
Subject: [-mm PATCH 4/8] Memory controller memory accounting (v2)

Posted by [Balbir Singh](#) on Fri, 06 Jul 2007 05:21:35 GMT

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

Add the accounting hooks. The accounting is carried out for RSS and Page Cache (unmapped) pages. There is now a common limit and accounting for both. The RSS accounting is accounted at `page_add_*_rmap()` and `page_remove_rmap()` time. Page cache is accounted at `add_to_page_cache()`, `__delete_from_page_cache()`. Swap cache is also accounted for.

Each page's `meta_page` is protected with a bit in page flags, this makes handling of race conditions involving simultaneous mappings of a page easier. A reference count is kept in the `meta_page` to deal with cases where a page might be unmapped from the RSS of all tasks, but still lives in the page cache.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

```
fs/exec.c          | 1
include/linux/memcontrol.h | 11 +++
include/linux/page-flags.h | 3 +
mm/filemap.c       | 8 ++
mm/memcontrol.c    | 132 ++++++-----
mm/memory.c        | 22 ++++++
mm/migrate.c       | 6 ++
mm/page_alloc.c    | 3 +
mm/rmap.c          | 2
mm/swap_state.c    | 8 ++
mm/swapfile.c      | 40 ++++++-----
11 files changed, 218 insertions(+), 18 deletions(-)
```

diff -puN fs/exec.c~mem-control-accounting fs/exec.c

--- linux-2.6.22-rc6/fs/exec.c~mem-control-accounting 2007-07-05 13:45:18.000000000 -0700

+++ linux-2.6.22-rc6-balbir/fs/exec.c 2007-07-05 13:45:18.000000000 -0700

@@ -51,6 +51,7 @@

#include <linux/cn_proc.h>

#include <linux/audit.h>

#include <linux/signalfd.h>

+#include <linux/memcontrol.h>

#include <asm/uaccess.h>

#include <asm/mmu_context.h>

diff -puN include/linux/memcontrol.h~mem-control-accounting include/linux/memcontrol.h

--- linux-2.6.22-rc6/include/linux/memcontrol.h~mem-control-accounting 2007-07-05

13:45:18.000000000 -0700

+++ linux-2.6.22-rc6-balbir/include/linux/memcontrol.h 2007-07-05 18:27:26.000000000 -0700

@@ -24,6 +24,8 @@ extern void mm_init_container(struct mm_

```

extern void mm_free_container(struct mm_struct *mm);
extern void page_assign_meta_page(struct page *page, struct meta_page *mp);
extern struct meta_page *page_get_meta_page(struct page *page);
+extern int mem_container_charge(struct page *page, struct mm_struct *mm);
+extern void mem_container_uncharge(struct meta_page *mp);

#else /* CONFIG_CONTAINER_MEM_CONT */
static inline void mm_init_container(struct mm_struct *mm,
@@ -45,6 +47,15 @@ static inline struct meta_page *page_get
return NULL;
}

+static inline int mem_container_charge(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void mem_container_uncharge(struct meta_page *mp)
+{
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/page-flags.h~mem-control-accounting include/linux/page-flags.h
--- linux-2.6.22-rc6/include/linux/page-flags.h~mem-control-accounting 2007-07-05
13:45:18.000000000 -0700
+++ linux-2.6.22-rc6-balbir/include/linux/page-flags.h 2007-07-05 13:45:18.000000000 -0700
@@ -98,6 +98,9 @@
#define PG_checked PG_owner_priv_1 /* Used by some filesystems */
#define PG_pinned PG_owner_priv_1 /* Xen pinned pagetable */

+#define PG_metapage 21 /* Used for checking if a meta_page */
+ /* is associated with a page */
+
#if (BITS_PER_LONG > 32)
/*
* 64-bit-only flags build down from bit 31
diff -puN mm/filemap.c~mem-control-accounting mm/filemap.c
--- linux-2.6.22-rc6/mm/filemap.c~mem-control-accounting 2007-07-05 13:45:18.000000000 -0700
+++ linux-2.6.22-rc6-balbir/mm/filemap.c 2007-07-05 18:26:29.000000000 -0700
@@ -31,6 +31,7 @@
#include <linux/syscalls.h>
#include <linux/cpuset.h>
#include <linux/hardirq.h> /* for BUG_ON(!in_atomic()) only */
+#include <linux/memcontrol.h>
#include "internal.h"

```

```

/*
@@ -116,6 +117,7 @@ void __remove_from_page_cache(struct pag
{
    struct address_space *mapping = page->mapping;

+ mem_container_uncharge(page_get_meta_page(page));
    radix_tree_delete(&mapping->page_tree, page->index);
    page->mapping = NULL;
    mapping->numpages--;
@@ -442,6 +444,11 @@ int add_to_page_cache(struct page *page,
    int error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);

    if (error == 0) {
+
+ error = mem_container_charge(page, current->mm);
+ if (error)
+ goto out;
+
        write_lock_irq(&mapping->tree_lock);
        error = radix_tree_insert(&mapping->page_tree, offset, page);
        if (!error) {
@@ -455,6 +462,7 @@ int add_to_page_cache(struct page *page,
        write_unlock_irq(&mapping->tree_lock);
        radix_tree_preload_end();
    }
+out:
    return error;
}
EXPORT_SYMBOL(add_to_page_cache);
diff -puN mm/memcontrol.c~mem-control-accounting mm/memcontrol.c
--- linux-2.6.22-rc6/mm/memcontrol.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700
+++ linux-2.6.22-rc6-balbir/mm/memcontrol.c 2007-07-05 18:27:29.000000000 -0700
@@ -16,6 +16,9 @@
#include <linux/memcontrol.h>
#include <linux/container.h>
#include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/bit_spinlock.h>
+#include <linux/rcupdate.h>

struct container_subsys mem_container_subsys;

@@ -26,7 +29,9 @@ struct container_subsys mem_container_su
* to help the administrator determine what knobs to tune.
*
* TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark.

```

```
+ * we hit the water mark. May be even add a low water mark, such that
+ * no reclaim occurs from a container at it's low water mark, this is
+ * a feature that will be implemented much later in the future.
```

```
*/
struct mem_container {
    struct container_subsys_state css;
@@ -51,6 +56,7 @@ struct meta_page {
    struct list_head list; /* per container LRU list */
    struct page *page;
    struct mem_container *mem_container;
+ atomic_t ref_cnt;
};
```

```
@@ -87,6 +93,128 @@ struct meta_page *page_get_meta_page(str
    return page->meta_page;
}
```

```
+void __always_inline lock_meta_page(struct page *page)
+{
+ bit_spin_lock(PG_metapage, &page->flags);
+}
+
+void __always_inline unlock_meta_page(struct page *page)
+{
+ bit_spin_unlock(PG_metapage, &page->flags);
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the container is over its limit
+ */
+int mem_container_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_container *mem;
+ struct meta_page *mp;
+
+ /*
+ * Should meta_page's go to their own slab?
+ * One could optimize the performance of the charging routine
+ * by saving a bit in the page_flags and using it as a lock
+ * to see if the container page already has a meta_page associated
+ * with it
+ */
+ lock_meta_page(page);
+ mp = page_get_meta_page(page);
```

```

+ /*
+  * The meta_page exists and the page has already been accounted
+  */
+ if (mp) {
+     atomic_inc(&mp->ref_cnt);
+     goto done;
+ }
+
+ unlock_meta_page(page);
+
+ mp = kzalloc(sizeof(struct meta_page), GFP_KERNEL);
+ if (mp == NULL)
+     goto err;
+
+ rcu_read_lock();
+ /*
+  * We always charge the container the mm_struct belongs to
+  * the mm_struct's mem_container changes on task migration if the
+  * thread group leader migrates. It's possible that mm is not
+  * set, if so charge the init_mm (happens for pagecache usage).
+  */
+ if (!mm)
+     mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_container);
+ /*
+  * For every charge from the container, increment reference
+  * count
+  */
+ css_get(&mem->css);
+ rcu_read_unlock();
+
+ /*
+  * If we created the meta_page, we should free it on exceeding
+  * the container limit.
+  */
+ if (res_counter_charge(&mem->res, 1))
+     goto free_mp;
+
+ lock_meta_page(page);
+ /*
+  * Check if somebody else beat us to allocating the meta_page
+  */
+ if (page_get_meta_page(page)) {
+     atomic_inc(&mp->ref_cnt);
+     res_counter_uncharge(&mem->res, 1);
+     goto done;
+ }

```

```

+
+ atomic_set(&mp->ref_cnt, 1);
+ mp->mem_container = mem;
+ mp->page = page;
+ page_assign_meta_page(page, mp);
+
+done:
+ unlock_meta_page(page);
+ return 0;
+free_mp:
+ kfree(mp);
+ return -ENOMEM;
+err:
+ unlock_meta_page(page);
+ return -ENOMEM;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_container_uncharge(struct meta_page *mp)
+{
+ struct mem_container *mem;
+ struct page *page;
+
+ /*
+ * This can happen for PAGE_ZERO
+ */
+ if (!mp)
+ return;
+
+ if (atomic_dec_and_test(&mp->ref_cnt)) {
+ page = mp->page;
+ lock_meta_page(page);
+ mem = mp->mem_container;
+ css_put(&mem->css);
+ page_assign_meta_page(page, NULL);
+ unlock_meta_page(page);
+ res_counter_uncharge(&mem->res, 1);
+ kfree(mp);
+ }
+}
+
+static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
+ struct file *file, char __user *userbuf, size_t nbytes,
+ loff_t *ppos)
@@ -142,6 +270,8 @@ static int mem_container_create(struct c

```

```

    res_counter_init(&mem->res);
    cont->subsys[mem_container_subsys_id] = &mem->css;
    mem->css.container = cont;
+ INIT_LIST_HEAD(&mem->active_list);
+ INIT_LIST_HEAD(&mem->inactive_list);
    return 0;
}

```

```

diff -puN mm/memory.c~mem-control-accounting mm/memory.c
--- linux-2.6.22-rc6/mm/memory.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700

```

```

+++ linux-2.6.22-rc6-balbir/mm/memory.c 2007-07-05 13:45:18.000000000 -0700

```

```

@@ -50,6 +50,7 @@

```

```

#include <linux/delayacct.h>

```

```

#include <linux/init.h>

```

```

#include <linux/writeback.h>

```

```

+#include <linux/memcontrol.h>

```

```

#include <asm/pgalloc.h>

```

```

#include <asm/uaccess.h>

```

```

@@ -1226,6 +1227,10 @@ static int insert_page(struct mm_struct
    pte_t *pte;
    spinlock_t *ptl;

```

```

+ retval = mem_container_charge(page, mm);

```

```

+ if (retval)

```

```

+ goto out;

```

```

+

```

```

    retval = -EINVAL;

```

```

    if (PageAnon(page))

```

```

        goto out;

```

```

@@ -1731,6 +1736,9 @@ gotten:

```

```

    cow_user_page(new_page, old_page, address, vma);

```

```

}

```

```

+ if (mem_container_charge(new_page, mm))

```

```

+ goto oom;

```

```

+

```

```

/*

```

```

 * Re-check the pte - we dropped the lock

```

```

*/

```

```

@@ -2188,6 +2196,11 @@ static int do_swap_page(struct mm_struct

```

```

}

```

```

    delayacct_clear_flag(DELAYACCT_PF_SWAPIN);

```

```

+ if (mem_container_charge(page, mm)) {

```

```

+ ret = VM_FAULT_OOM;

```

```

+ goto out;

```

```

+ }
+
mark_page_accessed(page);
lock_page(page);

@@ -2255,6 +2268,7 @@ static int do_anonymous_page(struct mm_s
pte_t entry;

if (write_access) {
+
/* Allocate our own private page. */
pte_unmap(page_table);

@@ -2264,6 +2278,9 @@ static int do_anonymous_page(struct mm_s
if (!page)
goto oom;

+ if (mem_container_charge(page, mm))
+ goto oom;
+
entry = mk_pte(page, vma->vm_page_prot);
entry = maybe_mkdirty(entry, vma);

@@ -2397,6 +2414,11 @@ static int __do_fault(struct mm_struct *

}

+ if (mem_container_charge(page, mm)) {
+ fdata.type = VM_FAULT_OOM;
+ goto out;
+ }
+
page_table = pte_offset_map_lock(mm, pmd, address, &ptl);

/*
diff -puN mm/migrate.c~mem-control-accounting mm/migrate.c
--- linux-2.6.22-rc6/mm/migrate.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700
+++ linux-2.6.22-rc6-balbir/mm/migrate.c 2007-07-05 13:45:18.000000000 -0700
@@ -28,6 +28,7 @@
#include <linux/mempolicy.h>
#include <linux/vmalloc.h>
#include <linux/security.h>
+#include <linux/memcontrol.h>

#include "internal.h"

@@ -157,6 +158,11 @@ static void remove_migration_pte(struct

```



```

    return;
}

+ if (mem_container_charge(new, mm)) {
+ pte_unmap(pte);
+ return;
+ }
+
    pte = pte_lockptr(mm, pmd);
    spin_lock(pte);
    pte = *pte;
diff -puN mm/page_alloc.c~mem-control-accounting mm/page_alloc.c
--- linux-2.6.22-rc6/mm/page_alloc.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700
+++ linux-2.6.22-rc6-balbir/mm/page_alloc.c 2007-07-05 13:45:18.000000000 -0700
@@ -41,6 +41,7 @@
#include <linux/pfn.h>
#include <linux/backing-dev.h>
#include <linux/fault-inject.h>
+#include <linux/memcontrol.h>

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -1015,6 +1016,7 @@ static void fastcall free_hot_cold_page(

    if (!PageHighMem(page))
        debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+ page_assign_meta_page(page, NULL);
    arch_free_page(page, 0);
    kernel_map_pages(page, 1, 0);

@@ -2576,6 +2578,7 @@ void __meminit memmap_init_zone(unsigned
    set_page_links(page, zone, nid, pfn);
    init_page_count(page);
    reset_page_mapcount(page);
+ page_assign_meta_page(page, NULL);
    SetPageReserved(page);

/*
diff -puN mm/rmap.c~mem-control-accounting mm/rmap.c
--- linux-2.6.22-rc6/mm/rmap.c~mem-control-accounting 2007-07-05 13:45:18.000000000 -0700
+++ linux-2.6.22-rc6-balbir/mm/rmap.c 2007-07-05 13:45:18.000000000 -0700
@@ -643,6 +643,8 @@ void page_remove_rmap(struct page *page,
    page_clear_dirty(page);
    set_page_dirty(page);
}
+
+ mem_container_uncharge(page_get_meta_page(page));

```

```

    __dec_zone_page_state(page,
    PageAnon(page) ? NR_ANON_PAGES : NR_FILE_MAPPED);
}
diff -puN mm/swapfile.c~mem-control-accounting mm/swapfile.c
--- linux-2.6.22-rc6/mm/swapfile.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700
+++ linux-2.6.22-rc6-balbir/mm/swapfile.c 2007-07-05 13:45:18.000000000 -0700
@@ -506,9 +506,12 @@ unsigned int count_swap_pages(int type,
 * just let do_wp_page work it out if a write is requested later - to
 * force COW, vm_page_prot omits write permission from any private vma.
 */
-static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
    unsigned long addr, swp_entry_t entry, struct page *page)
{
+ if (mem_container_charge(page, vma->vm_mm))
+ return -ENOMEM;
+
    inc_mm_counter(vma->vm_mm, anon_rss);
    get_page(page);
    set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +523,7 @@ static void unuse_pte(struct vm_area_str
 * immediately swapped out again after swapon.
 */
    activate_page(page);
+ return 1;
}

static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -529,7 +533,7 @@ static int unuse_pte_range(struct vm_are
    pte_t swp_pte = swp_entry_to_pte(entry);
    pte_t *pte;
    spinlock_t *ptl;
- int found = 0;
+ int ret = 0;

    pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
    do {
@@ -538,13 +542,12 @@ static int unuse_pte_range(struct vm_are
 * Test inline before going to call unuse_pte.
 */
    if (unlikely(pte_same(*pte, swp_pte))) {
- unuse_pte(vma, pte++, addr, entry, page);
- found = 1;
+ ret = unuse_pte(vma, pte++, addr, entry, page);
+ break;
    }
} while (pte++, addr += PAGE_SIZE, addr != end);

```

```

    pte_unmap_unlock(pte - 1, ptl);
- return found;
+ return ret;
}

static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
@@ -553,14 +556,16 @@ static inline int unuse_pmd_range(struct
{
    pmd_t *pmd;
    unsigned long next;
+ int ret;

    pmd = pmd_offset(pud, addr);
    do {
        next = pmd_addr_end(addr, end);
        if (pmd_none_or_clear_bad(pmd))
            continue;
- if (unuse_pte_range(vma, pmd, addr, next, entry, page))
- return 1;
+ ret = unuse_pte_range(vma, pmd, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pmd++, addr = next, addr != end);
    return 0;
}
@@ -571,14 +576,16 @@ static inline int unuse_pud_range(struct
{
    pud_t *pud;
    unsigned long next;
+ int ret;

    pud = pud_offset(pgd, addr);
    do {
        next = pud_addr_end(addr, end);
        if (pud_none_or_clear_bad(pud))
            continue;
- if (unuse_pmd_range(vma, pud, addr, next, entry, page))
- return 1;
+ ret = unuse_pmd_range(vma, pud, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pud++, addr = next, addr != end);
    return 0;
}
@@ -588,6 +595,7 @@ static int unuse_vma(struct vm_area_stru
{
    pgd_t *pgd;
    unsigned long addr, end, next;

```

```

+ int ret;

    if (page->mapping) {
        addr = page_address_in_vma(page, vma);
@@ -605,8 +613,9 @@ static int unuse_vma(struct vm_area_stru
        next = pgd_addr_end(addr, end);
        if (pgd_none_or_clear_bad(pgd))
            continue;
- if (unuse_pud_range(vma, pgd, addr, next, entry, page))
- return 1;
+ ret = unuse_pud_range(vma, pgd, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pgd++, addr = next, addr != end);
    return 0;
}
@@ -615,6 +624,7 @@ static int unuse_mm(struct mm_struct *mm
    swp_entry_t entry, struct page *page)
{
    struct vm_area_struct *vma;
+ int ret = 0;

    if (!down_read_trylock(&mm->mmap_sem)) {
/*
@@ -627,15 +637,11 @@ static int unuse_mm(struct mm_struct *mm
    lock_page(page);
    }
    for (vma = mm->mmap; vma; vma = vma->vm_next) {
- if (vma->anon_vma && unuse_vma(vma, entry, page))
+ if (vma->anon_vma && (ret = unuse_vma(vma, entry, page)))
        break;
    }
    up_read(&mm->mmap_sem);
- /*
- * Currently unuse_mm cannot fail, but leave error handling
- * at call sites for now, since we change it from time to time.
- */
- return 0;
+ return ret;
    }

/*
diff -puN mm/swap_state.c~mem-control-accounting mm/swap_state.c
--- linux-2.6.22-rc6/mm/swap_state.c~mem-control-accounting 2007-07-05 13:45:18.000000000
-0700
+++ linux-2.6.22-rc6-balbir/mm/swap_state.c 2007-07-05 18:28:08.000000000 -0700
@@ -17,6 +17,7 @@
#include <linux/backing-dev.h>

```

```

#include <linux/pagevec.h>
#include <linux/migrate.h>
#include <linux/memcontrol.h>

#include <asm/pgtable.h>

@@ -79,6 +80,11 @@ static int __add_to_swap_cache(struct pa
    BUG_ON(PagePrivate(page));
    error = radix_tree_preload(gfp_mask);
    if (!error) {
+
+ error = mem_container_charge(page, current->mm);
+ if (error)
+ goto out;
+
    write_lock_irq(&swapper_space.tree_lock);
    error = radix_tree_insert(&swapper_space.page_tree,
        entry.val, page);
@@ -93,6 +99,7 @@ static int __add_to_swap_cache(struct pa
    write_unlock_irq(&swapper_space.tree_lock);
    radix_tree_preload_end();
    }
+out:
    return error;
    }

@@ -129,6 +136,7 @@ void __delete_from_swap_cache(struct pag
    BUG_ON(PageWriteback(page));
    BUG_ON(PagePrivate(page));

+ mem_container_uncharge(page_get_meta_page(page));
    radix_tree_delete(&swapper_space.page_tree, page_private(page));
    set_page_private(page, 0);
    ClearPageSwapCache(page);

```

—

--

Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

Containers mailing list
 Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
