:21:58 GMT

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

---

On Tue, 26 Jun 2012 02:36:27 +0400

Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

```

> On 06/25/2012 10:29 PM, Tejun Heo wrote:
> > Feeling like a nit pervert but..
> >
> > On Mon, Jun 25, 2012 at 06:15:26PM +0400, Glauber Costa wrote:
> >> @@ -287,7 +287,11 @@ struct mem_cgroup {
> >>   * Should the accounting and control be hierarchical, per subtree?
> >>   */
> >>   bool use_hierarchy;
> >> - bool kmem_accounted;
> >> + /*
> >> +  * bit0: accounted by this cgroup
> >> +  * bit1: accounted by a parent.
> >> +  */
> >> + volatile unsigned long kmem_accounted;
> >

```

> > Is the volatile declaration really necessary? Why is it necessary?  
> > Why no comment explaining it?  
>  
> Seems to be required by set\_bit and friends. gcc will complain if it is  
> not volatile (take a look at the bit function headers)

That would be a broken gcc. We run test\_bit()/set\_bit() and friends against plain old 'unsigned long' in thousands of places. There's nothing special about this one!

---

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [akpm](#) on Mon, 25 Jun 2012 23:23:52 GMT

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

---

On Mon, 25 Jun 2012 18:15:26 +0400

Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

```
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -287,7 +287,11 @@ struct mem_cgroup {
>  * Should the accounting and control be hierarchical, per subtree?
>  */
>  bool use_hierarchy;
> - bool kmem_accounted;
> + /*
> +  * bit0: accounted by this cgroup
> +  * bit1: accounted by a parent.
> +  */
> + volatile unsigned long kmem_accounted;
```

I suggest

```
unsigned long kmem_accounted; /* See KMEM_ACCOUNTED_*, below */
```

```
>  bool oom_lock;
>  atomic_t under_oom;
> @@ -340,6 +344,9 @@ struct mem_cgroup {
> #endif
> };
>
> +#define KMEM_ACCOUNTED_THIS 0
> +#define KMEM_ACCOUNTED_PARENT 1
```

And then document the fields here.

---

---

Subject: Re: [PATCH 00/11] kmem controller for memcg: stripped down version

Posted by [akpm](#) on Mon, 25 Jun 2012 23:27:45 GMT

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

---

On Mon, 25 Jun 2012 18:15:17 +0400

Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

> What I am proposing with this series is a stripped down version of the  
> kmem controller for memcg that would allow us to merge significant parts  
> of the infrastructure, while leaving out, for now, the polemic bits about  
> the slab while it is being reworked by Cristoph.  
>  
> My reasoning for that is that after the last change to introduce a gfp  
> flag to mark kernel allocations, it became clear to me that tracking other  
> resources like the stack would then follow extremely naturally. I figured  
> that at some point we'd have to solve the issue pointed by David, and avoid  
> testing the Slab flag in the page allocator, since it would soon be made  
> more generic. I do that by having the callers to explicit mark it.  
>  
> So to demonstrate how it would work, I am introducing a stack tracker here,  
> that is already a functionality per-se: it successfully stops fork bombs to  
> happen. (Sorry for doing all your work, Frederic =p ). Note that after all  
> memcg infrastructure is deployed, it becomes very easy to track anything.  
> The last patch of this series is extremely simple.  
>  
> The infrastructure is exactly the same we had in memcg, but stripped down  
> of the slab parts. And because what we have after those patches is a feature  
> per-se, I think it could be considered for merging.

hm. None of this new code makes the kernel smaller, faster, easier to understand or more fun to read!

Presumably we're getting some benefit for all the downside. When the time is appropriate, please do put some time into explaining that benefit, so that others can agree that it is a worthwhile tradeoff.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.

Posted by [Suleiman Souhlal](#) on Mon, 25 Jun 2012 23:33:29 GMT

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

---

On Mon, Jun 25, 2012 at 7:15 AM, Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

> From: Suleiman Souhlal <[ssouhlal@FreeBSD.org](mailto:ssouhlal@FreeBSD.org)>  
>  
> mem\_cgroup\_do\_charge() was written before slab accounting, and expects  
> three cases: being called for 1 page, being called for a stock of 32 pages,  
> or being called for a hugepage. If we call for 2 or 3 pages (and several  
> slabs used in process creation are such, at least with the debug options I

> had), it assumed it's being called for stock and just retried without reclaiming.  
>  
> Fix that by passing down a minsize argument in addition to the csize.  
>  
> And what to do about that (csize == PAGE\_SIZE && ret) retry? If it's  
> needed at all (and presumably is since it's there, perhaps to handle  
> races), then it should be extended to more than PAGE\_SIZE, yet how far?  
> And should there be a retry count limit, of what? For now retry up to  
> COSTLY\_ORDER (as page\_alloc.c does), stay safe with a cond\_resched(),  
> and make sure not to do it if \_\_GFP\_NORETRY.

The commit description mentions COSTLY\_ORDER, but it's not actually used in the patch.

-- Suleiman

---

---

Subject: Re: [PATCH 01/11] memcg: Make it possible to use the stock for more than one page.

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 04:01:54 GMT

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

---

On Mon, 25 Jun 2012, Glauber Costa wrote:

> From: Suleiman Souhlal <ssouhlal@FreeBSD.org>  
>  
> Signed-off-by: Suleiman Souhlal <suleiman@google.com>  
> Signed-off-by: Glauber Costa <glommer@parallels.com>  
> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Acked-by: David Rientjes <rientjes@google.com>

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 04:09:11 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 9304db2..8e601e8 100644  
> --- a/mm/memcontrol.c  
> +++ b/mm/memcontrol.c  
> @@ -2158,8 +2158,16 @@ enum {  
> CHARGE\_OOM\_DIE, /\* the current is killed because of OOM \*/  
> };  
>

```
> +/*
> + * We need a number that is small enough to be likely to have been
> + * reclaimed even under pressure, but not too big to trigger unnecessary
```

Whitespace.

```
> + * retries
> + */
> + #define NR_PAGES_TO_RETRY 2
> +
```

Should be  $1 \ll \text{PAGE\_ALLOC\_COSTLY\_ORDER}$ ? Where does this number come from?  
The changelog doesn't specify.

```
> static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t gfp_mask,
> - unsigned int nr_pages, bool oom_check)
> + unsigned int nr_pages, unsigned int min_pages,
> + bool oom_check)
> {
> unsigned long csize = nr_pages * PAGE_SIZE;
> struct mem_cgroup *mem_over_limit;
> @@ -2182,18 +2190,18 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
gfp_t gfp_mask,
> } else
> mem_over_limit = mem_cgroup_from_res_counter(fail_res, res);
> /*
> - * nr_pages can be either a huge page (HPAGE_PMD_NR), a batch
> - * of regular pages (CHARGE_BATCH), or a single regular page (1).
> - *
> - * Never reclaim on behalf of optional batching, retry with a
> - * single page instead.
> - */
> - if (nr_pages == CHARGE_BATCH)
> + if (nr_pages > min_pages)
> return CHARGE_RETRY;
>
> if (!(gfp_mask & __GFP_WAIT))
> return CHARGE_WOULDBLOCK;
>
> + if (gfp_mask & __GFP_NORETRY)
> + return CHARGE_NOMEM;
> +
> ret = mem_cgroup_reclaim(mem_over_limit, gfp_mask, flags);
> if (mem_cgroup_margin(mem_over_limit) >= nr_pages)
> return CHARGE_RETRY;
> @@ -2206,7 +2214,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
gfp_t gfp_mask,
> - * unlikely to succeed so close to the limit, and we fall back
```

```
> * to regular pages anyway in case of failure.
> */
> - if (nr_pages == 1 && ret)
> + if (nr_pages <= NR_PAGES_TO_RETRY && ret)
>   return CHARGE_RETRY;
>
> /*
> @@ -2341,7 +2349,8 @@ again:
>   nr_oom_retries = MEM_CGROUP_RECLAIM_RETRIES;
> }
>
> - ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, oom_check);
> + ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, nr_pages,
> +   oom_check);
>   switch (ret) {
>   case CHARGE_OK:
>     break;
```

---

---

Subject: Re: [PATCH 03/11] memcg: change defines to an enum  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 04:11:55 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 8e601e8..9352d40 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -387,9 +387,12 @@ enum charge_type {
> };
>
> /* for encoding cft->private value on file */
> #define _MEM (0)
> #define _MEMSWAP (1)
> #define _OOM_TYPE (2)
> +enum res_type {
> + _MEM,
> + _MEMSWAP,
> + _OOM_TYPE,
> +};
> +
> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
> #define MEMFILE_TYPE(val) ((val) >> 16 & 0xffff)
> #define MEMFILE_ATTR(val) ((val) & 0xffff)
```

Shouldn't everything that does MEMFILE\_TYPE() now be using type enum res\_type rather than int?

---

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);

```

---

Subject: Re: [PATCH 05/11] Add a \_\_GFP\_KMEMCG flag  
 Posted by [David Rientjes](#) on Tue, 26 Jun 2012 04:25:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, 25 Jun 2012, Glauber Costa wrote:

```

> This flag is used to indicate to the callees that this allocation will be
> serviced to the kernel. It is not supposed to be passed by the callers
> of kmem_cache_alloc, but rather by the cache core itself.

```

>

Not sure what "serviced to the kernel" means, does this mean that the memory will not be accounted for to the root memcg?

```
> 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>
> CC: Suleiman Souhlal <suleiman@google.com>
> ---
> include/linux/gfp.h | 8 ++++++-
> 1 file changed, 7 insertions(+), 1 deletion(-)
>
> diff --git a/include/linux/gfp.h b/include/linux/gfp.h
> index 1e49be4..8f4079f 100644
> --- a/include/linux/gfp.h
> +++ b/include/linux/gfp.h
> @@ -37,6 +37,9 @@ struct vm_area_struct;
> #define __GFP_NO_KSWAPD 0x400000u
> #define __GFP_OTHER_NODE 0x800000u
> #define __GFP_WRITE 0x1000000u
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +#define __GFP_KMEMCG 0x2000000u
> +#endif
>
> /*
>  * GFP bitmasks..
> @@ -88,13 +91,16 @@ struct vm_area_struct;
> #define __GFP_OTHER_NODE ((__force gfp_t) __GFP_OTHER_NODE) /* On behalf of other
node */
> #define __GFP_WRITE ((__force gfp_t) __GFP_WRITE) /* Allocator intends to dirty page */
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +#define __GFP_KMEMCG ((__force gfp_t) __GFP_KMEMCG) /* Allocation comes from a
memcg-accounted resource */
> +#endif
```

Needs a space.

```
> /*
>  * This may seem redundant, but it's a way of annotating false positives vs.
>  * allocations that simply cannot be supported (e.g. page tables).
>  */
> #define __GFP_NOTRACK_FALSE_POSITIVE (__GFP_NOTRACK)
>
> -#define __GFP_BITS_SHIFT 25 /* Room for N __GFP_FOO bits */
```

```
> +#define __GFP_BITS_SHIFT 26 /* Room for N __GFP_FOO bits */
> #define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))
>
> /* This equals 0, but use constants in case they ever change */
```

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 04:57:54 GMT

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

---

On Mon, 25 Jun 2012, Glauber Costa wrote:

```
> diff --git a/include/linux/thread_info.h b/include/linux/thread_info.h
> index ccc1899..914ec07 100644
> --- a/include/linux/thread_info.h
> +++ b/include/linux/thread_info.h
> @@ -61,6 +61,12 @@ extern long do_no_restart_syscall(struct restart_block *parm);
> # define THREADINFO_GFP (GFP_KERNEL | __GFP_NOTRACK)
> #endif
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP | __GFP_KMEMCG)
> +#else
> +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP)
> +#endif
> +
```

This type of requirement is going to become nasty very quickly if nobody can use \_\_GFP\_KMEMCG without testing for CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM. Perhaps define \_\_GFP\_KMEMCG to be 0x0 if it's not enabled, similar to how kmemcheck does?

---

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 05:23:59 GMT

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

---

On Mon, 25 Jun 2012, Andrew Morton wrote:

```
> > > */
> > > bool use_hierarchy;
> > > - bool kmem_accounted;
> > > + /*
> > > + * bit0: accounted by this cgroup
> > > + * bit1: accounted by a parent.
```

```

> > > + */
> > > + volatile unsigned long kmem_accounted;
> > >
> > > Is the volatile declaration really necessary? Why is it necessary?
> > > Why no comment explaining it?
> > >
> > > Seems to be required by set_bit and friends. gcc will complain if it is
> > > not volatile (take a look at the bit function headers)
> > >
> > > That would be a broken gcc. We run test_bit()/set_bit() and friends
> > > against plain old 'unsigned long' in thousands of places. There's
> > > nothing special about this one!
> > >
> > >

```

No version of gcc would complain about this, even with 4.6 and later with -fstrict-volatile-bitfields, it's a qualifier that determines whether or not the access to memory is the exact size of the bitfield and aligned to its natural boundary. If the type isn't qualified as such then it's simply going to compile to access the native word size of the architecture. No special consideration is needed for a member of struct mem\_cgroup.

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 05:24:44 GMT

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

---

On Mon, 25 Jun 2012, Andrew Morton wrote:

```

> > --- a/mm/memcontrol.c
> > +++ b/mm/memcontrol.c
> > @@ -287,7 +287,11 @@ struct mem_cgroup {
> >  * Should the accounting and control be hierarchical, per subtree?
> >  */
> >  bool use_hierarchy;
> > - bool kmem_accounted;
> > + /*
> > +  * bit0: accounted by this cgroup
> > +  * bit1: accounted by a parent.
> > +  */
> > + volatile unsigned long kmem_accounted;
> > +
> > I suggest
> >
> > unsigned long kmem_accounted; /* See KMEM_ACCOUNTED_*, below */
> >
> > bool oom_lock;

```

```
> > atomic_t under_oom;
> > @@ -340,6 +344,9 @@ struct mem_cgroup {
> > #endif
> > };
> >
> > +#define KMEM_ACCOUNTED_THIS 0
> > +#define KMEM_ACCOUNTED_PARENT 1
>
> And then document the fields here.
>
```

In hex, please?

---

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [akpm](#) on Tue, 26 Jun 2012 05:31:21 GMT

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

---

On Mon, 25 Jun 2012 22:24:44 -0700 (PDT) David Rientjes <[rientjes@google.com](mailto:rientjes@google.com)> wrote:

```
> > > +#define KMEM_ACCOUNTED_THIS 0
> > > +#define KMEM_ACCOUNTED_PARENT 1
> >
> > And then document the fields here.
> >
>
> In hex, please?
```

Well, they're bit numbers, not masks. Decimal 0-31 is OK, or an enum.

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [KAMEZAWA Hiroyuki](#) on Tue, 26 Jun 2012 05:35:34 GMT

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

---

(2012/06/26 13:57), David Rientjes wrote:

> On Mon, 25 Jun 2012, Glauber Costa wrote:

```
>
>> diff --git a/include/linux/thread_info.h b/include/linux/thread_info.h
>> index ccc1899..914ec07 100644
>> --- a/include/linux/thread_info.h
>> +++ b/include/linux/thread_info.h
>> @@ -61,6 +61,12 @@ extern long do_no_restart_syscall(struct restart_block *parm);
>> # define THREADINFO_GFP (GFP_KERNEL | __GFP_NOTRACK)
>> #endif
```

```
>>
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP | __GFP_KMEMCG)
>> +#else
>> +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP)
>> +#endif
>> +
>
> This type of requirement is going to become nasty very quickly if nobody
> can use __GFP_KMEMCG without testing for CONFIG_CGROUP_MEM_RES_CTLR_KMEM.
> Perhaps define __GFP_KMEMCG to be 0x0 if it's not enabled, similar to how
> kmemcheck does?
```

I agree.

Thanks,  
-Kame

---

Subject: Re: [PATCH 08/11] memcg: disable kmem code when not in use.  
Posted by [KAMEZAWA Hiroyuki](#) on Tue, 26 Jun 2012 05:51:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

(2012/06/25 23:15), Glauber Costa wrote:

```
> We can use jump labels to patch the code in or out
> when not used.
>
> Because the assignment: memcg->kmem_accounted = true
> is done after the jump labels increment, we guarantee
> that the root memcg will always be selected until
> all call sites are patched (see mem_cgroup_kmem_enabled).
> This guarantees that no mischarges are applied.
>
> Jump label decrement happens when the last reference
> count from the memcg dies. This will only happen when
> the caches are all dead.
>
> 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>
> CC: Suleiman Souhlal <suleiman@google.com>
```

Acked-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---



Subject: Re: [PATCH 10/11] memcg: allow a memcg with kmem charges to be destructed.

Posted by [KAMEZAWA Hiroyuki](#) on Tue, 26 Jun 2012 05:59:56 GMT

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

---

(2012/06/25 23:15), Glauber Costa wrote:

> Because the ultimate goal of the kmem tracking in memcg is to  
> track slab pages as well, we can't guarantee that we'll always  
> be able to point a page to a particular process, and migrate  
> the charges along with it - since in the common case, a page  
> will contain data belonging to multiple processes.

>  
> Because of that, when we destroy a memcg, we only make sure  
> the destruction will succeed by discounting the kmem charges  
> from the user charges when we try to empty the cgroup.

>  
> 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>  
> CC: Suleiman Souhlal <suleiman@google.com>

> ---

> mm/memcontrol.c | 10 ++++++++-

> 1 file changed, 9 insertions(+), 1 deletion(-)

>

> diff --git a/mm/memcontrol.c b/mm/memcontrol.c

> index a6a440b..bb9b6fe 100644

> --- a/mm/memcontrol.c

> +++ b/mm/memcontrol.c

> @@ -598,6 +598,11 @@ static void disarm\_kmem\_keys(struct mem\_cgroup \*memcg)

> {

> if (test\_bit(KMEM\_ACCOUNTED\_THIS, &memcg->kmem\_accounted))

> static\_key\_slow\_dec(&mem\_cgroup\_kmem\_enabled\_key);

> + /\*

> + \* This check can't live in kmem destruction function,

> + \* since the charges will outlive the cgroup

> + \*/

> + BUG\_ON(res\_counter\_read\_u64(&memcg->kmem, RES\_USAGE) != 0);

> }

> #else

> static void disarm\_kmem\_keys(struct mem\_cgroup \*memcg)

> @@ -3838,6 +3843,7 @@ static int mem\_cgroup\_force\_empty(struct mem\_cgroup \*memcg,  
> bool free\_all)

> int node, zid, shrink;

> int nr\_retries = MEM\_CGROUP\_RECLAIM\_RETRIES;

> struct cgroup \*cgrp = memcg->css.cgroup;

> + u64 usage;

```

>
>  css_get(&memcg->css);
>
> @@ -3877,8 +3883,10 @@ move_account:
>  if (ret == -ENOMEM)
>    goto try_to_free;
>  cond_resched();
> + usage = res_counter_read_u64(&memcg->res, RES_USAGE) -
> + res_counter_read_u64(&memcg->kmem, RES_USAGE);
>  /* "ret" should also be checked to ensure all lists are empty. */
> - } while (res_counter_read_u64(&memcg->res, RES_USAGE) > 0 || ret);
> + } while (usage > 0 || ret);
>  out:
>  css_put(&memcg->css);
>  return ret;
>

```

Hm....maybe work enough. Could you add more comments on the code ?

Acked-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---



---

Subject: Re: [PATCH 05/11] Add a \_\_GFP\_KMEMCG flag  
 Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:08:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 08:25 AM, David Rientjes wrote:

> On Mon, 25 Jun 2012, Glauber Costa wrote:

>

>> >This flag is used to indicate to the callees that this allocation will be  
 >> >serviced to the kernel. It is not supposed to be passed by the callers  
 >> >of kmem\_cache\_alloc, but rather by the cache core itself.

>> >

> Not sure what "serviced to the kernel" means, does this mean that the  
 > memory will not be accounted for to the root memcg?

>

In this context, it means that is a kernel allocation, not a userspace  
 one (but in process context, of course), \*and\* it is to be accounted a  
 specific memcg.

---



---

Subject: Re: [PATCH 04/11] kmem slab accounting basic infrastructure  
 Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:09:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 08:22 AM, David Rientjes wrote:

> 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?
```

Yes, it can.

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
 Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:12:22 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

```
>
>> + * retries
>> + */
>> + #define NR_PAGES_TO_RETRY 2
>> +
>
> Should be 1 << PAGE_ALLOC_COSTLY_ORDER? Where does this number come from?
> The changelog doesn't specify.
```

Hocko complained about that, and I changed. Where the number comes from, is stated in the comments: it is a number small enough to have high changes of had been freed by the previous reclaim, and yet around the number of pages of a kernel allocation.

Of course there are allocations for nr\_pages > 2. But 2 will already service the stack most of the time, and most of the slab caches.

```
>> static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t gfp_mask,
>> - unsigned int nr_pages, bool oom_check)
>> + unsigned int nr_pages, unsigned int min_pages,
>> + bool oom_check)
>> {
>> unsigned long csize = nr_pages * PAGE_SIZE;
>> struct mem_cgroup *mem_over_limit;
>> @@ -2182,18 +2190,18 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
>> gfp_t gfp_mask,
>> } else
>> mem_over_limit = mem_cgroup_from_res_counter(fail_res, res);
>> /*
```

```

>> - * nr_pages can be either a huge page (HPAGE_PMD_NR), a batch
>> - * of regular pages (CHARGE_BATCH), or a single regular page (1).
>> - *
>>   * Never reclaim on behalf of optional batching, retry with a
>>   * single page instead.
>>   */
>> - if (nr_pages == CHARGE_BATCH)
>> + if (nr_pages > min_pages)
>>   return CHARGE_RETRY;
>>
>>   if (!(gfp_mask & __GFP_WAIT))
>>   return CHARGE_WOULDBLOCK;
>>
>> + if (gfp_mask & __GFP_NORETRY)
>> + return CHARGE_NOMEM;
>> +
>>   ret = mem_cgroup_reclaim(mem_over_limit, gfp_mask, flags);
>>   if (mem_cgroup_margin(mem_over_limit) >= nr_pages)
>>   return CHARGE_RETRY;
>> @@ -2206,7 +2214,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
gfp_t gfp_mask,
>>   * unlikely to succeed so close to the limit, and we fall back
>>   * to regular pages anyway in case of failure.
>>   */
>> - if (nr_pages == 1 && ret)
>> + if (nr_pages <= NR_PAGES_TO_RETRY && ret)
>>   return CHARGE_RETRY;
>>
>>   /*
>> @@ -2341,7 +2349,8 @@ again:
>>   nr_oom_retries = MEM_CGROUP_RECLAIM_RETRIES;
>>   }
>>
>> - ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, oom_check);
>> + ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, nr_pages,
>> +   oom_check);
>>   switch (ret) {
>>   case CHARGE_OK:
>>     break;

```

---

Subject: Re: [PATCH 00/11] kmem controller for memcg: stripped down version  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:17:49 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 03:27 AM, Andrew Morton wrote:  
> On Mon, 25 Jun 2012 18:15:17 +0400  
> Glauber Costa <glommer@parallels.com> wrote:

>  
>> What I am proposing with this series is a stripped down version of the  
>> kmem controller for memcg that would allow us to merge significant parts  
>> of the infrastructure, while leaving out, for now, the polemic bits about  
>> the slab while it is being reworked by Cristoph.  
>>  
>> Me reasoning for that is that after the last change to introduce a gfp  
>> flag to mark kernel allocations, it became clear to me that tracking other  
>> resources like the stack would then follow extremely naturally. I figured  
>> that at some point we'd have to solve the issue pointed by David, and avoid  
>> testing the Slab flag in the page allocator, since it would soon be made  
>> more generic. I do that by having the callers to explicit mark it.  
>>  
>> So to demonstrate how it would work, I am introducing a stack tracker here,  
>> that is already a functionality per-se: it successfully stops fork bombs to  
>> happen. (Sorry for doing all your work, Frederic =p ). Note that after all  
>> memcg infrastructure is deployed, it becomes very easy to track anything.  
>> The last patch of this series is extremely simple.  
>>  
>> The infrastructure is exactly the same we had in memcg, but stripped down  
>> of the slab parts. And because what we have after those patches is a feature  
>> per-se, I think it could be considered for merging.  
>  
> hm. None of this new code makes the kernel smaller, faster, easier to  
> understand or more fun to read!  
Not sure if this is a general comment - in case I agree - or if targeted  
to my statement that this is "stripped down". If so, it is of course  
smaller relative to my previous slab accounting patches.

The infrastructure is largely common, but I realized that a future user,  
tracking the stack, would be a lot simpler and could be done first.

> Presumably we're getting some benefit for all the downside. When the  
> time is appropriate, please do put some time into explaining that  
> benefit, so that others can agree that it is a worthwhile tradeoff.  
>

Well, for one thing, we stop fork bombs for processes inside cgroups.  
I think the justification for that was already given when you asked  
people about reasoning for merging Frederic's process tracking cgroup.

Just that wasn't merged because people were largely unhappy with the  
form it took. I can't speak for everybody here, but AFAIK, tracking the  
stack through the memory it used, therefore using my proposed kmem  
controller, was an idea that good quite a bit of traction with the  
memcg/memory people. So here you have something that people already  
asked a lot for, in a shape and interface that seem to be acceptable.

---

---

Subject: Re: [PATCH 10/11] memcg: allow a memcg with kmem charges to be destructed.

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:21:22 GMT

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

---

On 06/26/2012 09:59 AM, Kamezawa Hiroyuki wrote:

> (2012/06/25 23:15), Glauber Costa wrote:

>> Because the ultimate goal of the kmem tracking in memcg is to  
>> track slab pages as well, we can't guarantee that we'll always  
>> be able to point a page to a particular process, and migrate  
>> the charges along with it - since in the common case, a page  
>> will contain data belonging to multiple processes.

>>

>> Because of that, when we destroy a memcg, we only make sure  
>> the destruction will succeed by discounting the kmem charges  
>> from the user charges when we try to empty the cgroup.

>>

>> 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>

>> CC: Suleiman Souhlal <suleiman@google.com>

>> ---

>> mm/memcontrol.c | 10 +++++++--

>> 1 file changed, 9 insertions(+), 1 deletion(-)

>>

>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c

>> index a6a440b..bb9b6fe 100644

>> --- a/mm/memcontrol.c

>> +++ b/mm/memcontrol.c

>> @@ -598,6 +598,11 @@ static void disarm\_kmem\_keys(struct mem\_cgroup \*memcg)

>> {

>> if (test\_bit(KMEM\_ACCOUNTED\_THIS, &memcg->kmem\_accounted))

>> static\_key\_slow\_dec(&mem\_cgroup\_kmem\_enabled\_key);

>> + /\*

>> + \* This check can't live in kmem destruction function,

>> + \* since the charges will outlive the cgroup

>> + \*/

>> + BUG\_ON(res\_counter\_read\_u64(&memcg->kmem, RES\_USAGE) != 0);

>> }

>> #else

>> static void disarm\_kmem\_keys(struct mem\_cgroup \*memcg)

>> @@ -3838,6 +3843,7 @@ static int mem\_cgroup\_force\_empty(struct mem\_cgroup \*memcg,  
>> bool free\_all)

>> int node, zid, shrink;

>> int nr\_retries = MEM\_CGROUP\_RECLAIM\_RETRIES;

>> struct cgroup \*cgrp = memcg->css.cgroup;

```

>> + u64 usage;
>>
>>  css_get(&memcg->css);
>>
>> @@ -3877,8 +3883,10 @@ move_account:
>>  if (ret == -ENOMEM)
>>      goto try_to_free;
>>  cond_resched();
>> + usage = res_counter_read_u64(&memcg->res, RES_USAGE) -
>> + res_counter_read_u64(&memcg->kmem, RES_USAGE);
>>  /* "ret" should also be checked to ensure all lists are empty. */
>> - } while (res_counter_read_u64(&memcg->res, RES_USAGE) > 0 || ret);
>> + } while (usage > 0 || ret);
>>  out:
>>  css_put(&memcg->css);
>>  return ret;
>>
> Hm....maybe work enough. Could you add more comments on the code ?
>
> Acked-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>

```

I always can.

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:23:04 GMT

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

---

On 06/26/2012 08:57 AM, David Rientjes wrote:

> On Mon, 25 Jun 2012, Glauber Costa wrote:

```

>
>> diff --git a/include/linux/thread_info.h b/include/linux/thread_info.h
>> index ccc1899..914ec07 100644
>> --- a/include/linux/thread_info.h
>> +++ b/include/linux/thread_info.h
>> @@ -61,6 +61,12 @@ extern long do_no_restart_syscall(struct restart_block *parm);
>>  # define THREADINFO_GFP (GFP_KERNEL | __GFP_NOTRACK)
>>  #endif
>>
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>>  # define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP | __GFP_KMEMCG)
>>  #else
>>  # define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP)
>>  #endif
>> +
>

```



> This type of requirement is going to become nasty very quickly if nobody  
> can use \_\_GFP\_KMEMCG without testing for CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM.  
> Perhaps define \_\_GFP\_KMEMCG to be 0x0 if it's not enabled, similar to how  
> kmemcheck does?

>  
That is what I've done in my first version of this patch. At that time,  
Christoph wanted it to be this way so we would make sure it would never  
be used with #CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM defined. A value of zero  
will generate no errors. Undefined value will.

Now, if you ask me, I personally prefer following what kmemcheck does  
here...

---

Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to  
children

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 07:23:43 GMT

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

---

On 06/26/2012 09:31 AM, Andrew Morton wrote:

> On Mon, 25 Jun 2012 22:24:44 -0700 (PDT) David Rientjes <[rientjes@google.com](mailto:rientjes@google.com)> wrote:

>  
>>>> +#define KMEM\_ACCOUNTED\_THIS 0  
>>>> +#define KMEM\_ACCOUNTED\_PARENT 1

>>>

>>> And then document the fields here.

>>>

>>

>> In hex, please?

>

> Well, they're bit numbers, not masks. Decimal 0-31 is OK, or an enum.

>

enum it will be.

---

Subject: Re: [PATCH 03/11] memcg: change defines to an enum

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 08:28:34 GMT

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

---

On 06/26/2012 08:11 AM, David Rientjes wrote:

> On Mon, 25 Jun 2012, Glauber Costa wrote:

>

>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c

>> index 8e601e8..9352d40 100644

>> --- a/mm/memcontrol.c

>> +++ b/mm/memcontrol.c

>> @@ -387,9 +387,12 @@ enum charge\_type {

```

>> };
>>
>> /* for encoding cft->private value on file */
>> #define _MEM (0)
>> #define _MEMSWAP (1)
>> #define _OOM_TYPE (2)
>> +enum res_type {
>> + _MEM,
>> + _MEMSWAP,
>> + _OOM_TYPE,
>> +};
>> +
>> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
>> #define MEMFILE_TYPE(val) ((val) >> 16 & 0xffff)
>> #define MEMFILE_ATTR(val) ((val) & 0xffff)
>
> Shouldn't everything that does MEMFILE_TYPE() now be using type
> enum res_type rather than int?
>
If you mean the following three fields, no, since they are masks and
operations.

```

If you mean something else, what do you mean ?

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
 Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 08:39:55 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 03:33 AM, Suleiman Souhlal wrote:

```

> On Mon, Jun 25, 2012 at 7:15 AM, Glauber Costa <glommer@parallels.com> wrote:
>> From: Suleiman Souhlal <ssouhlal@FreeBSD.org>
>>
>> mem_cgroup_do_charge() was written before slab accounting, and expects
>> three cases: being called for 1 page, being called for a stock of 32 pages,
>> or being called for a hugepage. If we call for 2 or 3 pages (and several
>> slabs used in process creation are such, at least with the debug options I
>> had), it assumed it's being called for stock and just retried without reclaiming.
>>
>> Fix that by passing down a minsize argument in addition to the csize.
>>
>> And what to do about that (csize == PAGE_SIZE && ret) retry? If it's
>> needed at all (and presumably is since it's there, perhaps to handle
>> races), then it should be extended to more than PAGE_SIZE, yet how far?
>> And should there be a retry count limit, of what? For now retry up to
>> COSTLY_ORDER (as page_alloc.c does), stay safe with a cond_resched(),
>> and make sure not to do it if __GFP_NORETRY.
>

```

> The commit description mentions COSTLY\_ORDER, but it's not actually  
> used in the patch.  
>  
> -- Suleiman  
>  
Yeah, forgot to update the changelog =(

But much more importantly, are you still happy with those changes?

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 08:44:13 GMT

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

---

On 06/26/2012 12:45 PM, David Rientjes wrote:

> On Tue, 26 Jun 2012, Glauber Costa wrote:

>  
>>>> diff --git a/include/linux/thread\_info.h b/include/linux/thread\_info.h  
>>>> index ccc1899..914ec07 100644  
>>>> --- a/include/linux/thread\_info.h  
>>>> +++ b/include/linux/thread\_info.h  
>>>> @@ -61,6 +61,12 @@ extern long do\_no\_restart\_syscall(struct restart\_block  
>>>> \*parm);  
>>>> # define THREADINFO\_GFP (GFP\_KERNEL | \_\_GFP\_NOTRACK)  
>>>> #endif  
>>>>  
>>>> #ifdef CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM  
>>>> # define THREADINFO\_GFP\_ACCOUNTED (THREADINFO\_GFP | \_\_GFP\_KMEMCG)  
>>>> #else  
>>>> # define THREADINFO\_GFP\_ACCOUNTED (THREADINFO\_GFP)  
>>>> #endif  
>>>> +  
>>>>  
>>> This type of requirement is going to become nasty very quickly if nobody  
>>> can use \_\_GFP\_KMEMCG without testing for  
CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM.  
>>> Perhaps define \_\_GFP\_KMEMCG to be 0x0 if it's not enabled, similar to how  
>>> kmemcheck does?  
>>>  
>> That is what I've done in my first version of this patch. At that time,  
>> Christoph wanted it to be this way so we would make sure it would never be  
>> used with #CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM defined. A value of zero will  
>> generate no errors. Undefined value will.  
>>  
>> Now, if you ask me, I personally prefer following what kmemcheck does here...  
>>  
>

> Right, because I'm sure that `__GFP_KMEMCG` will be used in additional  
> places outside of this patchset and it will be a shame if we have to  
> always add `#ifdef`'s. I see no reason why we would care if `__GFP_KMEMCG`  
> was used when `CONFIG_CGROUP_MEM_RES_CTLR_KMEM=n` with the semantics that it  
> as in this patchset. It's much cleaner by making it 0x0 when disabled.  
>

What I can do, instead, is to `WARN_ON` conditionally to the config option  
in the page allocator, and make sure no one is actually passing the flag  
in that case.

---

Subject: Re: [PATCH 11/11] protect architectures where `THREAD_SIZE >=`  
`PAGE_SIZE` against fork bombs

Posted by [David Rientjes](#) on Tue, 26 Jun 2012 08:45:30 GMT

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

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

```
> > > diff --git a/include/linux/thread_info.h b/include/linux/thread_info.h
> > > index ccc1899..914ec07 100644
> > > --- a/include/linux/thread_info.h
> > > +++ b/include/linux/thread_info.h
> > > @@ -61,6 +61,12 @@ extern long do_no_restart_syscall(struct restart_block
> > > *parm);
> > > # define THREADINFO_GFP (GFP_KERNEL | __GFP_NOTRACK)
> > > #endif
> > >
> > > +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> > > +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP | __GFP_KMEMCG)
> > > +#else
> > > +# define THREADINFO_GFP_ACCOUNTED (THREADINFO_GFP)
> > > +#endif
> > > +
> >
> > This type of requirement is going to become nasty very quickly if nobody
> > can use __GFP_KMEMCG without testing for
> > CONFIG_CGROUP_MEM_RES_CTLR_KMEM.
> > Perhaps define __GFP_KMEMCG to be 0x0 if it's not enabled, similar to how
> > kmemcheck does?
> >
> > That is what I've done in my first version of this patch. At that time,
> > Christoph wanted it to be this way so we would make sure it would never be
> > used with #CONFIG_CGROUP_MEM_RES_CTLR_KMEM defined. A value of zero will
> > generate no errors. Undefined value will.
>
> Now, if you ask me, I personally prefer following what kmemcheck does here...
>
```

Right, because I'm sure that `__GFP_KMEMCG` will be used in additional places outside of this patchset and it will be a shame if we have to always add `#ifdef`'s. I see no reason why we would care if `__GFP_KMEMCG` was used when `CONFIG_CGROUP_MEM_RES_CTLR_KMEM=n` with the semantics that it as in this patchset. It's much cleaner by making it 0x0 when disabled.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 08:54:09 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

```
> > + * retries
> > + */
> > + #define NR_PAGES_TO_RETRY 2
> > +
> >
> > Should be 1 << PAGE_ALLOC_COSTLY_ORDER? Where does this number come from?
> > The changelog doesn't specify.
>
> Hocko complained about that, and I changed. Where the number comes from, is
> stated in the comments: it is a number small enough to have high changes of
> had been freed by the previous reclaim, and yet around the number of pages of
> a kernel allocation.
>
```

`PAGE_ALLOC_COSTLY_ORDER` `_is_` the threshold used to determine where reclaim and compaction is deemed to be too costly to continuously retry, I'm not sure why this is any different?

And this is certainly not "around the number of pages of a kernel allocation", that depends very heavily on the slab allocator being used; slub very often uses order-2 and order-3 page allocations as the default settings (it is capped at, you guessed it, `PAGE_ALLOC_COSTLY_ORDER` internally by default) and can be significantly increased on the command line.

```
> Of course there are allocations for nr_pages > 2. But 2 will already service
> the stack most of the time, and most of the slab caches.
>
```

Nope, have you checked the output of `/sys/kernel/slub/.../order` when running slub? On my workstation 127 out of 316 caches have order-2 or higher by default.

---

Subject: Re: [PATCH 03/11] memcg: change defines to an enum  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 09:01:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

```
> > > diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> > > index 8e601e8..9352d40 100644
> > > --- a/mm/memcontrol.c
> > > +++ b/mm/memcontrol.c
> > > @@ -387,9 +387,12 @@ enum charge_type {
> > > };
> > >
> > > /* for encoding cft->private value on file */
> > > #define _MEM (0)
> > > #define _MEMSWAP (1)
> > > #define _OOM_TYPE (2)
> > > +enum res_type {
> > > + _MEM,
> > > + _MEMSWAP,
> > > + _OOM_TYPE,
> > > +};
> > > +
> > > #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
> > > #define MEMFILE_TYPE(val) ((val) >> 16 & 0xffff)
> > > #define MEMFILE_ATTR(val) ((val) & 0xffff)
> >
> > Shouldn't everything that does MEMFILE_TYPE() now be using type
> > enum res_type rather than int?
> >
> > If you mean the following three fields, no, since they are masks and
> > operations.
>
```

No, I mean everything in mm/memcontrol.c that does

```
int type = MEMFILE_TYPE(...).
```

Why define a non-anonymous enum if you're not going to use its type?  
Either use enum res\_type in place of int or define the enum to be  
anonymous.

It's actually quite effective since gcc will warn if you're using the  
value of an enum type in your switch() statements later in this series and  
one of the enum fields is missing (if you avoid using a "default" case  
statement) if you pass -Wswitch, which is included in -Wall.

---

---

Subject: Re: [PATCH 05/11] Add a \_\_GFP\_KMEMCG flag  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 09:03:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

> > > This flag is used to indicate to the callees that this allocation will be  
> > > serviced to the kernel. It is not supposed to be passed by the callers  
> > > of kmem\_cache\_alloc, but rather by the cache core itself.  
> > >  
> > Not sure what "serviced to the kernel" means, does this mean that the  
> > memory will not be accounted for to the root memcg?  
> >  
> In this context, it means that is a kernel allocation, not a userspace one  
> (but in process context, of course), \*and\* it is to be accounted a  
> specific memcg.  
>

Ah, that makes sense. I think it would help if this was included in the changelog as well as a specifying that it is accounted to current's memcg at the time of the allocation in a comment in the code.

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 09:05:47 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

> > Right, because I'm sure that \_\_GFP\_KMEMCG will be used in additional  
> > places outside of this patchset and it will be a shame if we have to  
> > always add #ifdef's. I see no reason why we would care if \_\_GFP\_KMEMCG  
> > was used when CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM=n with the semantics that it  
> > as in this patchset. It's much cleaner by making it 0x0 when disabled.  
> >  
>  
> What I can do, instead, is to WARN\_ON conditionally to the config option in  
> the page allocator, and make sure no one is actually passing the flag in that  
> case.  
>

I don't think adding a conditional to the page allocator's fastpath when CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM=n is appropriate. I don't understand why this can't be 0x0 for such a configuration, \_\_GFP\_KMEM certainly means nothing when we don't have it enabled so how is this different at all from kmemcheck?

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 09:08:48 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 12:54 PM, David Rientjes wrote:

> On Tue, 26 Jun 2012, Glauber Costa wrote:

>  
>>>> + \* retries  
>>>> + \*/  
>>>> +#define NR\_PAGES\_TO\_RETRY 2  
>>>> +  
>>>> +  
>>>> +  
>>>> Should be 1 << PAGE\_ALLOC\_COSTLY\_ORDER? Where does this number come from?  
>>>> The changelog doesn't specify.  
>>>>  
>>>> Hocko complained about that, and I changed. Where the number comes from, is  
>>>> stated in the comments: it is a number small enough to have high changes of  
>>>> had been freed by the previous reclaim, and yet around the number of pages of  
>>>> a kernel allocation.  
>>>>  
>>>>  
>>>> PAGE\_ALLOC\_COSTLY\_ORDER \_is\_ the threshold used to determine where reclaim  
>>>> and compaction is deemed to be too costly to continuously retry, I'm not  
>>>> sure why this is any different?  
>>>>  
>>>> And this is certainly not "around the number of pages of a kernel  
>>>> allocation", that depends very heavily on the slab allocator being used;  
>>>> slub very often uses order-2 and order-3 page allocations as the default  
>>>> settings (it is capped at, you guessed it, PAGE\_ALLOC\_COSTLY\_ORDER  
>>>> internally by default) and can be significantly increased on the command  
>>>> line.

I am obviously okay with either.

Maybe Michal can comment on this?

>>>> Of course there are allocations for nr\_pages > 2. But 2 will already service  
>>>> the stack most of the time, and most of the slab caches.  
>>>>  
>>>>  
>>>> Nope, have you checked the output of /sys/kernel/slab/.../order when  
>>>> running slub? On my workstation 127 out of 316 caches have order-2 or  
>>>> higher by default.  
>>>>

Well, this is still on the side of my argument, since this is still a majority of them being low ordered. The code here does not necessarily have to retry - if I understand it correctly - we just retry for very small allocations because that is where our likelihood of succeeding is.

---

---



Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 09:12:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, 25 Jun 2012, Glauber Costa wrote:

```
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 83e7ba9..22479eb 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;
> @@ -409,6 +410,12 @@ struct sock;
> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> void sock_update_memcg(struct sock *sk);
> void sock_release_memcg(struct sock *sk);
> +
> +#define mem_cgroup_kmem_on 1
> +bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
> +void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
> +void __mem_cgroup_free_kmem_page(struct page *page, int order);
> +#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)
> #else
> static inline void sock_update_memcg(struct sock *sk)
> {
> @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
> static inline void sock_release_memcg(struct sock *sk)
> {
> }
> +
> +#define mem_cgroup_kmem_on 0
> +#define __mem_cgroup_new_kmem_page(a, b, c) false
> +#define __mem_cgroup_free_kmem_page(a,b )
> +#define __mem_cgroup_commit_kmem_page(a, b, c)
> +#define is_kmem_tracked_alloc (false)
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> +
> +static __always_inline
> +bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
> +{
> + if (!mem_cgroup_kmem_on)
> + return true;
> + if (!is_kmem_tracked_alloc)
```

```

> + return true;
> + if (!current->mm)
> + return true;
> + if (in_interrupt())
> + return true;

```

You can't test for current->mm in irq context, so you need to check for in\_interrupt() first. Also, what prevents \_\_mem\_cgroup\_new\_kmem\_page() from being called for a kthread that has called use\_mm() before unuse\_mm()?

```

> + if (gfp & __GFP_NOFAIL)
> + return true;
> + return __mem_cgroup_new_kmem_page(gfp, handle, order);
> +}
> +
> +static __always_inline
> +void mem_cgroup_free_kmem_page(struct page *page, int order)
> +{
> + if (mem_cgroup_kmem_on)
> + __mem_cgroup_free_kmem_page(page, order);
> +}
> +
> +static __always_inline
> +void mem_cgroup_commit_kmem_page(struct page *page, struct mem_cgroup *handle,
> + int order)
> +{
> + if (mem_cgroup_kmem_on)
> + __mem_cgroup_commit_kmem_page(page, handle, order);
> +}
> #endif /* _LINUX_MEMCONTROL_H */
>

```

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 09:17:37 GMT

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

---

On 06/26/2012 01:12 PM, David Rientjes wrote:

```

> On Mon, 25 Jun 2012, Glauber Costa wrote:
>
>> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
>> index 83e7ba9..22479eb 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;
>> @@ -409,6 +410,12 @@ struct sock;
>> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> void sock_update_memcg(struct sock *sk);
>> void sock_release_memcg(struct sock *sk);
>> +
>> +#define mem_cgroup_kmem_on 1
>> +bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order);
>> +void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
>> +void __mem_cgroup_free_kmem_page(struct page *page, int order);
>> +#define is_kmem_tracked_alloc (gfp & __GFP_KMEMCG)
>> #else
>> static inline void sock_update_memcg(struct sock *sk)
>> {
>> @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
>> static inline void sock_release_memcg(struct sock *sk)
>> {
>> }
>> +
>> +#define mem_cgroup_kmem_on 0
>> +#define __mem_cgroup_new_kmem_page(a, b, c) false
>> +#define __mem_cgroup_free_kmem_page(a,b )
>> +#define __mem_cgroup_commit_kmem_page(a, b, c)
>> +#define is_kmem_tracked_alloc (false)
>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>> +
>> +static __always_inline
>> +bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
>> +{
>> + if (!mem_cgroup_kmem_on)
>> + return true;
>> + if (!is_kmem_tracked_alloc)
>> + return true;
>> + if (!current->mm)
>> + return true;
>> + if (in_interrupt())
>> + return true;
>
> You can't test for current->mm in irq context, so you need to check for
> in_interrupt() first.
>
> Right, thanks.

```

> Also, what prevents \_\_mem\_cgroup\_new\_kmem\_page()

> from being called for a kthread that has called use\_mm() before  
> unuse\_mm()?

Nothing, but I also don't see how to prevent that.

At a first glance, it seems fair to me to say that if a kernel thread uses the mm of a process, it poses as this process for any accounting purpose.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [David Rientjes](#) on Tue, 26 Jun 2012 09:17:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

> > Nope, have you checked the output of /sys/kernel/slab/.../order when  
> > running slub? On my workstation 127 out of 316 caches have order-2 or  
> > higher by default.  
> >  
>  
> Well, this is still on the side of my argument, since this is still a majority  
> of them being low ordered.

Ok, so what happens if I pass slub\_min\_order=2 on the command line? We never retry?

> The code here does not necessarily have to retry -  
> if I understand it correctly - we just retry for very small allocations  
> because that is where our likelihood of succeeding is.  
>

Well, the comment for NR\_PAGES\_TO\_RETRY says

```
/*  
 * We need a number that is small enough to be likely to have been  
 * reclaimed even under pressure, but not too big to trigger unnecessary  
 * retries  
 */
```

and mmzone.h says

```
/*  
 * PAGE_ALLOC_COSTLY_ORDER is the order at which allocations are deemed  
 * costly to service. That is between allocation orders which should  
 * coalesce naturally under reasonable reclaim pressure and those which  
 * will not.  
 */  
#define PAGE_ALLOC_COSTLY_ORDER 3
```

so I'm trying to reconcile which one is correct.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 09:23:40 GMT

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

---

On 06/26/2012 01:17 PM, David Rientjes wrote:

> On Tue, 26 Jun 2012, Glauber Costa wrote:

>

>>> Nope, have you checked the output of /sys/kernel/slub/.../order when  
>>> running slub? On my workstation 127 out of 316 caches have order-2 or  
>>> higher by default.

>>>

>>

>> Well, this is still on the side of my argument, since this is still a majority  
>> of them being low ordered.

>

> Ok, so what happens if I pass slub\_min\_order=2 on the command line? We  
> never retry?

Well, indeed. The function has many other RETRY points, but I believe we'd reach none of them without triggering oom first.

>> The code here does not necessarily have to retry -  
>> if I understand it correctly - we just retry for very small allocations  
>> because that is where our likelihood of succeeding is.

>>

>

> Well, the comment for NR\_PAGES\_TO\_RETRY says

>

> /\*

> \* We need a number that is small enough to be likely to have been  
> \* reclaimed even under pressure, but not too big to trigger unnecessary  
> \* retries  
> \*/

>

> and mmzone.h says

>

> /\*

> \* PAGE\_ALLOC\_COSTLY\_ORDER is the order at which allocations are deemed  
> \* costly to service. That is between allocation orders which should  
> \* coalesce naturally under reasonable reclaim pressure and those which  
> \* will not.

> \*/

> #define PAGE\_ALLOC\_COSTLY\_ORDER 3

>

> so I'm trying to reconcile which one is correct.

>

I am not myself against reverting back to costly order. The check we have here, of course, is stricter than costly order, so there is nothing to reconcile. Now if we really need a stricter check or not, is the subject of discussion.

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 12:48:08 GMT

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

---

On 06/25/2012 10:38 PM, Tejun Heo wrote:

> On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:

>> On 06/25/2012 04:15 PM, Glauber Costa wrote:

>>>

>>> Because those architectures will draw their stacks directly from  
>>> the page allocator, rather than the slab cache, we can directly  
>>> pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.

>>>

>>> This code path is taken when the architecture doesn't define  
>>> CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
>>> THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
>>> remaining architectures fall in this category.

>>>

>>> This will guarantee that every stack page is accounted to the memcg  
>>> the process currently lives on, and will have the allocations to fail  
>>> if they go over limit.

>>>

>>> For the time being, I am defining a new variant of THREADINFO\_GFP, not  
>>> to mess with the other path. Once the slab is also tracked by memcg,  
>>> we can get rid of that flag.

>>>

>>> Tested to successfully protect against :(){ :|:& };:

>>>

>>> 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>

>>> CC: Suleiman Souhlal <suleiman@google.com>

>>

>>

>> Acked-by: Frederic Weisbecker <fweisbec@redhat.com>

>

> Frederic, does this (with proper slab accounting added later) achieve  
> what you wanted with the task counter?  
>

A note: Frederic may confirm, but I think he doesn't even need  
the slab accounting to follow to achieve that goal.

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 13:37:41 GMT

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

---

On 06/26/2012 05:38 PM, Frederic Weisbecker wrote:

> On Tue, Jun 26, 2012 at 04:48:08PM +0400, Glauber Costa wrote:

>> On 06/25/2012 10:38 PM, Tejun Heo wrote:

>>> On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:

>>>> On 06/25/2012 04:15 PM, Glauber Costa wrote:

>>>>>

>>>>> Because those architectures will draw their stacks directly from  
>>>>> the page allocator, rather than the slab cache, we can directly  
>>>>> pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.

>>>>>

>>>>> This code path is taken when the architecture doesn't define  
>>>>> CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
>>>>> THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
>>>>> remaining architectures fall in this category.

>>>>>

>>>>> This will guarantee that every stack page is accounted to the memcg  
>>>>> the process currently lives on, and will have the allocations to fail  
>>>>> if they go over limit.

>>>>>

>>>>> For the time being, I am defining a new variant of THREADINFO\_GFP, not  
>>>>> to mess with the other path. Once the slab is also tracked by memcg,  
>>>>> we can get rid of that flag.

>>>>>

>>>>> Tested to successfully protect against :(){ :|:& };;

>>>>>

>>>>> 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>

>>>>> CC: Suleiman Souhlal <suleiman@google.com>

>>>>>

>>>>>

>>>>> Acked-by: Frederic Weisbecker <fweisbec@redhat.com>

>>>  
>>> Frederic, does this (with proper slab accounting added later) achieve  
>>> what you wanted with the task counter?  
>>>  
>>  
>> A note: Frederic may confirm, but I think he doesn't even need  
>> the slab accounting to follow to achieve that goal.  
>  
> Limiting is enough. But that requires internal accounting.  
>  
Yes, but why the \*slab\* needs to get involved?  
accounting task stack pages should be equivalent to what you  
were doing, even without slab accounting. Right ?

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs

Posted by [Frederic Weisbecker](#) on Tue, 26 Jun 2012 13:38:41 GMT

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

---

On Tue, Jun 26, 2012 at 04:48:08PM +0400, Glauber Costa wrote:

> On 06/25/2012 10:38 PM, Tejun Heo wrote:  
> > On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:  
> >> On 06/25/2012 04:15 PM, Glauber Costa wrote:  
> >>  
> >>> Because those architectures will draw their stacks directly from  
> >>> the page allocator, rather than the slab cache, we can directly  
> >>> pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.  
> >>>  
> >>> This code path is taken when the architecture doesn't define  
> >>> CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
> >>> THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
> >>> remaining architectures fall in this category.  
> >>>  
> >>> This will guarantee that every stack page is accounted to the memcg  
> >>> the process currently lives on, and will have the allocations to fail  
> >>> if they go over limit.  
> >>>  
> >>> For the time being, I am defining a new variant of THREADINFO\_GFP, not  
> >>> to mess with the other path. Once the slab is also tracked by memcg,  
> >>> we can get rid of that flag.  
> >>>  
> >>> Tested to successfully protect against :(){ :|:& };:  
> >>>  
> >>> 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>  
> >>>CC: Suleiman Souhlal <suleiman@google.com>  
> >>  
> >>  
> >>Acked-by: Frederic Weisbecker <fweisbec@redhat.com>  
> >  
> >Frederic, does this (with proper slab accounting added later) achieve  
> >what you wanted with the task counter?  
> >  
>  
> A note: Frederic may confirm, but I think he doesn't even need  
> the slab accounting to follow to achieve that goal.

Limiting is enough. But that requires internal accounting.

---

---

Subject: Re: [PATCH 11/11] protect architectures where THREAD\_SIZE >= PAGE\_SIZE against fork bombs  
Posted by [Frederic Weisbecker](#) on Tue, 26 Jun 2012 13:44:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Jun 26, 2012 at 05:37:41PM +0400, Glauber Costa wrote:  
> On 06/26/2012 05:38 PM, Frederic Weisbecker wrote:  
> >On Tue, Jun 26, 2012 at 04:48:08PM +0400, Glauber Costa wrote:  
> >>On 06/25/2012 10:38 PM, Tejun Heo wrote:  
> >>>On Mon, Jun 25, 2012 at 06:55:35PM +0200, Frederic Weisbecker wrote:  
> >>>>On 06/25/2012 04:15 PM, Glauber Costa wrote:  
> >>>>  
> >>>>>Because those architectures will draw their stacks directly from  
> >>>>>the page allocator, rather than the slab cache, we can directly  
> >>>>>pass \_\_GFP\_KMEMCG flag, and issue the corresponding free\_pages.  
> >>>>>  
> >>>>>This code path is taken when the architecture doesn't define  
> >>>>>CONFIG\_ARCH\_THREAD\_INFO\_ALLOCATOR (only ia64 seems to), and has  
> >>>>>THREAD\_SIZE >= PAGE\_SIZE. Luckily, most - if not all - of the  
> >>>>>remaining architectures fall in this category.  
> >>>>>  
> >>>>>This will guarantee that every stack page is accounted to the memcg  
> >>>>>the process currently lives on, and will have the allocations to fail  
> >>>>>if they go over limit.  
> >>>>>  
> >>>>>For the time being, I am defining a new variant of THREADINFO\_GFP, not  
> >>>>>to mess with the other path. Once the slab is also tracked by memcg,  
> >>>>>we can get rid of that flag.  
> >>>>>  
> >>>>>Tested to successfully protect against :(){ :|:& };;  
> >>>>>

> >>>>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>  
> >>>>CC: Suleiman Souhlal <suleiman@google.com>  
> >>>>  
> >>>>  
> >>>>Acked-by: Frederic Weisbecker <fweisbec@redhat.com>  
> >>>>  
> >>>>Frederic, does this (with proper slab accounting added later) achieve  
> >>>>what you wanted with the task counter?  
> >>>>  
> >>>>  
> >>>>A note: Frederic may confirm, but I think he doesn't even need  
> >>>>the slab accounting to follow to achieve that goal.  
> >>>>  
> >>>>Limiting is enough. But that requires internal accounting.  
> >>>>  
> >>>>Yes, but why the \*slab\* needs to get involved?  
> >>>>accounting task stack pages should be equivalent to what you  
> >>>>were doing, even without slab accounting. Right ?

Yeah that alone should be fine.

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 14:40:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 03:17 AM, Andrew Morton wrote:

```
>> }  
>> >+  
>> >+#define mem_cgroup_kmem_on 0  
>> >+#define __mem_cgroup_new_kmem_page(a, b, c) false  
>> >+#define __mem_cgroup_free_kmem_page(a,b )  
>> >+#define __mem_cgroup_commit_kmem_page(a, b, c)  
> I suggest that the naming consistently follow the model  
> "mem_cgroup_kmem_foo". So "mem_cgroup_kmem_" becomes the well-known  
> identifier for this subsystem.  
>  
> Then, s/mem_cgroup/memcg/g/ - show us some mercy here!  
>  
I always prefer shorter names, but mem_cgroup, and not memcg, seems to  
be the default for external functions.
```

I am nothing but a follower =)

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 15:01:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 03:17 AM, Andrew Morton wrote:

```
>> + memcg_uncharge_kmem(memcg, size);
>> >+ mem_cgroup_put(memcg);
>> >+}
>> >+EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
>> > #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>> >
>> > #if defined(CONFIG_INET) && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
>> >@@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
>> > __setup("swapaccount=", enable_swap_account);
>> >
>> > #endif
>> >+
>> >+ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> gargh. CONFIG_MEMCG_KMEM, please!
>
```

Here too. I like it as much as you do.

But that is consistent with the rest of the file, and I'd rather have it this way.

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 15:29:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 03:17 AM, Andrew Morton wrote:

```
>> + if (ret == -EINTR) {
>> >+ nofail = true;
>> >+ /*
>> >+ * __mem_cgroup_try_charge() chose to bypass to root due
>> >+ * to OOM kill or fatal signal.
> Is "bypass" correct? Maybe "fall back"?
>
```

Heh, forgot this one, sorry =(

\_\_mem\_cgroup\_try\_charge does "goto bypass", so I believe the term "bypass" is better to allow whoever is following this code to follow it.

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure

---

Posted by [akpm](#) on Tue, 26 Jun 2012 18:01:42 GMT

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

---

On Tue, 26 Jun 2012 19:01:15 +0400 Glauber Costa <glommer@parallels.com> wrote:

```
> On 06/26/2012 03:17 AM, Andrew Morton wrote:
> >> + memcg_uncharge_kmem(memcg, size);
> >> >+ mem_cgroup_put(memcg);
> >> >+}
> >> >+EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
> >> > #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> >> >
> >> > #if defined(CONFIG_INET) && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
> >> >@@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
> >> > __setup("swapaccount=", enable_swap_account);
> >> >
> >> > #endif
> >> >+
> >> >+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> > gargh. CONFIG_MEMCG_KMEM, please!
> >
>
> Here too. I like it as much as you do.
>
> But that is consistent with the rest of the file, and I'd rather have
> it this way.
```

There's not much point in being consistent with something which is so unpleasant. I'm on a little campaign to rename CONFIG\_CGROUP\_MEM\_RES\_CTLR to CONFIG\_MEMCG, only nobody has taken my bait yet. Be first!

---

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure

Posted by [Tejun Heo](#) on Tue, 26 Jun 2012 18:08:05 GMT

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

---

On Tue, Jun 26, 2012 at 11:01:42AM -0700, Andrew Morton wrote:

```
> On Tue, 26 Jun 2012 19:01:15 +0400 Glauber Costa <glommer@parallels.com> wrote:
>
> > On 06/26/2012 03:17 AM, Andrew Morton wrote:
> > >> + memcg_uncharge_kmem(memcg, size);
> > >> >+ mem_cgroup_put(memcg);
> > >> >+}
> > >> >+EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
> > >> > #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> > >> >
> > >> > #if defined(CONFIG_INET) && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
```

```

> > > > @@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
> > > > __setup("swapaccount=", enable_swap_account);
> > > >
> > > > #endif
> > > >+
> > > >+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> > > gargh. CONFIG_MEMCG_KMEM, please!
> > >
> >
> > Here too. I like it as much as you do.
> >
> > But that is consistent with the rest of the file, and I'd rather have
> > it this way.
>
> There's not much point in being consistent with something which is so
> unpleasant. I'm on a little campaign to rename
> CONFIG_CGROUP_MEM_RES_CTLR to CONFIG_MEMCG, only nobody has taken my
> bait yet. Be first!

```

+1.

Block cgroup recently did blkio / blkioCG / blkio\_cgroup -> blkcg.  
Join the cool crowd! :P

--  
tejun

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Glauber Costa](#) on Tue, 26 Jun 2012 18:14:51 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 10:01 PM, Andrew Morton wrote:  
> On Tue, 26 Jun 2012 19:01:15 +0400 Glauber Costa <glommer@parallels.com> wrote:  
>  
>> On 06/26/2012 03:17 AM, Andrew Morton wrote:  
>>>> + memcg\_uncharge\_kmem(memcg, size);  
>>>>> + mem\_cgroup\_put(memcg);  
>>>>> +}  
>>>>> +EXPORT\_SYMBOL(\_\_mem\_cgroup\_free\_kmem\_page);  
>>>>> #endif /\* CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM \*/  
>>>>>  
>>>>> #if defined(CONFIG\_INET) && defined(CONFIG\_CGROUP\_MEM\_RES\_CTLR\_KMEM)  
>>>>> @@ -5645,3 +5751,69 @@ static int \_\_init enable\_swap\_account(char \*s)  
>>>>> \_\_setup("swapaccount=", enable\_swap\_account);  
>>>>>  
>>>>> #endif  
>>>>> +

```

>>>> +ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>>> gargh. CONFIG_MEMCG_KMEM, please!
>>>
>>
>> Here too. I like it as much as you do.
>>
>> But that is consistent with the rest of the file, and I'd rather have
>> it this way.
>
> There's not much point in being consistent with something which is so
> unpleasant. I'm on a little campaign to rename
> CONFIG_CGROUP_MEM_RES_CTLR to CONFIG_MEMCG, only nobody has taken my
> bait yet. Be first!
>
>

```

If you are okay with a preparation mechanical patch to convert the whole file, I can change mine too.

But you'll be responsible for arguing with whoever stepping up opposing this =p

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
 Posted by [akpm](#) on Tue, 26 Jun 2012 19:20:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012 22:14:51 +0400  
 Glauber Costa <glommer@parallels.com> wrote:

```

> On 06/26/2012 10:01 PM, Andrew Morton wrote:
> > On Tue, 26 Jun 2012 19:01:15 +0400 Glauber Costa <glommer@parallels.com> wrote:
> >
> >> On 06/26/2012 03:17 AM, Andrew Morton wrote:
> >>> + memcg_uncharge_kmem(memcg, size);
> >>>> + mem_cgroup_put(memcg);
> >>>> +}
> >>>> +EXPORT_SYMBOL(__mem_cgroup_free_kmem_page);
> >>>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> >>>>
> >>>> #if defined(CONFIG_INET) &&
defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
> >>>> @@ -5645,3 +5751,69 @@ static int __init enable_swap_account(char *s)
> >>>> __setup("swapaccount=", enable_swap_account);
> >>>>
> >>>> #endif
> >>>> +
> >>>> +ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> >>> gargh. CONFIG_MEMCG_KMEM, please!

```

> >>>  
> >>  
> >> Here too. I like it as much as you do.  
> >>  
> >> But that is consistent with the rest of the file, and I'd rather have  
> >> it this way.  
> >  
> > There's not much point in being consistent with something which is so  
> > unpleasant. I'm on a little campaign to rename  
> > CONFIG\_CGROUP\_MEM\_RES\_CTLR to CONFIG\_MEMCG, only nobody has taken my  
> > bait yet. Be first!  
> >  
>  
> If you are okay with a preparation mechanical patch to convert the whole  
> file, I can change mine too.  
>  
> But you'll be responsible for arguing with whoever stepping up opposing  
> this =p  
>

From: Andrew Morton <akpm@linux-foundation.org>  
Subject: memcg: rename config variables

Sanity:

```
CONFIG_CGROUP_MEM_RES_CTLR -> CONFIG_MEMCG
CONFIG_CGROUP_MEM_RES_CTLR_SWAP -> CONFIG_MEMCG_SWAP
CONFIG_CGROUP_MEM_RES_CTLR_SWAP_ENABLED ->
CONFIG_MEMCG_SWAP_ENABLED
CONFIG_CGROUP_MEM_RES_CTLR_KMEM -> CONFIG_MEMCG_KMEM
```

Cc: Glauber Costa <glommer@parallels.com>  
Cc: Michal Hocko <mhocko@suse.cz>  
Cc: Johannes Weiner <hannes@cmpxchg.org>  
Cc: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>  
Cc: Hugh Dickins <hughd@google.com>  
Cc: Tejun Heo <tj@kernel.org>  
Cc: Aneesh Kumar K.V <aneesh.kumar@linux.vnet.ibm.com>  
Cc: David Rientjes <rientjes@google.com>  
Cc: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>  
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

---

```
include/linux/cgroup_subsys.h | 2 +-
include/linux/memcontrol.h   | 14 ++++++-----
include/linux/mmzone.h       | 8 +++++-
include/linux/page_cgroup.h  | 10 +++++-----
include/linux/sched.h        | 2 +-

```

```

init/Kconfig          | 14 ++++++-----
kernel/fork.c         | 2 +-
mm/hwpoison-inject.c  | 2 +-
mm/memcontrol.c       | 20 ++++++++-----
mm/memory-failure.c   | 2 +-
mm/mmzone.c           | 2 +-
mm/oom_kill.c         | 2 +-
mm/page_cgroup.c      | 2 +-
mm/vmscan.c           | 4 ++--
14 files changed, 43 insertions(+), 43 deletions(-)

```

```

diff -puN kernel/fork.c~a kernel/fork.c
--- a/kernel/fork.c~a
+++ a/kernel/fork.c
@@ -1302,7 +1302,7 @@ static struct task_struct *copy_process(
#ifdef CONFIG_DEBUG_MUTEXES
    p->blocked_on = NULL; /* not blocked yet */
#endif
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
    p->memcg_batch.do_batch = 0;
    p->memcg_batch.memcg = NULL;
#endif

```

```

diff -puN mm/hwpoison-inject.c~a mm/hwpoison-inject.c
--- a/mm/hwpoison-inject.c~a
+++ a/mm/hwpoison-inject.c
@@ -123,7 +123,7 @@ static int pfn_inject_init(void)
    if (!dentry)
        goto fail;

```

```

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
    dentry = debugfs_create_u64("corrupt-filter-memcg", 0600,
        hwpoison_dir, &hwpoison_filter_memcg);
    if (!dentry)

```

```

diff -puN mm/memcontrol.c~a mm/memcontrol.c
--- a/mm/memcontrol.c~a
+++ a/mm/memcontrol.c
@@ -61,12 +61,12 @@ struct cgroup_subsys mem_cgroup_subsys _
#define MEM_CGROUP_RECLAIM_RETRIES 5
static struct mem_cgroup *root_mem_cgroup __read_mostly;

```

```

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
/* Turned on only when memory cgroup is enabled && really_do_swap_account = 1 */
int do_swap_account __read_mostly;

/* for remember boot option*/

```



```

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR_SWAP_ENABLED
+#ifndef CONFIG_MEMCG_SWAP_ENABLED
static int really_do_swap_account __initdata = 1;
#else
static int really_do_swap_account __initdata = 0;
@@ -407,7 +407,7 @@ static void mem_cgroup_get(struct mem_cg
static void mem_cgroup_put(struct mem_cgroup *memcg);

/* Writing them here to avoid exposing memcg's inner layout */
-#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifndef CONFIG_MEMCG_KMEM
#include <net/sock.h>
#include <net/ip.h>

@@ -466,9 +466,9 @@ struct cg_proto *tcp_proto_cgroup(struct
}
EXPORT_SYMBOL(tcp_proto_cgroup);
#endif /* CONFIG_INET */
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+#endif /* CONFIG_MEMCG_KMEM */

-#if defined(CONFIG_INET) && defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
+#if defined(CONFIG_INET) && defined(CONFIG_MEMCG_KMEM)
static void disarm_sock_keys(struct mem_cgroup *memcg)
{
    if (!memcg_proto_activated(&memcg->tcp_mem.cg_proto))
@@ -3085,7 +3085,7 @@ mem_cgroup_uncharge_swapcache(struct pag
}
#endif

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifndef CONFIG_MEMCG_SWAP
/*
 * called from swap_entry_free(). remove record in swap_cgroup and
 * uncharge "memsw" account.
@@ -4518,7 +4518,7 @@ static int mem_cgroup_oom_control_write(
return 0;
}

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifndef CONFIG_MEMCG_KMEM
static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
{
    return mem_cgroup_sockets_init(memcg, ss);
@@ -4608,7 +4608,7 @@ static struct cftype mem_cgroup_files[]
.read_seq_string = mem_control_numa_stat_show,
},
#endif

```

```

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
{
    .name = "memsw.usage_in_bytes",
    .private = MEMFILE_PRIVATE(_MEMSWAP, RES_USAGE),
@@ -4795,7 +4795,7 @@ struct mem_cgroup *parent_mem_cgroup(str
}
EXPORT_SYMBOL(parent_mem_cgroup);

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
static void __init enable_swap_cgroup(void)
{
    if (!mem_cgroup_disabled() && really_do_swap_account)
@@ -5526,7 +5526,7 @@ struct cgroup_subsys mem_cgroup_subsys =
    .__DEPRECATED_clear_css_refs = true,
};

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
static int __init enable_swap_account(char *s)
{
    /* consider enabled if no parameter or 1 is given */
diff -puN mm/memory-failure.c~a mm/memory-failure.c
--- a/mm/memory-failure.c~a
+++ a/mm/memory-failure.c
@@ -128,7 +128,7 @@ static int hwpoison_filter_flags(struct
    * can only guarantee that the page either belongs to the memcg tasks, or is
    * a freed page.
    */

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
u64 hwpoison_filter_memcg;
EXPORT_SYMBOL_GPL(hwpoison_filter_memcg);
static int hwpoison_filter_task(struct page *p)
diff -puN mm/mmzone.c~a mm/mmzone.c
--- a/mm/mmzone.c~a
+++ a/mm/mmzone.c
@@ -96,7 +96,7 @@ void lruvec_init(struct lruvec *lruvec,
    for_each_lru(lru)
        INIT_LIST_HEAD(&lruvec->lists[lru]);

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
    lruvec->zone = zone;
#endif
}
diff -puN mm/oom_kill.c~a mm/oom_kill.c

```

```

--- a/mm/oom_kill.c~a
+++ a/mm/oom_kill.c
@@ -541,7 +541,7 @@ static void check_panic_on_oom(enum oom_
    sysctl_panic_on_oom == 2 ? "compulsory" : "system-wide");
}

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR
+#ifndef CONFIG_MEMCG
void mem_cgroup_out_of_memory(struct mem_cgroup *memcg, gfp_t gfp_mask,
    int order)
{
diff -puN mm/page_cgroup.c~a mm/page_cgroup.c
--- a/mm/page_cgroup.c~a
+++ a/mm/page_cgroup.c
@@ -317,7 +317,7 @@ void __meminit pgdat_page_cgroup_init(st
#endif

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifndef CONFIG_MEMCG_SWAP

static DEFINE_MUTEX(swap_cgroup_mutex);
struct swap_cgroup_ctrl {
diff -puN mm/vmscan.c~a mm/vmscan.c
--- a/mm/vmscan.c~a
+++ a/mm/vmscan.c
@@ -133,7 +133,7 @@ long vm_total_pages; /* The total number
static LIST_HEAD(shrinker_list);
static DECLARE_RWSEM(shrinker_rwsem);

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR
+#ifndef CONFIG_MEMCG
static bool global_reclaim(struct scan_control *sc)
{
    return !sc->target_mem_cgroup;
@@ -2152,7 +2152,7 @@ unsigned long try_to_free_pages(struct z
    return nr_reclaimed;
}

-#ifndef CONFIG_CGROUP_MEM_RES_CTLR
+#ifndef CONFIG_MEMCG

unsigned long mem_cgroup_shrink_node_zone(struct mem_cgroup *memcg,
    gfp_t gfp_mask, bool noswap,
diff -puN init/Kconfig~a init/Kconfig
--- a/init/Kconfig~a
+++ a/init/Kconfig
@@ -686,7 +686,7 @@ config RESOURCE_COUNTERS

```

This option enables controller independent resource accounting infrastructure that works with cgroups.

-config CGROUP\_MEM\_RES\_CTLR

+config MEMCG

bool "Memory Resource Controller for Control Groups"

depends on RESOURCE\_COUNTERS

select MM\_OWNER

@@ -709,9 +709,9 @@ config CGROUP\_MEM\_RES\_CTLR

This config option also selects MM\_OWNER config option, which could in turn add some fork/exit overhead.

-config CGROUP\_MEM\_RES\_CTLR\_SWAP

+config MEMCG\_SWAP

bool "Memory Resource Controller Swap Extension"

- depends on CGROUP\_MEM\_RES\_CTLR && SWAP

+ depends on MEMCG && SWAP

help

Add swap management feature to memory resource controller. When you enable this, you can limit mem+swap usage per cgroup. In other words,

@@ -726,9 +726,9 @@ config CGROUP\_MEM\_RES\_CTLR\_SWAP

if boot option "swapaccount=0" is set, swap will not be accounted.

Now, memory usage of swap\_cgroup is 2 bytes per entry. If swap page size is 4096bytes, 512k per 1Gbytes of swap.

-config CGROUP\_MEM\_RES\_CTLR\_SWAP\_ENABLED

+config MEMCG\_SWAP\_ENABLED

bool "Memory Resource Controller Swap Extension enabled by default"

- depends on CGROUP\_MEM\_RES\_CTLR\_SWAP

+ depends on MEMCG\_SWAP

default y

help

Memory Resource Controller Swap Extension comes with its price in

@@ -739,9 +739,9 @@ config CGROUP\_MEM\_RES\_CTLR\_SWAP\_ENABLED

For those who want to have the feature enabled by default should

select this option (if, for some reason, they need to disable it then swapaccount=0 does the trick).

-config CGROUP\_MEM\_RES\_CTLR\_KMEM

+config MEMCG\_KMEM

bool "Memory Resource Controller Kernel Memory accounting (EXPERIMENTAL)"

- depends on CGROUP\_MEM\_RES\_CTLR && EXPERIMENTAL

+ depends on MEMCG && EXPERIMENTAL

default n

help

The Kernel Memory extension for Memory Resource Controller can limit

diff -puN include/linux/cgroup\_subsys.h~a include/linux/cgroup\_subsys.h

--- a/include/linux/cgroup\_subsys.h~a

+++ a/include/linux/cgroup\_subsys.h

@@ -31,7 +31,7 @@ SUBSYS(cpuacct)

```

/* */

#ifndef CONFIG_CGROUP_MEM_RES_CTLR
#ifdef CONFIG_MEMCG
SUBSYS(mem_cgroup)
#endif

diff -puN include/linux/memcontrol.h~a include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~a
+++ a/include/linux/memcontrol.h
@@ -38,7 +38,7 @@ struct mem_cgroup_reclaim_cookie {
    unsigned int generation;
};

#ifndef CONFIG_CGROUP_MEM_RES_CTLR
#ifdef CONFIG_MEMCG
/*
 * All "charge" functions with gfp_mask should use GFP_KERNEL or
 * (gfp_mask & GFP_RECLAIM_MASK). In current implementatin, memcg doesn't
@@ -124,7 +124,7 @@ extern void mem_cgroup_print_oom_info(st
extern void mem_cgroup_replace_page_cache(struct page *oldpage,
    struct page *newpage);

#ifndef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
#ifdef CONFIG_MEMCG_SWAP
extern int do_swap_account;
#endif

@@ -193,7 +193,7 @@ void mem_cgroup_split_huge_fixup(struct
bool mem_cgroup_bad_page_check(struct page *page);
void mem_cgroup_print_bad_page(struct page *page);
#endif
#else /* CONFIG_CGROUP_MEM_RES_CTLR */
#else /* CONFIG_MEMCG */
struct mem_cgroup;

static inline int mem_cgroup_newpage_charge(struct page *page,
@@ -384,9 +384,9 @@ static inline void mem_cgroup_replace_pa
    struct page *newpage)
{
}
#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
#endif /* CONFIG_MEMCG */

#ifndef !defined(CONFIG_CGROUP_MEM_RES_CTLR) || !defined(CONFIG_DEBUG_VM)
#ifndef !defined(CONFIG_MEMCG) || !defined(CONFIG_DEBUG_VM)
static inline bool

```

```

mem_cgroup_bad_page_check(struct page *page)
{
@@ -406,7 +406,7 @@ enum {
};

struct sock;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifdef CONFIG_MEMCG_KMEM
void sock_update_memcg(struct sock *sk);
void sock_release_memcg(struct sock *sk);
#else
@@ -416,6 +416,6 @@ static inline void sock_update_memcg(str
static inline void sock_release_memcg(struct sock *sk)
{
}
-#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+#endif /* CONFIG_MEMCG_KMEM */
#endif /* _LINUX_MEMCONTROL_H */

diff -puN include/linux/mmzone.h~a include/linux/mmzone.h
--- a/include/linux/mmzone.h~a
+++ a/include/linux/mmzone.h
@@ -200,7 +200,7 @@ struct zone_reclaim_stat {
struct lruvec {
    struct list_head lists[NR_LRU_LISTS];
    struct zone_reclaim_stat reclaim_stat;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
    struct zone *zone;
#endif
};
@@ -670,7 +670,7 @@ typedef struct pglist_data {
    int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    struct page *node_mem_map;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
    struct page_cgroup *node_page_cgroup;
#endif
#endif
@@ -735,7 +735,7 @@ extern void lruvec_init(struct lruvec *l
static inline struct zone *lruvec_zone(struct lruvec *lruvec)
{
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
    return lruvec->zone;
#else

```

```

return container_of(lruvec, struct zone, lruvec);
@@ -1051,7 +1051,7 @@ struct mem_section {

/* See declaration of similar field in struct zone */
unsigned long *pageblock_flags;
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
/*
 * If !SPARSEMEM, pgdat doesn't have page_cgroup pointer. We use
 * section. (see memcontrol.h/page_cgroup.h about this.)
diff -puN include/linux/page_cgroup.h~a include/linux/page_cgroup.h
--- a/include/linux/page_cgroup.h~a
+++ a/include/linux/page_cgroup.h
@@ -12,7 +12,7 @@ enum {
#ifdef __GENERATING_BOUNDS_H
#include <generated/bounds.h>

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_MEMCG
#include <linux/bit_spinlock.h>

/*
@@ -82,7 +82,7 @@ static inline void unlock_page_cgroup(st
    bit_spin_unlock(PCG_LOCK, &pc->flags);
}

-#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+#else /* CONFIG_MEMCG */
struct page_cgroup;

static inline void __meminit pgdat_page_cgroup_init(struct pglist_data *pgdat)
@@ -102,11 +102,11 @@ static inline void __init page_cgroup_in
{
}

-#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+#endif /* CONFIG_MEMCG */

#include <linux/swap.h>

-#ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
+#ifdef CONFIG_MEMCG_SWAP
extern unsigned short swap_cgroup_cmpxchg(swp_entry_t ent,
    unsigned short old, unsigned short new);
extern unsigned short swap_cgroup_record(swp_entry_t ent, unsigned short id);
@@ -138,7 +138,7 @@ static inline void swap_cgroup_swapoff(i
    return;
}

```

```

-#endif /* CONFIG_CGROUP_MEM_RES_CTLR_SWAP */
+#endif /* CONFIG_MEMCG_SWAP */

#endif /* !__GENERATING_BOUNDS_H */

diff -puN include/linux/sched.h~a include/linux/sched.h
--- a/include/linux/sched.h~a
+++ a/include/linux/sched.h
@@ -1581,7 +1581,7 @@ struct task_struct {
    /* bitmask and counter of trace recursion */
    unsigned long trace_recursion;
#endif /* CONFIG_TRACING */
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR /* memcg uses this to do batch job */
+#ifdef CONFIG_MEMCG /* memcg uses this to do batch job */
    struct memcg_batch_info {
        int do_batch; /* incremented when batch uncharge started */
        struct mem_cgroup *memcg; /* target memcg of uncharge */
diff -puN include/linux/swap.h~a include/linux/swap.h
-

```

---

Subject: Re: [PATCH 00/11] kmem controller for memcg: stripped down version  
 Posted by [akpm](#) on Tue, 26 Jun 2012 21:55:39 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012 11:17:49 +0400  
 Glauber Costa <glommer@parallels.com> wrote:

```

> On 06/26/2012 03:27 AM, Andrew Morton wrote:
> > On Mon, 25 Jun 2012 18:15:17 +0400
> > Glauber Costa <glommer@parallels.com> wrote:
> >
> >> What I am proposing with this series is a stripped down version of the
> >> kmem controller for memcg that would allow us to merge significant parts
> >> of the infrastructure, while leaving out, for now, the polemic bits about
> >> the slab while it is being reworked by Cristoph.
> >>
> >> Me reasoning for that is that after the last change to introduce a gfp
> >> flag to mark kernel allocations, it became clear to me that tracking other
> >> resources like the stack would then follow extremely naturally. I figured
> >> that at some point we'd have to solve the issue pointed by David, and avoid
> >> testing the Slab flag in the page allocator, since it would soon be made
> >> more generic. I do that by having the callers to explicit mark it.
> >>
> >> So to demonstrate how it would work, I am introducing a stack tracker here,
> >> that is already a functionality per-se: it successfully stops fork bombs to
> >> happen. (Sorry for doing all your work, Frederic =p ). Note that after all

```



> >> memcg infrastructure is deployed, it becomes very easy to track anything.  
> >> The last patch of this series is extremely simple.  
> >>  
> >> The infrastructure is exactly the same we had in memcg, but stripped down  
> >> of the slab parts. And because what we have after those patches is a feature  
> >> per-se, I think it could be considered for merging.  
> >  
> > hm. None of this new code makes the kernel smaller, faster, easier to  
> > understand or more fun to read!  
> Not sure if this is a general comment - in case I agree - or if targeted  
> to my statement that this is "stripped down". If so, it is of course  
> smaller relative to my previous slab accounting patches.

It's a general comment. The patch adds overhead: runtime costs and maintenance costs. Do its benefits justify that cost?

> The infrastructure is largely common, but I realized that a future user,  
> tracking the stack, would be a lot simpler and could be done first.  
>  
> > Presumably we're getting some benefit for all the downside. When the  
> > time is appropriate, please do put some time into explaining that  
> > benefit, so that others can agree that it is a worthwhile tradeoff.  
> >  
>  
> Well, for one thing, we stop fork bombs for processes inside cgroups.

"inside cgroups" is a significant limitation! Is this capability important enough to justify adding the new code? That's unobvious (to me).

Are there any other user-facing things which we can do with this feature? Present, or planned?

> I can't speak for everybody here, but AFAIK, tracking the stack through  
> the memory it used, therefore using my proposed kmem controller, was an  
> idea that good quite a bit of traction with the memcg/memory people.  
> So here you have something that people already asked a lot for, in a  
> shape and interface that seem to be acceptable.

mm, maybe. Kernel developers tend to look at code from the point of view "does it work as designed", "is it clean", "is it efficient", "do I understand it", etc. We often forget to step back and really consider whether or not it should be merged at all.

I mean, unless the code is an explicit simplification, we should have a very strong bias towards "don't merge".

Subject: Re: [PATCH 00/11] kmem controller for memcg: stripped down version  
Posted by [David Rientjes](#) on Wed, 27 Jun 2012 01:08:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Andrew Morton wrote:

> mm, maybe. Kernel developers tend to look at code from the point of  
> view "does it work as designed", "is it clean", "is it efficient", "do  
> I understand it", etc. We often forget to step back and really  
> consider whether or not it should be merged at all.  
>

It's appropriate for true memory isolation so that applications cannot cause an excess of slab to be consumed. This allows other applications to have higher reservations without the risk of incurring a global oom condition as the result of the usage of other memcgs.

I'm not sure whether it would ever be appropriate to limit the amount of slab for an individual slab cache, however, instead of limiting the sum of all slab for a set of processes. With cache merging in slub this would seem to be difficult to do correctly.

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [David Rientjes](#) on Wed, 27 Jun 2012 04:01:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 26 Jun 2012, Glauber Costa wrote:

```
> > > @@ -416,6 +423,43 @@ static inline void sock_update_memcg(struct sock *sk)
> > > static inline void sock_release_memcg(struct sock *sk)
> > > {
> > > }
> > > +
> > > + #define mem_cgroup_kmem_on 0
> > > + #define __mem_cgroup_new_kmem_page(a, b, c) false
> > > + #define __mem_cgroup_free_kmem_page(a, b)
> > > + #define __mem_cgroup_commit_kmem_page(a, b, c)
> > > + #define is_kmem_tracked_alloc (false)
> > > + #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> > > +
> > > + static __always_inline
> > > + bool mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int order)
> > > + {
> > > + if (!mem_cgroup_kmem_on)
> > > + return true;
> > > + if (!is_kmem_tracked_alloc)
> > > + return true;
```

```
> > > + if (!current->mm)
> > > + return true;
> > > + if (in_interrupt())
> > > + return true;
> >
> > You can't test for current->mm in irq context, so you need to check for
> > in_interrupt() first.
> >
> > Right, thanks.
>
> > Also, what prevents __mem_cgroup_new_kmem_page()
> > from being called for a kthread that has called use_mm() before
> > unuse_mm()?
>
> > Nothing, but I also don't see how to prevent that.
```

You can test for current->flags & PF\_KTHREAD following the check for in\_interrupt() and return true, it's what you were trying to do with the check for !current->mm.

---

---

Subject: Re: [PATCH 00/11] kmem controller for memcg: stripped down version  
Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 08:39:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/27/2012 05:08 AM, David Rientjes wrote:

```
> On Tue, 26 Jun 2012, Andrew Morton wrote:
>
>> mm, maybe. Kernel developers tend to look at code from the point of
>> view "does it work as designed", "is it clean", "is it efficient", "do
>> I understand it", etc. We often forget to step back and really
>> consider whether or not it should be merged at all.
>>
>
> > It's appropriate for true memory isolation so that applications cannot
> > cause an excess of slab to be consumed. This allows other applications to
> > have higher reservations without the risk of incurring a global oom
> > condition as the result of the usage of other memcgs.
```

Just a note for Andrew, we we're in the same page: The slab cache limitation is not included in *this* particular series. The goal was always to have other kernel resources limited as well, and the general argument from David holds: we want a set of applications to run truly independently from others, without creating memory pressure on the global system.

The way history develop in this series, I started from the slab cache, and a page-level tracking appeared on that series. I then figured it

would be better to start tracking something that is totally page-based, such as the stack - that already accounts for 70 % of the infrastructure, and then merge the slab code later. In this sense, it was just a strategy inversion. But both are, and were, in the goals.

> I'm not sure whether it would ever be appropriate to limit the amount of  
> slab for an individual slab cache, however, instead of limiting the sum of  
> all slab for a set of processes. With cache merging in slub this would  
> seem to be difficult to do correctly.

Yes, I do agree.

---

Subject: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down versio

Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 09:29:04 GMT

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

---

On 06/27/2012 01:55 AM, Andrew Morton wrote:

> On Tue, 26 Jun 2012 11:17:49 +0400

> Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

>

>> On 06/26/2012 03:27 AM, Andrew Morton wrote:

>>> On Mon, 25 Jun 2012 18:15:17 +0400

>>> Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:

>>>

>>>> What I am proposing with this series is a stripped down version of the  
>>>> kmem controller for memcg that would allow us to merge significant parts  
>>>> of the infrastructure, while leaving out, for now, the polemic bits about  
>>>> the slab while it is being reworked by Cristoph.

>>>>

>>>> Me reasoning for that is that after the last change to introduce a gfp  
>>>> flag to mark kernel allocations, it became clear to me that tracking other  
>>>> resources like the stack would then follow extremely naturally. I figured  
>>>> that at some point we'd have to solve the issue pointed by David, and avoid  
>>>> testing the Slab flag in the page allocator, since it would soon be made  
>>>> more generic. I do that by having the callers to explicit mark it.

>>>>

>>>> So to demonstrate how it would work, I am introducing a stack tracker here,  
>>>> that is already a functionality per-se: it successfully stops fork bombs to  
>>>> happen. (Sorry for doing all your work, Frederic =p ). Note that after all  
>>>> memcg infrastructure is deployed, it becomes very easy to track anything.  
>>>> The last patch of this series is extremely simple.

>>>>

>>>> The infrastructure is exactly the same we had in memcg, but stripped down  
>>>> of the slab parts. And because what we have after those patches is a feature  
>>>> per-se, I think it could be considered for merging.

>>>

>>> hm. None of this new code makes the kernel smaller, faster, easier to  
>>> understand or more fun to read!  
>> Not sure if this is a general comment - in case I agree - or if targeted  
>> to my statement that this is "stripped down". If so, it is of course  
>> smaller relative to my previous slab accounting patches.  
>  
> It's a general comment. The patch adds overhead: runtime costs and  
> maintenance costs. Do its benefits justify that cost?

Despite potential disagreement on the following statement from my peer  
and friends,  
I am not crazy. This means I of course believe so =)

I will lay down my views below, but I believe the general justification  
was given already by people commenting in the task counter subsystem,  
and I invite them (newly CCd) to chime in here if they want.

For whoever is arriving in the thread now, this is about limiting the  
amount of kernel memory used by a set of processes, aka cgroup.

In this particular state - more is to follow - we're able to limit  
fork bombs in a clean way, since overlimit processes will be unable  
to allocate their stack.

>> The infrastructure is largely common, but I realized that a future user,  
>> tracking the stack, would be a lot simpler and could be done first.  
>>  
>>> Presumably we're getting some benefit for all the downside. When the  
>>> time is appropriate, please do put some time into explaining that  
>>> benefit, so that others can agree that it is a worthwhile tradeoff.  
>>>  
>>  
>> Well, for one thing, we stop fork bombs for processes inside cgroups.  
>  
> "inside cgroups" is a significant limitation! Is this capability  
> important enough to justify adding the new code? That's unobvious  
> (to me).

Again, for one thing. The general mechanism here is to limit the amount  
of kernel memory used by a particular set of processes. Processes use  
stack, and if they can't grab more pages for the stack, no new  
processes can be created.

But there are other things the general mechanism protects against.

Using too much of pinned dentry and inode cache, by touching files  
and leaving them in memory forever.

In fact, a simple:

```
while true; do mkdir x; cd x; done
```

can halt your system easily, because the file system limits are hard to reach (big disks), but the kernel memory is not.

Those are examples, but the list certainly don't stop here.

Our particular use case is concerned with people offering hosting services. In a physical box, we can put a limit to some resources, like total number of processes or threads. But in an environment where each independent user gets its own piece of the machine, we don't want a potentially malicious user to destroy good users' services

This might be true for systemd as well, that now groups services inside cgroups. They generally want to put forward a set of guarantees that limits the running service in a variety of ways, so that if they become badly behaved, they won't interfere with the rest of the system.

fork bombs are a way bad behaved processes interfere with the rest of the system. In here, I propose fork bomb stopping as a natural consequence of the fact that the amount of kernel memory can be limited, and each process uses 1 or 2 pages for the stack, that are freed when the process goes away.

The limitation "inside cgroups" is not as important as you seem to state. Everything can be put "inside cgroups". Being inside cgroups is just a way to state "I want this feature". And in this case: I want to pay the price.

Yes, because as you said yourself, of course there is a cost for that. But I am extensively using static branches to make sure that even if the feature is compiled in, even if we have a bunch of memory cgroups deployed (that already pay a price for that), this code will only be enabled after the first user of this service configures any limit.

The impact before that, is as low as it can be.

After that is enabled, processes living outside of limited cgroups will perform an inline test for a flag, and continue normal page allocation with only that price paid: a flag test.

> Are there any other user-facing things which we can do with this  
> feature? Present, or planned?  
>

Yes. We can establish all sorts of boundaries to a group of processes. The dentry / icache bombing limitation is an example of that as well.

But that will need the slab part of this patchset to follow. I not only have the code for that, but I posted it for review many times. That is not progressing fully so far because we're still looking for ways to keep the changes in the allocators to a minimum. Until I realized that the infrastructure could be used as-is, and stripped down from its original slab version to account pages, its proposed second user. And this is what you see here now.

So everything that uses resources in the kernel can proceed until it uses too much. Be it processes, files, or anything that consumes kernel memory - and may potentially leave it there forever.

>> I can't speak for everybody here, but AFAIK, tracking the stack through  
>> the memory it used, therefore using my proposed kmem controller, was an  
>> idea that good quite a bit of traction with the memcg/memory people.  
>> So here you have something that people already asked a lot for, in a  
>> shape and interface that seem to be acceptable.

>  
> mm, maybe. Kernel developers tend to look at code from the point of  
> view "does it work as designed", "is it clean", "is it efficient", "do  
> I understand it", etc. We often forget to step back and really  
> consider whether or not it should be merged at all.  
>  
> I mean, unless the code is an explicit simplification, we should have  
> a very strong bias towards "don't merge".

Well, simplifications are welcome - this series itself was simplified beyond what I thought initially possible through the valuable comments of other people.

But of course, this adds more complexity to the kernel as a whole. And this is true to every single new feature we may add, now or in the future.

What I can tell you about this particular one, is that the justification for it doesn't come out of nowhere, but from a rather real use case that we support and maintain in OpenVZ and our line of products for years.

It can potentially achieve more than that by being well used by schemes such as systemd that aim at limiting the extent through which a service can damage a system.

I hope all the above justification was enough to clarify any points you may have. I'll be happy to go further if necessary.



Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 09:33:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>>  
>> Nothing, but I also don't see how to prevent that.  
>  
> You can test for current->flags & PF\_KTHREAD following the check for  
> in\_interrupt() and return true, it's what you were trying to do with the  
> check for !current->mm.  
>

am I right to believe that if not in interrupt context - already ruled out - and !(current->flags & PF\_KTHREAD), I am guaranteed to have a mm context, and thus, don't need to test against it ?

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 10:03:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/26/2012 08:09 AM, David Rientjes wrote:  
> @@ -2206,7 +2214,7 @@ static int mem\_cgroup\_do\_charge(struct mem\_cgroup \*memcg,  
gfp\_t gfp\_mask,  
>> \* unlikely to succeed so close to the limit, and we fall back  
>> \* to regular pages anyway in case of failure.  
>> \*/  
>>- if (nr\_pages == 1 && ret)  
>>+ if (nr\_pages <= NR\_PAGES\_TO\_RETRY && ret)  
>> return CHARGE\_RETRY;

Changed to costly order.

One more thing. The original version of this patch included a cond\_resched() here, that was also removed. From my re-reading of the code in page\_alloc.c and vmscan.c now, I tend to think this is indeed not needed, since any cond\_resched()s that might be needed to ensure the safety of the code will be properly inserted by the reclaim code itself, so there is no need for us to include any when we signal that a retry is needed.

Do you/others agree?

---

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 12:28:14 GMT



On 06/27/2012 04:29 PM, Frederic Weisbecker wrote:

> On Wed, Jun 27, 2012 at 01:29:04PM +0400, Glauber Costa wrote:

>> On 06/27/2012 01:55 AM, Andrew Morton wrote:

>>>> I can't speak for everybody here, but AFAIK, tracking the stack through  
>>>> the memory it used, therefore using my proposed kmem controller, was an  
>>>> idea that good quite a bit of traction with the memcg/memory people.  
>>>> So here you have something that people already asked a lot for, in a  
>>>> shape and interface that seem to be acceptable.

>>>

>>> mm, maybe. Kernel developers tend to look at code from the point of  
>>> view "does it work as designed", "is it clean", "is it efficient", "do  
>>> I understand it", etc. We often forget to step back and really  
>>> consider whether or not it should be merged at all.

>>>

>>> I mean, unless the code is an explicit simplification, we should have  
>>> a very strong bias towards "don't merge".

>>

>> Well, simplifications are welcome - this series itself was  
>> simplified beyond what I thought initially possible through the  
>> valuable comments  
>> of other people.

>>

>> But of course, this adds more complexity to the kernel as a whole.  
>> And this is true to every single new feature we may add, now or in  
>> the  
>> future.

>>

>> What I can tell you about this particular one, is that the justification  
>> for it doesn't come out of nowhere, but from a rather real use case that  
>> we support and maintain in OpenVZ and our line of products for years.

>

> Right and we really need a solution to protect against forkbombs in LXC.  
Small correction: In containers. LXC is not the only one out there =p

---

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem  
controller for memcg: stripped down ve

Posted by [Frederic Weisbecker](#) on Wed, 27 Jun 2012 12:29:27 GMT

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

---

On Wed, Jun 27, 2012 at 01:29:04PM +0400, Glauber Costa wrote:

> On 06/27/2012 01:55 AM, Andrew Morton wrote:

> >> I can't speak for everybody here, but AFAIK, tracking the stack through  
> >> the memory it used, therefore using my proposed kmem controller, was an  
> >> idea that good quite a bit of traction with the memcg/memory people.  
> >> So here you have something that people already asked a lot for, in a

> >>shape and interface that seem to be acceptable.  
> >  
> >mm, maybe. Kernel developers tend to look at code from the point of  
> >view "does it work as designed", "is it clean", "is it efficient", "do  
> >I understand it", etc. We often forget to step back and really  
> >consider whether or not it should be merged at all.  
> >  
> >I mean, unless the code is an explicit simplification, we should have  
> >a very strong bias towards "don't merge".  
>  
> Well, simplifications are welcome - this series itself was  
> simplified beyond what I thought initially possible through the  
> valuable comments  
> of other people.  
>  
> But of course, this adds more complexity to the kernel as a whole.  
> And this is true to every single new feature we may add, now or in  
> the  
> future.  
>  
> What I can tell you about this particular one, is that the justification  
> for it doesn't come out of nowhere, but from a rather real use case that  
> we support and maintain in OpenVZ and our line of products for years.

Right and we really need a solution to protect against forkbombs in LXC.  
The task counter was more simple but only useful for our usecase and  
defining the number of tasks as a resource was considered unnatural.

So limiting kernel stack allocations works for us. This patchset implements  
this so I'm happy with it. If this is more broadly useful by limiting  
resources others are interested in, that's even better. I doubt we are  
interested in a solution that only concerns kernel stack allocation.

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem  
controller for memcg: stripped down ve

Posted by [Frederic Weisbecker](#) on Wed, 27 Jun 2012 12:35:47 GMT

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

---

On Wed, Jun 27, 2012 at 04:28:14PM +0400, Glauber Costa wrote:

> On 06/27/2012 04:29 PM, Frederic Weisbecker wrote:  
> > On Wed, Jun 27, 2012 at 01:29:04PM +0400, Glauber Costa wrote:  
> >> On 06/27/2012 01:55 AM, Andrew Morton wrote:  
> >>> I can't speak for everybody here, but AFAIK, tracking the stack through  
> >>> the memory it used, therefore using my proposed kmem controller, was an  
> >>> idea that good quite a bit of traction with the memcg/memory people.  
> >>> So here you have something that people already asked a lot for, in a  
> >>> shape and interface that seem to be acceptable.

> >>>  
> >>> mm, maybe. Kernel developers tend to look at code from the point of  
> >>> view "does it work as designed", "is it clean", "is it efficient", "do  
> >>> I understand it", etc. We often forget to step back and really  
> >>> consider whether or not it should be merged at all.  
> >>>  
> >>> I mean, unless the code is an explicit simplification, we should have  
> >>> a very strong bias towards "don't merge".  
> >>  
> >> Well, simplifications are welcome - this series itself was  
> >> simplified beyond what I thought initially possible through the  
> >> valuable comments  
> >> of other people.  
> >>  
> >> But of course, this adds more complexity to the kernel as a whole.  
> >> And this is true to every single new feature we may add, now or in  
> >> the  
> >> future.  
> >>  
> >> What I can tell you about this particular one, is that the justification  
> >> for it doesn't come out of nowhere, but from a rather real use case that  
> >> we support and maintain in OpenVZ and our line of products for years.  
> >  
> > Right and we really need a solution to protect against forkbombs in LXC.  
> Small correction: In containers. LXC is not the only one out there =p

Sure. I was just speaking for the specific project I'm working on :)  
But I'm definitely interested in solutions that work for everyone in containers in  
general. And if Openvz is also interested in forkbombs protection that's even  
better.

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [Suleiman Souhlal](#) on Wed, 27 Jun 2012 16:16:51 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Jun 26, 2012 at 1:39 AM, Glauber Costa <glommer@parallels.com> wrote:  
> Yeah, forgot to update the changelog =(  
>  
> But much more importantly, are you still happy with those changes?

Yes, I am OK with those changes.

Thanks,  
-- Suleiman

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [David Rientjes](#) on Wed, 27 Jun 2012 19:38:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 27 Jun 2012, Glauber Costa wrote:

> fork bombs are a way bad behaved processes interfere with the rest of  
> the system. In here, I propose fork bomb stopping as a natural  
> consequence of the fact that the amount of kernel memory can be limited,  
> and each process uses 1 or 2 pages for the stack, that are freed when the  
> process goes away.  
>

The obvious disadvantage is that if you use the full-featured kmem controller that builds upon this patchset, then you're limiting the about of all kmem, not just the stack that this particular set limits. I hope you're not proposing it to go upstream before full support for the kmem controller is added so that users who use it only to protect again forkbombs soon realize that's no longer possible if your applications do any substantial slab allocations, particularly anything that does a lot of I/O.

In other words, if I want to run netperf in a memcg with the full-featured kmem controller enabled, then its kmem limit must be high enough so that it doesn't degrade performance that any limitation on stack allocations would be too high to effectively stop forkbombs.

---

Subject: Re: [PATCH 06/11] memcg: kmem controller infrastructure  
Posted by [David Rientjes](#) on Wed, 27 Jun 2012 19:46:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 27 Jun 2012, Glauber Costa wrote:

> > > Nothing, but I also don't see how to prevent that.  
> >  
> > You can test for current->flags & PF\_KTHREAD following the check for  
> > in\_interrupt() and return true, it's what you were trying to do with the  
> > check for !current->mm.  
> >  
>  
> am I right to believe that if not in interrupt context - already ruled out -  
> and !(current->flags & PF\_KTHREAD), I am guaranteed to have a mm context, and  
> thus, don't need to test against it ?  
>

No, because an mm may have been detached in the exit path by running

exit\_mm()). We'd certainly hope that there are no slab allocations following that point, though, but you'd still need to test current->mm.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [David Rientjes](#) on Wed, 27 Jun 2012 19:48:41 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 27 Jun 2012, Glauber Costa wrote:

```
> > @@ -2206,7 +2214,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup
> > *memcg, gfp_t gfp_mask,
> > > * unlikely to succeed so close to the limit, and we fall back
> > > * to regular pages anyway in case of failure.
> > > */
> > > - if (nr_pages == 1 && ret)
> > > + if (nr_pages <= NR_PAGES_TO_RETRY && ret)
> > > return CHARGE_RETRY;
>
> Changed to costly order.
>
```

1 << PAGE\_ALLOC\_COSTLY\_ORDER was the suggestion.

```
> One more thing. The original version of this patch included
> a cond_resched() here, that was also removed. From my re-reading
> of the code in page_alloc.c and vmscan.c now, I tend to think
> this is indeed not needed, since any cond_resched()s that might
> be needed to ensure the safety of the code will be properly
> inserted by the reclaim code itself, so there is no need for us
> to include any when we signal that a retry is needed.
>
```

For \_\_GFP\_WAIT, that sounds like a safe guarantee.

---

---

Subject: Re: [PATCH 02/11] memcg: Reclaim when more than one page needed.  
Posted by [Glauber Costa](#) on Wed, 27 Jun 2012 20:47:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/27/2012 11:48 PM, David Rientjes wrote:

> On Wed, 27 Jun 2012, Glauber Costa wrote:

```
>
>>> @@ -2206,7 +2214,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup
>>> *memcg, gfp_t gfp_mask,
>>>> * unlikely to succeed so close to the limit, and we fall back
>>>> * to regular pages anyway in case of failure.
```

```
>>>> */
>>>> - if (nr_pages == 1 && ret)
>>>> + if (nr_pages <= NR_PAGES_TO_RETRY && ret)
>>>> return CHARGE_RETRY;
>>
>> Changed to costly order.
>>
>
> 1 << PAGE_ALLOC_COSTLY_ORDER was the suggestion.
```

That is what I meant - to the costly order suggestion - , should have been more explicit.

```
>> One more thing. The original version of this patch included
>> a cond_resched() here, that was also removed. From my re-reading
>> of the code in page_alloc.c and vmscan.c now, I tend to think
>> this is indeed not needed, since any cond_resched()s that might
>> be needed to ensure the safety of the code will be properly
>> inserted by the reclaim code itself, so there is no need for us
>> to include any when we signal that a retry is needed.
>>
>
> For __GFP_WAIT, that sounds like a safe guarantee.
>
```

---

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve

Posted by [Glauber Costa](#) on Thu, 28 Jun 2012 09:01:23 GMT

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

---

On 06/27/2012 11:38 PM, David Rientjes wrote:

> On Wed, 27 Jun 2012, Glauber Costa wrote:

```
>
>> fork bombs are a way bad behaved processes interfere with the rest of
>> the system. In here, I propose fork bomb stopping as a natural
>> consequence of the fact that the amount of kernel memory can be limited,
>> and each process uses 1 or 2 pages for the stack, that are freed when the
>> process goes away.
```

```
>>
>
> The obvious disadvantage is that if you use the full-featured kmem
> controller that builds upon this patchset, then you're limiting the about
> of all kmem, not just the stack that this particular set limits. I hope
> you're not proposing it to go upstream before full support for the kmem
> controller is added so that users who use it only to protect again
> forkbombs soon realize that's no longer possible if your applications do
> any substantial slab allocations, particularly anything that does a lot of
```

> I/O.

Point by point:

1) This is not a disadvantage. The whole point of implementing it as `kmem`, not as a "fork controller" or anything in the like, was our understanding that people should not be "protecting against a x number of processes". Because this is unnatural. All you have is total of kernel memory, because that's what the interface gives you. If you can overuse memory, you can't fork bomb. But that's a consequence.

I'll grant that maybe it was a mistake of mine to try to sell it this way here, because it may give the wrong idea. In this case I welcome your comment because it will allow me to be more careful in future communications. But this was just me trying to show off the feature.

2) No admin should never, ever tune the system to a particular `kmem` limit value. And again, this is the whole point of not limiting "number of processes". I agree that adding slab tracking will raise kernel memory consumption by a reasonable amount. But reality is that even if this controller is totally stable, that not only can, but will happen.

Unlike user memory, where whoever writes the application control its behavior (forget about the libraries for a moment), users never really control the internals of the kernel. If you ever rely on the fact that you are currently using X Gb of `kmem`, and that should be enough, your setup will break when the data structure grows - as they do - when the kernel memory consumption rises naturally by algorithms - as it does, and so on.

3) Agreeing that of course, preventing disruption if we can help it is good, this feature is not only marked experimental, but default of. This was done precisely not to disrupt the amount of `*user*` memory used, where I actually think it makes a lot of sense ("My application allocates 4G, that's what I'll give it!"). Even if support is compiled in, it won't be until you start limiting it that anything will happen. Kernel Memory won't even be tracked until then. And I also understand that our use case here may be quite unique: We want the kernel memory limit to be way lower than the user limit (like 20 % - but that's still a percentage, not a tune!). When I discussed this around with other people, the vast majority of them wanted to set `kmem` = `umem`. Which basically means "user memory is the limit, but we want the kernel memory to be accounted as well".

So yes, although I understand your point - but not fully agree, I would like to get it merged as is. I don't believe adding slab memory later will be disruptive, in the same way I didn't believe adding stack later - in my original slab tracking patch - would be. As I originally stated,



this would allow me to greatly simplify the slab tracking patches, since we'll be able to focus on that only, instead of a gigantic patch that does a lot.

Other people have interest in this, so this would allow them to start building on it as well. That said, I won't oppose adding more code if you suggest so to make sure people rely less on the accidental fact that we're only tracking a part of what we will, even if they shouldn't.

- It's already experimental
- It's already default off.
- I could add a warn message first time it is set.
- I could add a boot option.

Or anything like this.

Last, but not least, note that it is totally within my interests to merge the slab tracking as fast as we can. it'll be a matter of going back to it, and agreeing in the final form.

>  
> In other words, if I want to run netperf in a memcg with the full-featured  
> kmem controller enabled, then its kmem limit must be high enough so that  
> it doesn't degrade performance that any limitation on stack allocations  
> would be too high to effectively stop forkbombs.  
>

That is a user setup problem. As I explained before, That's exactly what we want to discourage by exposing "kernel memory" instead of a particular tunable.

As it is, I agree, it will stop fork bombs but will take a lot more time for it than it should. But this only makes it analogous to the evolutionary precursors to the human eye: It's not perfect, but will achieve something that may be already of great value for a class of users. Improvements are certain to follow.

Thank you David!

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [akpm](#) on Thu, 28 Jun 2012 22:25:40 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu, 28 Jun 2012 13:01:23 +0400  
Glauber Costa <[glommer@parallels.com](mailto:glommer@parallels.com)> wrote:



>  
> ...  
>

OK, that all sounds convincing ;) Please summarise and capture this discussion in the [patch 0/n] changelog so we (or others) don't have to go through this all again. And let's remember this in the next patchset!

> Last, but not least, note that it is totally within my interests to  
> merge the slab tracking as fast as we can. it'll be a matter of going  
> back to it, and agreeing in the final form.

Yes, I'd very much like to have the whole slab implementation in a reasonably mature state before proceeding too far with this base patchset.

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [Glauber Costa](#) on Tue, 03 Jul 2012 11:38:39 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/29/2012 02:25 AM, Andrew Morton wrote:  
> On Thu, 28 Jun 2012 13:01:23 +0400  
> Glauber Costa <glommer@parallels.com> wrote:

>  
>>  
>> ...  
>>  
>  
> OK, that all sounds convincing ;) Please summarise and capture this  
> discussion in the [patch 0/n] changelog so we (or others) don't have to  
> go through this all again. And let's remember this in the next  
> patchset!

Thanks, will surely do.

>> Last, but not least, note that it is totally within my interests to  
>> merge the slab tracking as fast as we can. it'll be a matter of going  
>> back to it, and agreeing in the final form.  
>  
> Yes, I'd very much like to have the whole slab implementation in a  
> reasonably mature state before proceeding too far with this base  
> patchset.

Does that means that you want to merge them together? I am more than happy to post the slab part again ontop of that to have people reviewing it.

But if possible, I believe that merging this part first would help us to split up testing in a beneficial way, in the sense that if it breaks, we know at least in which part it is. Not to mention, of course, that reviewers will have an easier time reviewing it as two pieces.

---

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve

Posted by [Frederic Weisbecker](#) on Thu, 12 Jul 2012 15:40:11 GMT

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

---

On Tue, Jul 03, 2012 at 03:38:39PM +0400, Glauber Costa wrote:

> On 06/29/2012 02:25 AM, Andrew Morton wrote:

> > On Thu, 28 Jun 2012 13:01:23 +0400

> > Glauber Costa <glommer@parallels.com> wrote:

> >

> >>

> >> ...

> >>

> >

> > OK, that all sounds convincing ;) Please summarise and capture this

> > discussion in the [patch 0/n] changelog so we (or others) don't have to

> > go through this all again. And let's remember this in the next

> > patchset!

>

> Thanks, will surely do.

>

> >> Last, but not least, note that it is totally within my interests to

> >> merge the slab tracking as fast as we can. it'll be a matter of going

> >> back to it, and agreeing in the final form.

> >

> > Yes, I'd very much like to have the whole slab implementation in a

> > reasonably mature state before proceeding too far with this base

> > patchset.

>

> Does that means that you want to merge them together? I am more than

> happy to post the slab part again ontop of that to have people reviewing it.

>

> But if possible, I believe that merging this part first would help us to

> split up testing in a beneficial way, in the sense that if it breaks, we

> know at least in which part it is. Not to mention, of course, that

> reviewers will have an easier time reviewing it as two pieces.

Definetly yeah. This makes the review easier for this tricky chunk.

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [Glauber Costa](#) on Tue, 07 Aug 2012 13:59:25 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 06/29/2012 02:25 AM, Andrew Morton wrote:  
> On Thu, 28 Jun 2012 13:01:23 +0400  
> Glauber Costa <glommer@parallels.com> wrote:  
>  
>>  
>> ...  
>>  
>  
> OK, that all sounds convincing ;) Please summarise and capture this  
> discussion in the [patch 0/n] changelog so we (or others) don't have to  
> go through this all again. And let's remember this in the next  
> patchset!  
>  
>> Last, but not least, note that it is totally within my interests to  
>> merge the slab tracking as fast as we can. it'll be a matter of going  
>> back to it, and agreeing in the final form.  
>  
> Yes, I'd very much like to have the whole slab implementation in a  
> reasonably mature state before proceeding too far with this base  
> patchset.  
>  
So, that was posted separately as well.

Although there is a thing to fix here and there - all of them I am working on already - I believe that to be mature enough.

Do you have any comments on that? Would you be willing to take this first part (modified with the comments on this thread itself) and let it start sitting in the tree?

Thanks!

---

---

Subject: Re: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down ve  
Posted by [Glauber Costa](#) on Wed, 08 Aug 2012 14:15:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 08/07/2012 05:59 PM, Glauber Costa wrote:  
> On 06/29/2012 02:25 AM, Andrew Morton wrote:  
>> On Thu, 28 Jun 2012 13:01:23 +0400  
>> Glauber Costa <glommer@parallels.com> wrote:  
>>

>>>  
>>> ...  
>>>  
>>  
>> OK, that all sounds convincing ;) Please summarise and capture this  
>> discussion in the [patch 0/n] changelog so we (or others) don't have to  
>> go through this all again. And let's remember this in the next  
>> patchset!  
>>  
>>> Last, but not least, note that it is totally within my interests to  
>>> merge the slab tracking as fast as we can. it'll be a matter of going  
>>> back to it, and agreeing in the final form.  
>>  
>> Yes, I'd very much like to have the whole slab implementation in a  
>> reasonably mature state before proceeding too far with this base  
>> patchset.  
>>  
> So, that was posted separately as well.  
>  
> Although there is a thing to fix here and there - all of them I am  
> working on already - I believe that to be mature enough.  
>  
> Do you have any comments on that? Would you be willing to take this  
> first part (modified with the comments on this thread itself) and let it  
> start sitting in the tree?  
>

In the mean time, for any interested parties, I've set up a tree at:

[git://github.com/glommer/linux.git](https://github.com/glommer/linux.git)

branches kmemcg-slab and kmemcg-stack

Intended to be a throw-away tree.

---